
Brian M. Adams, Mohamed S. Ebeida, Michael S. Eldred, John D. Jakeman, Laura P. Swiler, J. Adam Stephens, Dena M. Vigil, Timothy M. Wildey
Optimization and Uncertainty Quantification Department

William J. Bohnhoff
Radiation Transport Department

Keith R. Dalbey
Mission Analysis and Simulation Department

John P. Eddy
System Readiness and Sustainment Technologies Department

Kenneth T. Hu
Validation and Uncertainty Quantification Department

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

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

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

This report describes the Dakota class hierarchies. It is derived from annotation of the source code and provides detailed class documentation, including all member functions and attributes.
## Contents

1 Dakota Developers Manual 11  
1.1 Introduction 11  
1.2 Overview of Dakota 11  
1.3 Services 16  
1.4 Development Practices and Guidance 16  
1.5 Additional Resources 17  

2 Coding Style Guidelines and Conventions 19  
2.1 Introduction 19  
2.2 C++/c Style Guidelines 19  
2.3 File Naming Conventions 21  
2.4 Class Documentation Conventions 22  
2.5 CMake Style Guidelines 22  

3 Instructions for Modifying Dakota’s Input Specification 25  
3.1 Modify dakota.input.nspec 25  
3.2 Rebuild generated files 26  
3.3 Update NIDRProblemDescDB.cpp in Dakota/src 26  
3.4 Update ProblemDescDB.cpp in Dakota/src 27  
3.5 Update Corresponding Data Classes 28  
3.6 Use get::<data_type>() Functions 29  
3.7 Update the Documentation 29  

4 Understanding Iterator Flow 31  

5 Interfacing with Dakota as a Library 33  
5.1 Introduction 33  
5.2 Basic Dakota library instantiation 34  
5.3 Configuring Dakota operation 35  
5.4 Creating a simulator plugin interface 38  
5.5 Retrieving data after a run 41  
5.6 Linking against the Dakota library 41  

6 Performing Function Evaluations 43  
6.1 Synchronous function evaluations 43  
6.2 Asynchronous function evaluations 43  
6.3 Analyses within each function evaluation 44
## Working with Variable Containers and Views

- **7.1 Storage in Variables**
- **7.2 Storage in SharedVariablesData**
- **7.3 Active and inactive views**

## Namespace Index

- **8.1 Namespace List**

## Hierarchical Index

- **9.1 Class Hierarchy**

## Class Index

- **10.1 Class List**

## File Index

- **11.1 File List**

## Namespace Documentation

- **12.1 Dakota Namespace Reference**
- **12.2 SIM Namespace Reference**

## Class Documentation

- **13.1 ActiveSet Class Reference**
- **13.2 Analyzer Class Reference**
- **13.3 ApplicationInterface Class Reference**
- **13.4 Approximation Class Reference**
- **13.5 ApproximationInterface Class Reference**
- **13.6 APPSEvalMgr Class Reference**
- **13.7 APPSOptimizer Class Reference**
- **13.8 BaseConstructor Struct Reference**
- **13.9 callback\_data Struct Reference**
- **13.10 COLINApplication Class Reference**
- **13.11 COLINOptimizer Class Reference**
- **13.12 CollabHybridMetaIterator Class Reference**
- **13.13 CommandLineHandler Class Reference**
- **13.14 CommandShell Class Reference**
- **13.15 ConcurrentMetaIterator Class Reference**
- **13.16 CONMINOptimizer Class Reference**
- **13.17 Constraints Class Reference**
- **13.18 DataEnvironment Class Reference**
- **13.19 DataEnvironmentRep Class Reference**
- **13.20 DataFitSurrModel Class Reference**
- **13.21 DataInterface Class Reference**
- **13.22 DataMethod Class Reference**
- **13.23 DataMethodRep Class Reference**
- **13.24 DataModel Class Reference**
- **13.25 DataModelRep Class Reference**
- **13.26 DataResponses Class Reference**
- **13.27 DataResponsesRep Class Reference**
- **13.28 DataVariables Class Reference**
<table>
<thead>
<tr>
<th>Section</th>
<th>Class Reference</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.29</td>
<td>DataVariablesRep Class Reference</td>
<td>391</td>
</tr>
<tr>
<td>13.30</td>
<td>DDAEDesignCompExp Class Reference</td>
<td>402</td>
</tr>
<tr>
<td>13.31</td>
<td>DirectApplicInterface Class Reference</td>
<td>406</td>
</tr>
<tr>
<td>13.32</td>
<td>DiscrepancyCorrection Class Reference</td>
<td>410</td>
</tr>
<tr>
<td>13.33</td>
<td>DOTOptimizer Class Reference</td>
<td>413</td>
</tr>
<tr>
<td>13.34</td>
<td>JEGAOptimizer::Driver Class Reference</td>
<td>417</td>
</tr>
<tr>
<td>13.35</td>
<td>EffGlobalMinimizer Class Reference</td>
<td>419</td>
</tr>
<tr>
<td>13.36</td>
<td>EfficientSubspaceMethod Class Reference</td>
<td>421</td>
</tr>
<tr>
<td>13.37</td>
<td>EmbedHybridMetaIterator Class Reference</td>
<td>425</td>
</tr>
<tr>
<td>13.38</td>
<td>Environment Class Reference</td>
<td>426</td>
</tr>
<tr>
<td>13.39</td>
<td>NomadOptimizer::Evaluator Class Reference</td>
<td>430</td>
</tr>
<tr>
<td>13.40</td>
<td>JEGAOptimizer::Evaluator Class Reference</td>
<td>432</td>
</tr>
<tr>
<td>13.41</td>
<td>JEGAOptimizer::EvaluatorCreator Class Reference</td>
<td>437</td>
</tr>
<tr>
<td>13.42</td>
<td>ExecutableEnvironment Class Reference</td>
<td>438</td>
</tr>
<tr>
<td>13.43</td>
<td>ExperimentData Class Reference</td>
<td>439</td>
</tr>
<tr>
<td>13.44</td>
<td>ForkApplicInterface Class Reference</td>
<td>440</td>
</tr>
<tr>
<td>13.45</td>
<td>FSUDesignCompExp Class Reference</td>
<td>442</td>
</tr>
<tr>
<td>13.46</td>
<td>GaussProcApproximation Class Reference</td>
<td>445</td>
</tr>
<tr>
<td>13.47</td>
<td>GetLongOpt Class Reference</td>
<td>450</td>
</tr>
<tr>
<td>13.48</td>
<td>Graphics Class Reference</td>
<td>453</td>
</tr>
<tr>
<td>13.49</td>
<td>GridApplicInterface Class Reference</td>
<td>456</td>
</tr>
<tr>
<td>13.50</td>
<td>HierarchSurrModel Class Reference</td>
<td>457</td>
</tr>
<tr>
<td>13.51</td>
<td>Interface Class Reference</td>
<td>461</td>
</tr>
<tr>
<td>13.52</td>
<td>Iterator Class Reference</td>
<td>470</td>
</tr>
<tr>
<td>13.53</td>
<td>IteratorScheduler Class Reference</td>
<td>483</td>
</tr>
<tr>
<td>13.54</td>
<td>JEGAOptimizer Class Reference</td>
<td>488</td>
</tr>
<tr>
<td>13.55</td>
<td>LeastSq Class Reference</td>
<td>494</td>
</tr>
<tr>
<td>13.56</td>
<td>LibraryEnvironment Class Reference</td>
<td>498</td>
</tr>
<tr>
<td>13.57</td>
<td>MatlabInterface Class Reference</td>
<td>500</td>
</tr>
<tr>
<td>13.58</td>
<td>MetaIterator Class Reference</td>
<td>502</td>
</tr>
<tr>
<td>13.59</td>
<td>Minimizer Class Reference</td>
<td>503</td>
</tr>
<tr>
<td>13.60</td>
<td>MixedVarConstraints Class Reference</td>
<td>517</td>
</tr>
<tr>
<td>13.61</td>
<td>MixedVariables Class Reference</td>
<td>518</td>
</tr>
<tr>
<td>13.62</td>
<td>Model Class Reference</td>
<td>520</td>
</tr>
<tr>
<td>13.63</td>
<td>MPIManager Class Reference</td>
<td>548</td>
</tr>
<tr>
<td>13.64</td>
<td>MPIPackBuffer Class Reference</td>
<td>549</td>
</tr>
<tr>
<td>13.65</td>
<td>MPIUnpackBuffer Class Reference</td>
<td>551</td>
</tr>
<tr>
<td>13.66</td>
<td>NCSUOptimizer Class Reference</td>
<td>553</td>
</tr>
<tr>
<td>13.67</td>
<td>NestedModel Class Reference</td>
<td>556</td>
</tr>
<tr>
<td>13.68</td>
<td>NIDRProblemDescDB Class Reference</td>
<td>564</td>
</tr>
<tr>
<td>13.69</td>
<td>NL2Res Struct Reference</td>
<td>568</td>
</tr>
<tr>
<td>13.70</td>
<td>NL2SOLLeastSq Class Reference</td>
<td>568</td>
</tr>
<tr>
<td>13.71</td>
<td>NLQLOptimizer Class Reference</td>
<td>571</td>
</tr>
<tr>
<td>13.72</td>
<td>NLSSOLLeastSq Class Reference</td>
<td>575</td>
</tr>
<tr>
<td>13.73</td>
<td>NoDBaseConstructor Struct Reference</td>
<td>577</td>
</tr>
<tr>
<td>13.74</td>
<td>NomadOptimizer Class Reference</td>
<td>578</td>
</tr>
<tr>
<td>13.75</td>
<td>NonD Class Reference</td>
<td>580</td>
</tr>
<tr>
<td>13.76</td>
<td>NonDAdaptImpSampling Class Reference</td>
<td>590</td>
</tr>
<tr>
<td>13.77</td>
<td>NonDAdaptiveSampling Class Reference</td>
<td>594</td>
</tr>
<tr>
<td>Section</td>
<td>Class Reference</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>---------------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>13.78</td>
<td>NonDBayesCalibration Class Reference</td>
<td>598</td>
</tr>
<tr>
<td>13.79</td>
<td>NonDCalibration Class Reference</td>
<td>600</td>
</tr>
<tr>
<td>13.80</td>
<td>NonDCubature Class Reference</td>
<td>603</td>
</tr>
<tr>
<td>13.81</td>
<td>NonDDREAMBayesCalibration Class Reference</td>
<td>605</td>
</tr>
<tr>
<td>13.82</td>
<td>NonDExpansion Class Reference</td>
<td>609</td>
</tr>
<tr>
<td>13.83</td>
<td>NonDGlobalEvidence Class Reference</td>
<td>615</td>
</tr>
<tr>
<td>13.84</td>
<td>NonDGlobalInterval Class Reference</td>
<td>617</td>
</tr>
<tr>
<td>13.85</td>
<td>NonDGlobalReliability Class Reference</td>
<td>620</td>
</tr>
<tr>
<td>13.86</td>
<td>NonDGlobalSingleInterval Class Reference</td>
<td>622</td>
</tr>
<tr>
<td>13.87</td>
<td>NonDGPImpSampling Class Reference</td>
<td>624</td>
</tr>
<tr>
<td>13.88</td>
<td>NonDGPMSEBayesCalibration Class Reference</td>
<td>627</td>
</tr>
<tr>
<td>13.89</td>
<td>NonDIncremLHSampling Class Reference</td>
<td>630</td>
</tr>
<tr>
<td>13.90</td>
<td>NonDIntegration Class Reference</td>
<td>631</td>
</tr>
<tr>
<td>13.91</td>
<td>NonDInterval Class Reference</td>
<td>634</td>
</tr>
<tr>
<td>13.92</td>
<td>NonDLHSEvidence Class Reference</td>
<td>636</td>
</tr>
<tr>
<td>13.93</td>
<td>NonDHLHInterval Class Reference</td>
<td>637</td>
</tr>
<tr>
<td>13.94</td>
<td>NonDHLHSampling Class Reference</td>
<td>639</td>
</tr>
<tr>
<td>13.95</td>
<td>NonDHLHSSingleInterval Class Reference</td>
<td>641</td>
</tr>
<tr>
<td>13.96</td>
<td>NonDLocalEvidence Class Reference</td>
<td>642</td>
</tr>
<tr>
<td>13.97</td>
<td>NonDLocalInterval Class Reference</td>
<td>643</td>
</tr>
<tr>
<td>13.98</td>
<td>NonDLocalReliability Class Reference</td>
<td>646</td>
</tr>
<tr>
<td>13.99</td>
<td>NonDLocalSingleInterval Class Reference</td>
<td>655</td>
</tr>
<tr>
<td>13.100</td>
<td>NonDPOFDarts Class Reference</td>
<td>657</td>
</tr>
<tr>
<td>13.101</td>
<td>NonDPolynomialChaos Class Reference</td>
<td>659</td>
</tr>
<tr>
<td>13.102</td>
<td>NonDQuadrature Class Reference</td>
<td>663</td>
</tr>
<tr>
<td>13.103</td>
<td>NonDQUESOBayesCalibration Class Reference</td>
<td>666</td>
</tr>
<tr>
<td>13.104</td>
<td>NonDReliability Class Reference</td>
<td>669</td>
</tr>
<tr>
<td>13.105</td>
<td>NonDSampling Class Reference</td>
<td>671</td>
</tr>
<tr>
<td>13.106</td>
<td>NonDSparseGrid Class Reference</td>
<td>677</td>
</tr>
<tr>
<td>13.107</td>
<td>NonDStochCollocation Class Reference</td>
<td>680</td>
</tr>
<tr>
<td>13.108</td>
<td>NonlinearCGOptimizer Class Reference</td>
<td>682</td>
</tr>
<tr>
<td>13.109</td>
<td>NPSOLOptimizer Class Reference</td>
<td>685</td>
</tr>
<tr>
<td>13.110</td>
<td>OptDartsOptimizer Class Reference</td>
<td>687</td>
</tr>
<tr>
<td>13.111</td>
<td>Optimizer Class Reference</td>
<td>690</td>
</tr>
<tr>
<td>13.112</td>
<td>OutputManager Class Reference</td>
<td>695</td>
</tr>
<tr>
<td>13.113</td>
<td>ParallelConfiguration Class Reference</td>
<td>697</td>
</tr>
<tr>
<td>13.114</td>
<td>ParallelDirectApplicInterface Class Reference</td>
<td>699</td>
</tr>
<tr>
<td>13.115</td>
<td>ParallelLevel Class Reference</td>
<td>700</td>
</tr>
<tr>
<td>13.116</td>
<td>ParallelLibrary Class Reference</td>
<td>702</td>
</tr>
<tr>
<td>13.117</td>
<td>ParamResponsePair Class Reference</td>
<td>711</td>
</tr>
<tr>
<td>13.118</td>
<td>ParamStudy Class Reference</td>
<td>714</td>
</tr>
<tr>
<td>13.119</td>
<td>partial_prp_equality Struct Reference</td>
<td>718</td>
</tr>
<tr>
<td>13.120</td>
<td>partial_prp_hash Struct Reference</td>
<td>719</td>
</tr>
<tr>
<td>13.121</td>
<td>PecosApproximation Class Reference</td>
<td>719</td>
</tr>
<tr>
<td>13.122</td>
<td>ProblemDescDB Class Reference</td>
<td>724</td>
</tr>
<tr>
<td>13.123</td>
<td>ProcessApplicInterface Class Reference</td>
<td>732</td>
</tr>
<tr>
<td>13.124</td>
<td>ProcessHandleApplicInterface Class Reference</td>
<td>736</td>
</tr>
<tr>
<td>13.125</td>
<td>ProgramOptions Class Reference</td>
<td>739</td>
</tr>
<tr>
<td>13.126</td>
<td>StudyDACE Class Reference</td>
<td>744</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Reference</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>-----------</td>
</tr>
<tr>
<td>13.12</td>
<td>PSUADEDesignCompExp Class Reference</td>
<td>746</td>
</tr>
<tr>
<td>13.12</td>
<td>PythonInterface Class Reference</td>
<td>748</td>
</tr>
<tr>
<td>13.12</td>
<td>RecastBaseConstructor Struct Reference</td>
<td>750</td>
</tr>
<tr>
<td>13.13</td>
<td>RecastModel Class Reference</td>
<td>751</td>
</tr>
<tr>
<td>13.13</td>
<td>RelaxedVarConstraints Class Reference</td>
<td>759</td>
</tr>
<tr>
<td>13.13</td>
<td>RelaxedVariables Class Reference</td>
<td>760</td>
</tr>
<tr>
<td>13.13</td>
<td>Response Class Reference</td>
<td>762</td>
</tr>
<tr>
<td>13.13</td>
<td>ResponseRep Class Reference</td>
<td>766</td>
</tr>
<tr>
<td>13.13</td>
<td>ResultsDBAny Class Reference</td>
<td>772</td>
</tr>
<tr>
<td>13.13</td>
<td>ResultsEntry&lt;StoredType&gt; Class Template Reference</td>
<td>774</td>
</tr>
<tr>
<td>13.13</td>
<td>ResultsID Class Reference</td>
<td>775</td>
</tr>
<tr>
<td>13.13</td>
<td>ResultsManager Class Reference</td>
<td>776</td>
</tr>
<tr>
<td>13.13</td>
<td>ResultsNames Class Reference</td>
<td>778</td>
</tr>
<tr>
<td>13.14</td>
<td>RichExtrapVerification Class Reference</td>
<td>779</td>
</tr>
<tr>
<td>13.14</td>
<td>ScilabInterface Class Reference</td>
<td>781</td>
</tr>
<tr>
<td>13.14</td>
<td>SensAnalysisGlobal Class Reference</td>
<td>782</td>
</tr>
<tr>
<td>13.14</td>
<td>SeqHybridMetaIterator Class Reference</td>
<td>784</td>
</tr>
<tr>
<td>13.14</td>
<td>SerialDirectApplicInterface Class Reference</td>
<td>787</td>
</tr>
<tr>
<td>13.14</td>
<td>SharedApproxData Class Reference</td>
<td>789</td>
</tr>
<tr>
<td>13.14</td>
<td>SharedPecosApproxData Class Reference</td>
<td>793</td>
</tr>
<tr>
<td>13.14</td>
<td>SharedSurfpackApproxData Class Reference</td>
<td>796</td>
</tr>
<tr>
<td>13.14</td>
<td>SharedVariablesData Class Reference</td>
<td>798</td>
</tr>
<tr>
<td>13.14</td>
<td>SharedVariablesDataRep Class Reference</td>
<td>802</td>
</tr>
<tr>
<td>13.15</td>
<td>SingleModel Class Reference</td>
<td>805</td>
</tr>
<tr>
<td>13.15</td>
<td>NNLLBase Class Reference</td>
<td>807</td>
</tr>
<tr>
<td>13.15</td>
<td>NNLLLeastSq Class Reference</td>
<td>810</td>
</tr>
<tr>
<td>13.15</td>
<td>NNLOptimizer Class Reference</td>
<td>813</td>
</tr>
<tr>
<td>13.15</td>
<td>SOLBase Class Reference</td>
<td>819</td>
</tr>
<tr>
<td>13.15</td>
<td>SpawnApplicInterface Class Reference</td>
<td>822</td>
</tr>
<tr>
<td>13.15</td>
<td>SurfpackApproximation Class Reference</td>
<td>823</td>
</tr>
<tr>
<td>13.15</td>
<td>SurrBasedGlobalMinimizer Class Reference</td>
<td>826</td>
</tr>
<tr>
<td>13.15</td>
<td>SurrBasedLocalMinimizer Class Reference</td>
<td>827</td>
</tr>
<tr>
<td>13.15</td>
<td>SurrBasedMinimizer Class Reference</td>
<td>834</td>
</tr>
<tr>
<td>13.16</td>
<td>SurrogateModel Class Reference</td>
<td>838</td>
</tr>
<tr>
<td>13.16</td>
<td>SysCallApplicInterface Class Reference</td>
<td>842</td>
</tr>
<tr>
<td>13.16</td>
<td>TANA3Approximation Class Reference</td>
<td>845</td>
</tr>
<tr>
<td>13.16</td>
<td>TaylorApproximation Class Reference</td>
<td>847</td>
</tr>
<tr>
<td>13.16</td>
<td>TestDriverInterface Class Reference</td>
<td>848</td>
</tr>
<tr>
<td>13.16</td>
<td>TrackerHTTP Class Reference</td>
<td>853</td>
</tr>
<tr>
<td>13.16</td>
<td>Variables Class Reference</td>
<td>855</td>
</tr>
<tr>
<td>13.16</td>
<td>Verification Class Reference</td>
<td>863</td>
</tr>
</tbody>
</table>

### 14 File Documentation

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>14.1</td>
<td>dakota_dll_api.cpp File Reference</td>
<td>865</td>
</tr>
<tr>
<td>14.2</td>
<td>dakota_dll_api.h File Reference</td>
<td>866</td>
</tr>
<tr>
<td>14.3</td>
<td>dakota_tabular_io.hpp File Reference</td>
<td>867</td>
</tr>
<tr>
<td>14.4</td>
<td>dll_tester.cpp File Reference</td>
<td>868</td>
</tr>
<tr>
<td>14.5</td>
<td>JEGAOptimizer.cpp File Reference</td>
<td>869</td>
</tr>
<tr>
<td>14.6</td>
<td>JEGAOptimizer.hpp File Reference</td>
<td>869</td>
</tr>
<tr>
<td>14.7</td>
<td>library_mode.cpp File Reference</td>
<td>870</td>
</tr>
<tr>
<td>Section</td>
<td>File Reference</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------</td>
<td>------</td>
</tr>
<tr>
<td>14.8 library_split.cpp</td>
<td></td>
<td>873</td>
</tr>
<tr>
<td>14.9 main.cpp</td>
<td></td>
<td>874</td>
</tr>
<tr>
<td>14.10 restart_util.cpp</td>
<td></td>
<td>875</td>
</tr>
</tbody>
</table>

**Index**

876
Chapter 1

Dakota Developers Manual

Author

1.1 Introduction

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

The Developers Manual focuses on documentation of Dakota design principles and class structures; it derives principally from annotated source code. For information on input command syntax, refer to the Reference Manual, and for more details on Dakota features and capabilities, refer to the Users Manual.

1.2 Overview of Dakota

In Dakota, the environment manages execution modes and input/output streams and defines the top-level iterator. This top-level iterator may be either a standard iterator or a meta-iterator. In the former case, the iterator identifies a model and the environment executes the iterator on the model to perform a single study. In the latter case, iterator recursions are present and sub-iterators may identify their own models. In both cases, models may contain additional recursions in the case of nested iteration or surrogate modeling. In a simple example, a hybrid meta-iterator might manage a global optimizer operating on a low-fidelity model that feeds promising design points into a local optimizer operating on a high-fidelity model. And in a more advanced example, a surrogate-based optimization under uncertainty approach would employ an uncertainty quantification iterator nested within an
optimization iterator and would employ truth models contained 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 environment, method, model, variables, interface, and responses keyword specifications.

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

1.2.1 Environment

Class hierarchy: Environment.

Environments provide the top level abstraction for managing different execution modes and managing input and output streams. Specific environments include:

- **ExecutableEnvironment**: the environment for execution of Dakota as a stand-alone application.
- **LibraryEnvironment**: the environment for execution of Dakota as an embedded library service.

1.2.2 Iterators

Class hierarchy: Iterator. Iterator implementations may choose to split operations up into run-time phases as described in Understanding Iterator Flow.

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

The **MetaIterator** classes manage sequencing and collaboration among multiple methods with support for concurrent iterator parallelism. Methods include:

- **SeqHybridMetaIterator**: hybrid minimization using a set of iterators employing a corresponding set of models of varying fidelity. The sequential hybrid passes the best solutions from one method in as the starting points of the next method in the sequence.

- **CollabHybridMetaIterator**: hybrid minimization employing collaboration and sharing of response data among methods during the course if iteration. This class is currently a placeholder.

- **EmbedHybridMetaIterator**: hybrid minimization involving periodic use of a local search method for refinement during the iteration of an outer global method. This class is currently a placeholder.

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

The **Minimizer** classes address optimization and deterministic calibration and are grouped into:
1.2. OVERVIEW OF DAKOTA

- Optimization: Optimizer provides a base class for gradient-based (e.g., CONMINOptimizer and SNLLOptimizer) and derivative-free (e.g., NCSUOptimizer, JEGAOptimizer) optimization solvers. Most of these are wrappers for third-party libraries that implement the optimization algorithms. Classes APPSEvalMgr and COLINApplication provide the function evaluation interface for APPSOptimizer and COLINOptimizer, respectively.

- 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 (both optimization and nonlinear least squares): SurrBasedMinimizer provides a base class for SurrBasedLocalMinimizer, SurrBasedGlobalMinimizer, and EffGlobalMinimizer. The surrogate-based local and global methods employ a single iterator with any of the available SurrogateModel capabilities (local, multipoint, or global data fits or hierarchical approximations) and perform a sequence of approximate optimizations, each involving build, optimize, and verify steps. The efficient global method, on the other hand, hard-wires a recursion involving Gaussian process surrogate models coupled with the DIRECT global optimizer to maximize an expected improvement function.

The Analyzer classes are grouped into:

- Uncertainty quantification: NonD provides a base class for non-deterministic methods in several categories:
  - Sampling: NonDSampling is further specialized with the NonDLHSSampling class for Latin hypercube and Monte Carlo sampling, and a number of other classes supporting incremental and adaptive sampling such as NonDAdaptImpSampling for multi-modal adaptive importance sampling.
  - Reliability Analysis: NonDReliability is further specialized with local and global methods (NonDLocalReliability and NonDGlobalReliability). NonDPOFDarts implements a computational geometry-based reliability method.
  - Stochastic Expansions: NonDExpansion includes specializations for generalized polynomial chaos (NonDPolynomialChaos) and stochastic collocation (NonDStochCollocation) and is supported by the NonDIntegration helper class, which supplies cubature, tensor-product quadrature and Smolyak sparse grid methods (NonDCubature, NonDQuadrature, and NonDSparseGrid).
  - Bayesian Calibration: NonDCalibration provides a base class for nondeterministic calibration methods with specialization to Bayesian calibration in NonDBayesCalibration, and specific implementations such as NonDQUESOBayesCalibration.
  - NonDInterval provides a base class for epistemic interval-based UQ methods. Three interval analysis approaches are provided: LHS (NonDLHSInterval), efficient global optimization (NonDGlobalInterval), and local optimization (NonDLocalInterval). Each of these three has specializations for single interval and Dempster-Shafer Theory of Evidence approaches.
  - Experimental: EfficientSubspaceMethod implements a prototype input-space dimension reduction method for UQ.

- Parameter studies and design of experiments: PStudyDACE provides a base class for ParamStudy, which provides capabilities for directed parameter space interrogation, PSUADEDesignCompExp, 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 when in active all variables mode.

- Solution verification studies: Verification provides a base class for RichExtrapVerification (verification via Richardson extrapolation) and other solution verification methods in development.
1.2.3 Models

Class hierarchy: Model.

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

- **SingleModel**: variables are mapped into responses using a single Interface object. No sub-iterators or sub-models are used.

- **SurrogateModel**: variables are mapped into responses using an approximation. The approximation is built and/or corrected using data from a sub-model (the truth model) and the data may be obtained using a sub-iterator (a design of experiments iterator). SurrogateModel has two derived classes: DataFitSurrModel for data fit surrogates and HierarchSurrModel for hierarchical models of varying fidelity. The relationship of the sub-iterators and sub-models is considered to be "layered" since they are not used as part of every response evaluation on the top level model, but rather used periodically in surrogate update and verification steps.

- **NestedModel**: variables are mapped into responses using a combination of an optional Interface and a sub-iterator/sub-model pair. The relationship of the sub-iterators and sub-models is considered to be "nested" since they are used to perform a complete iterative study as part of every response evaluation on the top level model.

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

1.2.4 Variables

Class hierarchy: Variables.

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

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

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

Whereas domain types are defined based on the derived Variables class selection, the selection of active variable types is handled within each of these derived classes using variable views. These permit different algorithms to work on different subsets of variables. Data shared among Variables instances is stored in SharedVariablesData. For details on managing variables, see Working with Variable Containers and Views.

The Constraints hierarchy manages bound, linear, and nonlinear constraints and utilizes the same specializations for managing bounds on the variables (see MixedVarConstraints and RelaxedVarConstraints).
1.2. OVERVIEW OF DAKOTA

1.2.5 Interfaces

Class hierarchy: Interface.

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

- **SysCallApplicInterface**: the simulation is invoked using a system call (the C function `system()`). Asynchronous invocation utilizes a background system call. Utilizes the CommandShell utility.

- **ForkApplicInterface**: the simulation is invoked using a fork (the `fork/exec/wait` family of functions). Asynchronous invocation utilizes a nonblocking fork.

- **SpawnApplicInterface**: for Windows, fork is replaced by spawn. Asynchronous invocation utilizes a non-blocking spawn.

Fork and Spawn are inherited from ProcessHandleApplicInterface and System and ProcessHandle are inherited from ProcessApplicInterface. A semi-intrusive approach is also supported by:

- **DirectApplicInterface**: the simulation is linked into the Dakota executable and is invoked using a procedure call. Asynchronous invocations will utilize nonblocking threads (capability not yet available). Specializations of the direct interface are implemented in MatlabInterface, PythonInterface, ScilabInterface, and (for built-in testers) TestDriverInterface, while examples of plugin interfaces for library mode in serial and parallel, respectively, are included in SerialDirectApplicInterface and ParallelDirectApplicInterface.

Scheduling of jobs for asynchronous local, message passing, and hybrid parallelism approaches is performed in the ApplicationInterface class, with job initiation and job capture specifics implemented in the derived classes.

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

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

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

1.2.6 Responses

Class: Response.

The Response class provides an abstract data representation of response functions and their first and second derivatives (gradient vectors and Hessian matrices). These response functions can be interpreted as objective functions and constraints (optimization data set), residual functions and constraints (least squares data set), or generic response functions (uncertainty quantification data set). This class is not currently part of a class hierarchy, since the abstraction has been sufficiently general and has not required specialization.
1.3 Services

A variety of services and utilities are used 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 here.

- Multilevel parallel computing: Dakota supports multiple levels of nested parallelism. A meta-iterator can manage concurrent iterators, each of which manages concurrent function evaluations, each of which manages concurrent analyses executing on multiple processors. Partitioning of these levels with MPI communicators is managed in ParallelLibrary and scheduling routines for the levels are part of IteratorScheduler, ApplicationInterface, and ForkApplicInterface.

- Option management: Global options controlling behavior are managed in ProgramOptions, with the help of command-line option parsing in CommandLineHandler.

- Parsing: Dakota employs NIDR (New Input Deck Reader) via Dakota::ProblemDescDB::parse_inputs to parse user input files. NIDR uses the keyword handlers in the NIDRProblemDescDB derived class to populate data within the ProblemDescDB base class, which maintains a DataEnvironment specification and lists of DataMethod, DataModel, DataVariables, DataInterface, and DataResponses specifications. Procedures for modifying the parsing subsystem are described in Instructions for Modifying Dakota's Input Specification.

- Failure capturing: Simulation failures can be trapped and managed using exception handling in ApplicationInterface and its derived classes.

- Restart: Dakota maintains a record of all function evaluations both in memory (for capturing any duplication) and on the file system (for restarting runs). Restart options are managed through ProgramOptions (with the help of CommandLineHandler); file management in OutputManager; and restart file insertions occur in ApplicationInterface. The dakota_restart_util executable, built from restart_util.cpp, provides a variety of services for interrogating, converting, repairing, concatenating, and post-processing restart files.

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

- Graphics and Output: Dakota provides 2D iteration history graphics using Motif widgets. Graphics data can also be cataloged in a tabular data file for post-processing with 3rd party tools such as Matlab, Tecplot, etc. These capabilities are encapsulated within the Graphics class. An experimental results database is implemented in ResultsManager and ResultsDBAny. Options for controlling output and facilities for managing it are in OutputManager.

1.4 Development Practices and Guidance

The following links provide guidance for core software components or specific development activities:

- Coding Style Guidelines and Conventions - coding practices used by the Dakota development team.
• **Instructions for Modifying Dakota’s Input Specification** - how to interact with NIDR and the associated Dakota classes.

• **Interfacing with Dakota as a Library** - embed Dakota as a service within your application.

• **Understanding Iterator Flow** - explanation of the full granularity of steps in Iterator execution.

• **Performing Function Evaluations** - an overview of the classes and member functions involved in performing function evaluations synchronously or asynchronously.

• **Working with Variable Containers and Views** - discussion of data storage for variables and explanation of active and inactive views of this data.

1.5 **Additional Resources**

Additional development resources include:

• The Dakota Developer Portal linked from [http://dakota.sandia.gov/developer/](http://dakota.sandia.gov/developer/) includes information on getting started as a developer and links to project management resources.

• Project web pages are maintained at [http://dakota.sandia.gov/](http://dakota.sandia.gov/) including links to frequently asked questions, documentation, publications, mailing lists, and other resources.
Chapter 2

Coding Style Guidelines and Conventions

2.1 Introduction

Common code development practices can be extremely useful in multiple developer environments. Particular styles for code components lead to improved readability of the code and can provide important visual cues to other developers. Much of this recommended practices document is borrowed from the CUBIT mesh generation project, which in turn borrows its recommended practices from other projects, yielding some consistency across Sandia projects. While not strict requirements, these guidelines suggest a best-practices starting point for coding in Dakota.

2.2 C++/c Style Guidelines

Style guidelines involve the ability to discern at a glance the type and scope of a variable or function.

2.2.1 Class and variable styles

Class names should be composed of two or more descriptive words, with the first character of each word capitalized, e.g.:

class ClassName;

Class member variables should be composed of two or more descriptive words, with the first character of the second and succeeding words capitalized, e.g.:

double classMemberVariable;

Temporary (i.e. local) variables are lower case, with underscores separating words in a multiple word temporary variable, e.g.:

int temporary_variable;

Constants (i.e. parameters) and enumeration values are upper case, with underscores separating words, e.g.:

const double CONSTANT_VALUE;
CHAPTER 2. CODING STYLE GUIDELINES AND CONVENTIONS

2.2.2 Function styles

Function names are lower case, with underscores separating words, e.g.:

```c
int function_name();
```

There is no need to distinguish between member and non-member functions by style, as this distinction is usually clear by context. This style convention allows member function names which set and return the value of a similarly-named private member variable, e.g.:

```c
int memberVariable;
void member_variable(int a) { // set
    memberVariable = a;
}
int member_variable() const { // get
    return memberVariable;
}
```

In cases where the data to be set or returned is more than a few bytes, it is highly desirable to employ const references to avoid unnecessary copying, e.g.:

```c
void continuous_variables(const RealVector& c_vars) { // set
    continuousVariables = c_vars;
}
const RealVector& continuous_variables() const { // get
    return continuousVariables;
}
```

Note that it is not necessary to always accept the returned data as a const reference. If it is desired to be able change this data, then accepting the result as a new variable will generate a copy, e.g.:

```c
// reference to continuousVariables cannot be changed
const RealVector& c_vars = model.continuous_variables();
// local copy of continuousVariables can be changed
RealVector c_vars = model.continuous_variables();
```

2.2.3 Miscellaneous

Appearance of typedefs to redefine or alias basic types is isolated to a few header files (`data_types.h`, `template_defs.h`), so that issues like program precision can be changed by changing a few lines of typedefs rather than many lines of code, e.g.:

```c
typedef double Real;
```

`xemacs` is the preferred source code editor, as it has C++ modes for enhancing readability through color (turn on "Syntax highlighting"). Other helpful features include "Paren highlighting" for matching parentheses and the "New Frame" utility to have more than one window operating on the same set of files (note that this is still the same edit session, so all windows are synchronized with each other). Window width should be set to 80 internal columns, which can be accomplished by manual resizing, or preferably, using the following alias in your shell resource file (e.g., `.cshrc`):

```c
alias xemacs "xemacs -g 81x63"
```

where an external width of 81 gives 80 columns internal to the window and the desired height of the window will vary depending on monitor size. This window width imposes a coding standard since you should avoid line wrapping by continuing anything over 80 columns onto the next line.

Indenting increments are 2 spaces per indent and comments are aligned with the code they describe, e.g.:
2.3. **FILE NAMING CONVENTIONS**

```c
void abort_handler(int code)
{
    int initialized = 0;
    MPI_Init(&initialized);
    if (initialized) {
        // comment aligned to block it describes
        // int size;
        MPI_Comm_size(MPI_COMM_WORLD, &size);
        if (size>1)
            MPI_Abort(MPI_COMM_WORLD, code);
        else
            exit(code);
    }
    else
        exit(code);
}
```

Also, the continuation of a long command is indented 2 spaces, e.g.:

```c
const Strings 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\to be calculated by the " << methodSource << " finite difference routine." << endl;
```

Lastly, `#ifdef`’s are not indented (to make use of syntax highlighting in xemacs).

### 2.3 File Naming Conventions

In addition to the style outlined above, the following file naming conventions have been established for the Dakota project.

File names for C++ classes should, in general, use the same name as the class defined by the file. Exceptions include:

- with the introduction of the Dakota namespace, base classes which previously utilized prepended Dakota identifiers can now safely omit the identifiers. However, since file names do not have namespace protection from name collisions, they retain the prepended Dakota identifier. For example, a class previously named DakotaModel which resided in DakotaModel.cpp/hpp, is now Dakota::Model (class Model in namespace Dakota) residing in the same filenames. The retention of the previous filenames reduces the possibility of multiple instances of a Model.hpp causing problems. Derived classes (e.g., NestedModel) do not require a prepended Dakota identifier for either the class or file names.

- in a few cases, it is convenient to maintain several closely related classes in a single file, in which case the file name may reflect the top level class or some generalization of the set of classes (e.g., DakotaResponse.[CH] files contain Dakota::Response and Dakota::ResponseRep classes, and DakotaBinStream.[CH] files contain the Dakota::BiStream and Dakota::BoStream classes).

The type of file is determined by one of the four file name extensions listed below:

- **.hpp** A class header file ends in the suffix .hpp. The header file provides the class declaration. This file does not contain code for implementing the methods, except for the case of inline functions. Inline functions are to be placed at the bottom of the file with the keyword inline preceding the function name.

- **.cpp** A class implementation file ends in the suffix .cpp. An implementation file contains the definitions of the members of the class.
• **.h** A header file ends in the suffix .h. The header file contains information usually associated with procedures. Defined constants, data structures and function prototypes are typical elements of this file.

• **.c** A procedure file ends in the suffix .c. The procedure file contains the actual procedures.

### 2.4 Class Documentation Conventions

Class documentation uses the doxygen tool available from [http://www.doxygen.org](http://www.doxygen.org) and employs the Java-doc comment style. Brief comments appear in header files next to the attribute or function declaration. Detailed descriptions for functions should appear alongside their implementations (i.e., in the .cpp files for non-inlined, or in the headers next to the function definition for inlined). Detailed comments for a class or a class attribute must go in the header file as this is the only option.

NOTE: Previous class documentation utilities (class2frame and class2html) used the “//-” comment style and comment blocks such as this:

```c
//  Class: Model
//  Description: The model to be iterated by the Iterator.
//  Contains Variables, Interface, and Response objects.
//  Owner: Mike Eldred
```

These tools are no longer used, so remaining comment blocks of this type are informational only and will not appear in the documentation generated by doxygen.

### 2.5 CMake Style Guidelines

Dakota conventions for CMake files, such as CMakeLists.txt, FooConfig.cmake, etc., follow. Our goal is ease of reading, maintenance, and support, similar to the C++ code itself. Current CMake versions and build hints are maintained at the Developer Portal [http://dakota.sandia.gov/developer/](http://dakota.sandia.gov/developer/).

#### 2.5.1 CMake Code Formatting

- Indentation is 2 spaces, consistent with Dakota C++ style.
- Lines should be kept to less than 80 chars per line where possible.
- Wrapped lines may be indented two spaces or aligned with prior lines.
- For ease of viewing and correctness checking in Emacs, a customization file is available: [http://www.-cmake.org/CMakeDocs/cmake-mode.el](http://www.-cmake.org/CMakeDocs/cmake-mode.el)

#### 2.5.2 CMake Variable Naming Conventions

These variable naming conventions are especially important for those that ultimately become preprocessor defines and affect compilation of source files.

- Classic/core elements of the CMake language are set in lower_case, e.g., option, set, if, find_library.
- Static arguments to CMake functions and macros are set in UPPER_CASE, e.g. REQUIRED, NO_MODULE, QUIET.
- Minimize “global” variables, i.e., don’t use 2 variables with the same meaning when one will do the job.
2.5. **CMAKE STYLE GUIDELINES**

- Feature toggling: when possible, use the "HAVE_<pkg/feature>" convention already in use by many C-Make-enabled TPLs, e.g.,

  ```bash
  $ grep HAVE_SYSTEM Dakota/src/CMakeLists.txt
  
  check_function_exists(system HAVE_SYSTEM)
  if(HAVE_SYSTEM)
    add_definitions("-DHAVE_SYSTEM")
  endif(HAVE_SYSTEM)
  
  $ grep HAVE_CONMIN Dakota/src/CMakeLists.txt Dakota/packages/CMakeLists.txt
  Dakota/src/CMakeLists.txt:if(HAVE_CONMIN)
  Dakota/src/CMakeLists.txt:endif(HAVE_CONMIN)
  Dakota/packages/CMakeLists.txt:option(HAVE_CONMIN "Build the CONMIN package." ON)
  Dakota/packages/CMakeLists.txt:endif(HAVE_CONMIN)
  
  - When a variable/preprocessor macro could result in name clashes beyond Dakota scope, e.g., for library-mode users, consider prefixing the "HAVE_<pkg>" name with DAKOTA_, e.g. DAKOTA_HAVE_MPI. Currently, MPI is the only use case for such a variable in Dakota, but many examples can be found in the CMake Modules source, e.g.

    ```bash
    grep _HAVE_ <cmake_prefix_dir>/share/cmake-2.8/Modules/*
    ```
Chapter 3

Instructions for Modifying Dakota’s Input Specification

To modify Dakota’s input specification (for maintenance or addition of new input syntax), specification maintenance mode must be enabled at Dakota configure time with the -DENABLE_SPEC_MAINT option, e.g.,

```
./cmake -DENABLE_SPEC_MAINT:BOOL=ON ..
```

This will enable regeneration of NIDR and Dakota components which must be updated following a spec change.

3.1 Modify dakota.input.nspec

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

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

Warning

- Do not skip this step. Attempts to modify the NIDR_keywds.hpp file in Dakota/src without using the NIDR table generator are very error-prone. Moreover, the input specification provides a reference to the allowable inputs of a particular executable and should be kept in synch with the parser files; modifying the parser files independent of the input specification creates, at a minimum, undocumented features.

- All keywords in dakota.input.nspec are lower case by convention. All user inputs are converted to lower case by the parser prior to keyword match testing, resulting in case insensitive parsing.
• Since the NIDR parser allows abbreviation of keywords, you must avoid adding a keyword that could be misinterpreted as an abbreviation for a different keyword within the same top-level keyword, such as "environment" and "method". For example, adding the keyword "expansion" within the method specification would be a mistake if the keyword "expansion_factor" already was being used in this specification.

• The NIDR input is somewhat order-dependent, allowing the same keyword to be reused multiple times in the specification. This often happens with aliases, such as lower_bounds, upper_bounds and initial_point. Ambiguities are resolved by attaching a keyword to the most recently seen context in which it could appear, if such exists, or to the first relevant context that subsequently comes along in the input file. With the earlier IDR parser, non-exclusive specifications (those not in mutually exclusive blocks) were required to be unique. That is why there are such aliases for initial_point as cdv_initial_point and ddv_initial_point: so older input files can be used with no or fewer changes.

3.2 Rebuild generated files

When configured with -DENABLE_SPEC_MAINT, performing a make in Dakota/src will regenerate all files which derive from dakota.input.nspec, including NIDR_keywds.hpp, dakota.input.summary, NIDR_guikywds.h, and dakota.input.desc. If you commit changes to a source repository, be sure to commit any automatically generated files in addition to any modified in the following steps. It is not strictly necessary to run make at this point in the sequence, and in fact may generate errors if necessary handlers aren’t yet available.

3.3 Update NIDRProblemDescDB.cpp in Dakota/src

Many keywords have data associated with them: an integer, a floating-point number, a string, or arrays of such entities. Data requirements are specified in dakota.input.nspec by the tokens INTEGER, REAL, STRING, INTEGERLIST, REALLIST, STRINGLIST. (Some keywords have no associated data and hence no such token.) After each keyword and data token, the dakota.input.nspec file specifies functions that the NIDR parser should call to record the appearance of the keyword and deal with any associated data. The general form of this specification is

\{ startfcn, startdata, stopfcn, stopdata \}

i.e., a brace-enclosed list of one to four functions and data pointers, with trailing entities taken to be zero if not present; zero for a function means no function will be called. The startfcn must deal with any associated data. Otherwise, the distinction between startfcn and stopfcn is relevant only to keywords that begin a group of keywords (enclosed in parentheses or square brackets). The startfcn is called before other entities in the group are processed, and the stop function is called after they are processed. Top-level keywords often have both startfcn and stopfcn; stopfcn is uncommon but possible for lower-level keywords. The startdata and (if needed) stopdata values are usually pointers to little structures that provide keyword-specific details to generic functions for startfcn and stopfcn. Some keywords that begin groups (such as "approx_problem" within the top-level "environment" keyword) have no need of either a startfcn or a stopfcn; this is indicated by "\{0\}".

Most of the things within braces in dakota.input.nspec are invocations of macros defined in NIDRProblemDescDB.cpp. The macros simplify writing dakota.input.nspec and make it more readable. Most macro invocations refer to little structures defined in NIDRProblemDescDB.cpp, usually with the help of other macros, some of which have different definitions in different parts of NIDRProblemDescDB.cpp. When adding a keyword to dakota.input.nspec, you may need to add a structure definition or even introduce a new data type. NIDRProblemDescDB.cpp has sections corresponding to each top-level keyword. The top-level keywords are in alphabetical order, and most entities in the section for a top-level keyword are also in alphabetical order. While not required, it is probably good practice to maintain this structure, as it makes things easier to find.
Any integer, real, or string data associated with a keyword are provided to the keyword’s startfcn, whose second argument is a pointer to a Values structure, defined in header file nidr.h.

**Example 1:** if you added the specification:

```
[method_setting REAL {method_setting_start, &method_setting_details}]
```
you would provide a function

```cpp
void NIDRProblemDescDB::
method_setting_start(const char *keyname, Values *val, void **g, void *v)
{ ... }
```
in NIDRProblemDescDB.cpp. In this example, argument &method_setting_details would be passed as v, val->n (the number of values) would be 1 and *val->r would be the REAL value given for the method-setting keyword. The method_setting_start function would suitably store this value with the help of method_setting_details.

For some top-level keywords, g (the third argument to the startfcn and stopfcn) provides access to a relevant context. For example, method_start (the startfcn for the top-level method keyword) executes

```cpp
DataMethod *dm = new DataMethod;
g = (void*)dm;
```
(and supplies a couple of default values to dm). The start functions for lower-level keywords within the method keyword get access to dm through their g arguments. Here is an example:

```cpp
void NIDRProblemDescDB::
method_str(const char *keyname, Values *val, void **g, void *v)
{
  (*(DataMethod**)g)->**(String DataMethod::**)v = *val->s;
}
```
In this example, v points to a pointer-to-member, and an assignment is made to one of the components of the DataMethod object pointed to by *g. The corresponding stopfcn for the top-level method keyword is

```cpp
void NIDRProblemDescDB::
method_stop(const char *keyname, Values *val, void **g, void *v)
{
  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

```
[method_setting REALLIST {{N_mdm(RealL,methodCoeffs)}]
```
then method_RealL (defined in NIDRProblemDescDB.cpp) would be called as the startfcn, and methodCoeffs would be the name of a (currently nonexistent) component of DataMethod. The N_mdm macro is defined in NIDRProblemDescDB.cpp; among other things, it turns RealL into NIDRProblemDescDB::method_RealL. This function is used to process lists of REAL values for several keywords. By looking at the source, you can see that the list values are val->r[i] for 0 <= i < val->n.

## 3.4 Update ProblemDescDB.cpp in Dakota/src

### 3.4.1 Augment/update get_<data_type>() functions

The next update step involves extending the database retrieval functions in ProblemDescDB.cpp. These retrieval functions accept an identifier string and return a database attribute of a particular type, e.g., a RealVector:
const RealVector& get_rv(const String& entry_name);

The implementation of each of these functions contains tables of possible entry_name values and associated pointer-to-member values. There is one table for each relevant top-level keyword, with the top-level keyword omitted from the names in the table. Since binary search is used to look for names in these tables, each table must be kept in alphabetical order of its entry names. For example,

```c++
... else if ((L = Begins(entry_name, "model."))) {
   if ((dbRep->methodDBLocked)
      Locked_db();
#define P &DataModelRep::
    static KW<RealVector, DataModelRep> RVdmo[] = {
      "nested.primary_response_mapping", P primaryRespCoeffs,
      "nested.secondary_response_mapping", P secondaryRespCoeffs,
      "surrogate.kriging.conmin_seed", P krigingConminSeed,
      "surrogate.kriging.correlations", P krigingCorrelations,
      "surrogate.kriging.max_correlations", P krigingMaxCorrelations,
      "surrogate.kriging.min_correlations", P krigingMinCorrelations};
#undef P
    KW<RealVector, DataModelRep>* kw;
    if ((kw = (KW<RealVector, DataModelRep>*)Binsearch(RVdmo, L)))
       return dbRep->dataModelIter->dataModelRep->*kw->p;
}
```

is the "model" portion of ProblemDescDB::get_rv(). Based on entry_name, it returns the relevant attribute from a DataModel object. Since there may be multiple model specifications, the dataModelIter list iterator identifies which node in the list of DataModel objects is used. In particular, dataModellist contains a list of all of the data_model objects, one for each time a top-level model keyword was seen by the parser. The particular model object used for the data retrieval is managed by dataModelIter, which is set in a set_db_list_nodes() operation that will not be described here.

There may be multiple DataMethod, DataModel, DataVariables, DataInterface, and/or DataResponses objects. However, only one specification is currently allowed so a list of DataEnvironment objects is not needed. Rather, ProblemDescDB::environmentSpec is the lone DataEnvironment 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 dbRep->listIter->attribute syntax for variables, interface, responses, and method specifications. For example, the method setting example attribute would be added to get_drv() as:

```c++
{"method_name.method_setting", P methodSetting},
```

inserted at the beginning of the RVdmo array shown above (since the name in the existing first entry, i.e., "nested.-primary_response_mapping", comes alphabetically after "method_name.method_setting").

### 3.5 Update Corresponding Data Classes

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

#### 3.5.1 Update the Data class header file

Add a new attribute to the public data for each of the new specifications. Follow the style guide for class attribute naming conventions (or mimic the existing code).
3.6. USE GET,<DATA_TYPE>() FUNCTIONS

3.5.2 Update the .cpp file

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

3.6 Use get,<data_type>() Functions

At this point, the new specifications have been mapped through all of the database classes. The only remaining step is to retrieve the new data within the constructors of the classes that need it. This is done by invoking the get,<data_type>() function on the ProblemDescDB object using the identifier string you selected in Augment/update get,<data_type>() functions. For example:

```cpp
const String& interface_type = problem_db.get_string("interface.type");
```

passes the "interface.type" identifier string to the ProblemDescDB::get_string() retrieval function, which returns the desired attribute from the active DataInterface object.

Warning

Use of the get,<data_type>() functions is restricted to class constructors, since only in class constructors are the data list iterators (i.e., dataMethodIter, dataModelIter, dataVariablesIter, dataInterfaceIter, and dataResponsesIter) guaranteed to be set correctly. Outside of the constructors, the database list nodes will correspond to the last set operation, and may not return data from the desired list node.

3.7 Update the Documentation

Doxygen comments should be added to the Data class headers for the new attributes, and the reference manual sections describing the portions of dakota.input.nspec that have been modified should be updated. In particular, the reference manual tables summarizing keywords provide help data to the Jaguar user interface so need to be kept updated.
Chapter 4

Understanding Iterator Flow

This page explains the various phases comprising Iterator.run_iterator(). Prior to Iterator construction, when command-line options are parsed, Boolean run mode flags corresponding to PRERUN, RUN, and POSTRUN are set in ParallelLibrary. If the user didn’t specify any specific run modes, the default is for all three to be true (all phases will execute).

Iterator is constructed.

When called, run_iterator() sequences:

- initialize_run(): unconditionally called, virtual. Performs common initialization such as allocating workspaces, setting communicators and evaluation counts. When re-implementing this virtual, a derived class must call its nearest parent’s initialize_run(), typically before performing its own implementation steps.

- Not implemented: pre-run input

- IF PRERUN, invoke pre_run(): virtual function; default no-op. Purpose: derived classes should implement pre_run() if they are able to generate all parameter sets (variables) at once, separate from run(). Derived implementations should call their nearest parent’s pre_run(), typically before performing their own steps.

- IF PRERUN, invoke pre_output(): non-virtual function; if user requested, output variables to file.

- Not implemented: run input

- IF RUN, invoke virtual function run(). Purpose: at a minimum, evaluate parameter sets through computing responses; for iterators without pre/post capability, their entire implementation is in run() and this is a reasonable default for new Iterators.

- Not implemented: run output

- IF POSTRUN, invoke post_input(): virtual function, default only print helpful message on mode. Purpose: derived iterators supporting post-run input from file must implement to read file and populate variables/responses (and possibly best points) appropriately. Implementations must check if the user requested file input.

- IF POSTRUN, invoke post_run(): virtual function. Purpose: generate statistics / final results. Any analysis that can be done solely on tabular data read by post_input() can be done here. Derived re-implementations should call their nearest parent’s post-run(), typically after performing their specific post-run activities.

- Not implemented: post-run output
• `finalize_run()`: unconditionally called, virtual. Purpose: free workspaces. Default base class behavior is no-op, however, derived implementations should call their nearest parent’s `finalize_run` after performing their specialized portions.

`Iterator` is destructed.
Chapter 5

Interfacing with Dakota as a Library

5.1 Introduction

Tightly integrating or linking Dakota into another application can improve user experience by delivering a more unified, inter-operable software tool for optimization and UQ analyses, improving performance by eliminating file system-based interfaces, and reducing challenges with parallel computing inter-operation. This benefit has been realized within several Sandia and external simulation applications. This section describes how to link Dakota into another C++ application.

Dakota has two primary application programming interfaces (APIs). The LibraryEnvironment class facilitates use of Dakota as an algorithm service library within another application. In this case, the simulation application is providing a "front end" for Dakota. The second API, provided by the DirectApplicInterface class, provides an interface for Dakota to call the simulation code directly to perform function evaluations in core. This permits the simulation to be the "back end" for Dakota. The most complete library integration of Dakota would use both in combination, with the overall simulation framework providing both the front end and back end for Dakota, creating a sandwich, as loosely depicted here:

```
[------------
| Application
| |
| { ------
| { Dakota {LibraryEnvironment}
| |
| { { Function evaluation callback to Application (via DirectApplicInterface)
| { { |
| <--------
| { { |
| { |
| { |
| { |
| { ------
| |----------
```

Attention

Dakota may be integrated as a library in other software applications subject to the terms of the GNU Lesser General Public License (LGPL). Refer to http://www.gnu.org/licenses/lgpl.html or the LICENSE file included with Dakota.

When Dakota is compiled and installed, the relevant library API headers are installed to CMAKE_INSTALL_PREFIX/include and the runtime libraries primarily to CMAKE_INSTALL_PREFIX/lib/ (on some platforms,
CHAPTER 5. INTERFACING WITH DAKOTA AS A LIBRARY

to CMAKE_INSTALL_PREFIX/bin/. The core C/C++ code is in the library dakota_src, while Fortran code lives in the dakota_src_fortran library. Information on using the API in Dakota headers is included throughout this section, while considerations for configuring and linking against Dakota and its various required and optional third-party libraries are emphasized in the section Linking against the Dakota library.

Steps involved in integrating Dakota into another application typically include:

1. Writing C++ code for your application to instantiate, configure, and execute Dakota’s LibraryEnvironment (“front end”); see Basic Dakota library instantiation and Configuring Dakota operation.

2. Writing C++ code for Dakota to call a function in your application to perform function evaluations (“back end”); see Creating a simulator plugin interface.

3. Compiling Dakota and linking into your application (Linking against the Dakota library).

Several source code examples demonstrate Dakota library interfaces. The classes SIM::SerialDirectApplicInterface and SIM::ParallelDirectApplicInterface demonstrate serial and parallel simulation function evaluation plug-ins. The file library_mode.cpp includes a main program that exercises Dakota libraries in serial and parallel modes with these mock simulator programs, with various ways of configuring Dakota problem definition and operation. Finally, library_split.cpp demonstrates running Dakota as a library modular on an MPI sub-communicator.

5.2 Basic Dakota library instantiation

The function run_dakota_parse() in library_mode.cpp demonstrates the basic use of Dakota library objects as one would in another main application that embeds Dakota. In this example, Dakota is configured based on a typical user-provided text-based Dakota input file (the same that would be provided at the command line with dakota -i dakota_optimization.in) and a function evaluator derived from a DirectApplicInterface is plugged into the Dakota library environment.

First, an object of type ProgramOptions which manages top-level Dakota settings is instantiated and configured to specify the name of the Dakota user input file. Additional options for output and error redirection, restart operation, and more may be set via ProgramOptions. See its class documentation for details.

```
string dakota_input_file = "dakota_optimization.in";
Dakota::ProgramOptions opts;
opts.input_file(dakota_input_file);
```

Next, a LibraryEnvironment is created, passing the desired settings from opts:

```
Dakota::LibraryEnvironment env(opts);
```

This standard constructor will parse the specified input and create Dakota objects. It assumes many default settings, including that the parent application initialized MPI if running in parallel mode. (In this case, Dakota will detect whether MPI was initialized and not call MPI_Init or MPI_Finalize.) For more advanced use cases described below, alternate constructors allow constructing based on MPI communicators, with delayed finalization, and with Dakota database update function callbacks. Then the application’s function evaluator implementing Dakota’s DirectApplicInterface is plugged in with a convenience function serial_interface_plugin() or parallel_interface-plugin(). Finally, the Dakota analysis is run by calling

```
env.execute();
```

The next two sections offer additional details on (1) alternative and supplementary ways to configure Dakota’s operation (Configuring Dakota operation) and (2) how to specialize Dakota’s DirectApplicInterface to provide a function evaluator plugin to Dakota (Creating a simulator plugin interface).
5.3 Configuring Dakota operation

This section describes several alternate ways to initially set and later manipulate Dakota’s configuration, including
alternatives to using a text-based input file. The algorithm configuration for a particular Dakota analysis run
is managed in its ProblemDescDB, which can be populated via an input file, string literal, or C++ API, and
later modified through Dakota’s C++ API. All Dakota objects then draw information from this database upon
instantiation.

5.3.1 Input data parsing

The simplest way for an application to configure a Dakota analysis problem is to use Dakota’s normal input
parsing system to populate its problem database (ProblemDescDB). This is done by providing standard Dakota
input file syntax through the library interface, via either a file name or string literal. An advantage is that native
Dakota syntax can be used, but disadvantages include the requirement for an additional input file beyond those
already required by the parent application and that application users also need to know Dakota syntax.

The two ways to configure Dakota via input parsing are shown near the beginning of run_dakota_mixed() in
library_mode.cpp. Here the ProgramOptions are set to either parse from a named file:

Dakota::ProgramOptions opts;
opts.input_file(dakota_input_file);

or from a string literal provided by the wrapping application:

string serial_input = "& Dakota input file ...";
opts.input_string(serial_input);

This library approach is coarse-grained in that input is parsed, objects constructed, and the environment is
immediately ready to run. The next approaches are more modular.

5.3.2 Problem database insertion

A second approach to configuring Dakota’s operation is to bypass parsing phases and directly populate the
ProblemDescDB with information on the methods, variables, interface, responses, etc., that define the Dakota
analysis problem. This approach requires more interaction with Dakota classes and data structures. However,
it can offer usability benefit when the integrating application does not want their users to interact with the full
Dakota syntax, or efficiency benefit when for example there are a large number of variables to configure.

In the direct database population approach, Dakota 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 \texttt{insert Nodes()} . An example of this approach is available in \texttt{run_dakota-data()} in \texttt{library_mode.cpp}, where the OPT++ Quasi-Newton method is configured to work on a plugin version of \texttt{text_book} or \texttt{rosenbrock}. The data objects are populated with their default values upon instantiation and are often sufficient for basic Dakota studies. Only the non-default values need to be specified. Moreover the default Dakota \texttt{Model} is a \texttt{SingleModel}, so this object need not be configured unless tailoring its configuration or using a more advanced model type. Refer to the \texttt{DataMethod}, \texttt{DataModel}, \texttt{DataVariables}, \texttt{DataInterface}, and \texttt{DataResponses} class documentation and source code for lists of attributes and their defaults. Here is an excerpt of \texttt{run_dakota-data()} that specifies the OPT++ solver after default construction of \texttt{DataMethod}:

```cpp
Dakota::DataMethod dme;
Dakota::DataMethodRep* dmr = dme.data rep();
dmr->methodName = Dakota::OPTPP_QNEWTON;
```

When using direct database population, it is critical to leave the database in an open, accessible state after initial construction. In this \texttt{run_dakota-data()} example, a flag \texttt{check_bcast_construct} is passed into the \texttt{LibraryEnvironment} constructor, indicating that it should not finalize the database and construct Dakota objects. Moreover, it is only necessary to populate the database on rank 0 of the MPI Comm on which Dakota is running. After database objects are inserted or adjusted, the \texttt{LibraryEnvironment::done_modifying_db()} function must be called before proceeding to execute. This synchronizes problem data across all ranks and constructs Dakota objects needed to run the specified analysis.

```cpp
bool check_bcast_construct = false;
Dakota::LibraryEnvironment env(MPI_COMM_WORLD, opts, check_bcast_construct);
if (rank == 0) // insert/modify DB, then lock and proceed:
    env.done_modifying_db();
env.execute();
```

### 5.3.3 Mixed mode, callbacks, and late updates

The \texttt{LibraryEnvironment} API also supports mixed approaches that combine the parsing of a Dakota input file (or input string literal) with direct database updates. This approach is motivated by large-scale applications where large vectors are cumbersome to specify in a Dakota input file or where later updates to an input template are needed. The example \texttt{run_dakota_mixed()} in \texttt{library_mode.cpp} demonstrates the combination of these more advanced approaches: (1) input text parsing, (2) database updates via a callback, (3) database updates via direct manipulation, and (4) further runtime updates to the \texttt{Model} before running.

First, a \texttt{ProgramOptions} class is instantiated and configured to parse either an input file or input string literal (as in earlier examples). The passed input data must contain all required inputs so the parser can validate them. 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 and updated later through the database API. Only the variable/response counts necessary for sizing, e.g.:

```cpp
Dakota::RealVector drv(1000, 1.); // vector of length 1000, values initialized to 1.
problem_db.set("variables.continuous_design.initial_point", drv);
```

and not the lists of values are required in this case. To update or add data after this initial parse, we use the \texttt{ProblemDescDB::set()} family of overloaded functions, e.g.
where the string identifiers are the same identifiers used when pulling information from the database using one of the get_<datatype>() functions (refer to ProblemDescDB 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.

Second, the example demonstrates a user-provided callback function which Dakota will invoke after input parsing to update ProblemDescDB. In library_mode.cpp, callback_function() is a user-provided post-parse callback that implements the type Dakota::DbCallbackFunction.

```
static void callback_function(Dakota::ProblemDescDB* db, void *ptr);
```

When Dakota calls this function it will pass back pointers to the ProblemDescDB instance and to user-provided data, so the application may convey its settings by calling methods on the ProblemDescDB, optionally using the provided data. An example of a user data structure is demonstrated in callback_data. In this case, when the LibraryEnvironment is constructed, it is constructed with the input data to initially parse, the callback function, and to leave it unlocked for further updates:

```
bool done_with_db = false;
Dakota::LibraryEnvironment env(opts, done_with_db,
callback_function, &data);
```

Third, the example demonstrates changes to the database after parsing and callback-based updates. Again, these only need happen on Dakota’s rank 0 before finalizing the DB with LibraryEnvironment::done_modifying_db(). The example demonstrates:

1. Getting access to the database through env.problem_description_db()
2. Setting the database nodes to the appropriate method through problem_db.resolve_top_method()
3. Getting data from the DB with a get string array function: problem_db.get_sa("interface.application.-analysis_drivers")
4. Setting update data with problem_db.set("variables.continuous_design.initial_point", ip);

After any of these three types updates, calling LibraryEnvironment::done_modifying_db() will broadcast any updates (including potentially large vector data and 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.)

Fourth and finally, run_dakota_mixed() demonstrates modifying a Model’s data after database operations and interface plugin are complete. This involves finding the right Model (or other class) instance to modify, and directly adjusting its data through the public API. Since the database is finalized, any updates must be performed through direct set operations on the constructed objects. For example, 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);
```
Remarks

If performing such data updates within the constructor of a DirectApplicInterface extension/derivation (see Creating a simulator plugin 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 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 database set operations at a higher level (e.g., within your main program), prior to allowing Environment to broadcast, construct, and execute, 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.:

```c++
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:

```c++
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 hard-wiring specification identifiers is undesirable, then

```c++
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. This is the approach demonstrated in run_dakota_mixed(). In each of these cases, setting list nodes unlocks the corresponding portions of the database, allowing set/get operations.

### 5.4 Creating a simulator plugin interface

The DirectApplicInterface class provides an interface for Dakota to call the simulation code directly to perform function evaluations mapping variables to responses. This provides the "back end" for Dakota to call back to the simulation framework. Two approaches to defining this direct interface are described here. The first is less common, while the second is recommended when possible.

#### 5.4.1 Extension

The first approach involves extending one of the existing DirectApplicInterface subclasses (TestDriverInterface, MatlabInterface, etc.) to support additional direct simulation interfaces. For example, Dakota algebraic test problems are implemented in TestDriverInterface. One could add additional direct functions to Dakota in TestDriverInterface::derived_map_ac(). In addition, TestDriverInterface::derived_map_if() and TestDriverInterface::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 will 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 derivation approach described in the next section should be employed.
Remarks

If the new direct evaluation function implementation will not be a member function of one of the Dakota
classes, then the following prototype should be used in order to pass the required data:

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

If the new function will be a member function, e.g., in `TestDriverInterface`, then this can be simplified to

```cpp
int sim();
```

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

5.4.2 Derivation

The second approach is to derive a new interface from `DirectApplicInterface` and redefine several virtual functions.
As demonstrated in `SIM::SerialDirectApplicInterface` and `SIM::ParallelDirectApplicInterface`, 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
}

} // namespace SIM
```

where the new derived class resides in the simulation’s namespace. Similar to the case of `Extension`, the
`DirectApplicInterface::derived_map_ac()` function is the required redefinition, and `DirectApplicInterface::derived-
map_if()` and `DirectApplicInterface::derived_map_of()` are optional.

Typically the new derived_map_ac() implementation delegates to the main simulation application for a function
evaluation. Here Dakota variables would get mapped into the simulation’s data structures, the simulation executed,
and derived response data computed for return to Dakota.

Once a derived application class is created, it must be plugged in, or registered, with the appropriate Interface
in the `LibraryEnvironment`. In MPI cases where Dakota is potentially managing concurrent evaluations of
the simulation, the plugin must be configured to run on the right MPI sub-communicator, or Dakota analysis_comm.
The simpler case is demonstrated in `serial_interface_plugin()` in `library_mode.cpp`, while a more advanced case
using the analysis communicator is shown in `parallel_interface_plugin()`.

The Dakota `LibraryEnvironment` provides a convenience function to plugin an Interface. This example will re-
place any interface found matching the given model, interface, and analysis driver with the passed plugin interface:
std::string model_type("" Ow demo: empty string will match any model type
std::string interf_type("direct");
std::string an_driver("plugin_rosenbrook");
Dakota::ProblemDescDB& problem_db = env.problem_description_db();
Dakota::Interface* serial_iface = new SIM::SerialDirectApplicInterface(problem_db);
bool plugged_in = env.plugin_interface(model_type, interf_type, an_driver, serial_iface);

The LibraryEnvironment also provides convenience functions that allow the client to iterate the lists of available interfaces or models for more advanced cases. For instance if the client knows there is only a single interface active, it could get the list of available interfaces of length 1 and plugin to the first one. In the more advanced 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

Dakota::ModelList filt_models =
  env.filtered_model_list("single", "direct", "plugin_text_book");
Dakota::ProblemDescDB& problem_db = env.problem_description_db();
Dakota::ModelIter ml_iter;
for (ml_iter = filt_models.begin(); ml_iter != filt_models.end(); ++ml_iter) {
  // set DB nodes to input specification for this Model
  problem_db.set_db_model_nodes(ml_iter->model_id());
  Dakotaa::Interface& model_interface = ml_iter->derived_interface();
  // Parallel case: plug in derived Interface object with an analysisComm.
  // Note: retrieval and passing of analysisComm is necessary only if
  // parallel operations will be performed in the derived constructor.
  // retrieve the currently active analysisComm from the Model. In the most
  // general case, need an array of Comms to cover all Model configurations.
  const MPI_Comm& analysis_comm = ml_iter->analysis_comm();
  // don’t increment ref count since no other envelope shares this letter
  model_interface.assign_rep(new SIM::ParallelDirectApplicInterface(problem_db, analysis_comm), false);
}

The file library_mode.cpp demonstrates each of these approaches. Since a Model may be used in multiple parallel contexts and may therefore have a set of parallel configurations, a more general approach would extract and pass an array of analysis communicators to allow initialization for each of the parallel configurations.

New derived direct interface instances inherit various attributes of use in configuring the simulation. In particular, the ApplicationInterface::parallelLib reference provides access to MPI communicator data (e.g., the analysis communicators discussed above), 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 re-propagated to the plug-in instance. This is the reason that the analysis communicator should be passed in to the constructor of a parallel plug-in, if the constructor will be responsible for parallel application initialization.
5.5 Retrieving data after a run

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

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

// retrieve the final response values
const Response& resp = env.response_results();
```

In the case of optimization, the final design is returned, and in the case of uncertainty quantification, the final statistics are returned. Dakota has a prototype results database, which will eventually provide better access to the results from a study.

5.6 Linking against the Dakota library

This section presumes Dakota has been configured with CMake, compiled, and installed to a CMAKE_INSTALLPREFIX using make install or equivalent. The Dakota libraries against which you must link will typically install to CMAKE_INSTALLPREFIX/bin/ and CMAKE_INSTALLPREFIX/lib/, while headers are provided in CMAKE_INSTALLPREFIX/include/. The core Dakota C and C++ code is in the library dakota-src, while Fortran code lives in the dakota-src_fortran library. Runtime libraries for any configure-enabled Dakota third-party software components (such as DOT, NPSOL, OPT++, LHS, etc.) are also installed to the lib/directory. Applications link against these Dakota libraries by specifying appropriate include and link directives.

There are two primary ways to determine the necessary Dakota-related libraries and link order for linking your application. First, when running CMake, a list of required Dakota and Dakota-included third-party libraries will be output to the console, e.g.,

```
-- Dakota_LIBRARIES: dakota_src;dakota_src_fortran;...nrid;teuchos;pecos;pecos_src;lhs;mods;mod;fftpack;...seg
```

While external dependencies will be output as:

```
-- Dakota_TPL_LIBRARIES: /usr/lib64/libcurl.so;/usr/lib64/openmpi/lib/libmpi_cxx.so;debug;/usr/lib64/libz.so;
```

Note that depending on how you configured Dakota, some libraries may be omitted from these lists (for example commercial add-ons NPSOL, DOT, and NLPQL), or additional libraries may appear.

A second option is to check which libraries appear in CMAKE_INSTALLPREFIX/bin/ CMAKE_INSTALLPREFIX/lib/, or more accurately, see the file Makefile.export.Dakota in the Dakota build/src/ or installation include/ directory. Here are some additional notes on specific libraries:

- Some Boost libraries (boost_regex, boost_filesystem, boost_system, boost_serialization) are required, and other Boost library components may be required depending on configuration, e.g., boost-signals when configuring with HAVE_ACRO.BOOL=TRUE

- System compiler and math libraries may need to be included, as may additional system libraries such as Expat and Curl, depending on how Dakota is configured.

- If configuring with graphics, you will need to add the Dakota DGraphics library and system X libraries (partial list here):

```bash
-lXpm -lXm -lXt -lXmu -lXp -lXext -lX11 -lSM -lICE
```
When configuring with AMPL (\texttt{HAVE\_AMPL\_BOOL=ON}), the AMPL solver library may require \texttt{dl}, \texttt{funcadd0.o} and \texttt{fl} libraries. We have experienced problems with the creation of \texttt{libamplsolver.a} on some platforms; use the dakota-users mailing list to get help with any problems related to this.

Optional library GSL (discouraged due to GPL license) and if linking with system-provided GSL, \texttt{gslcblas} may be needed if Dakota was configured with them.

Newmat: as of Dakota 5.2, \texttt{-lnewmat} is no longer required

Finally, it is important to use the same C++ compiler (possibly an MPI wrapper) for compiling Dakota and your application and potentially include Dakota-related preprocessor defines as emitted by CMake during compilation of Dakota and included in Makefile.export.Dakota. This ensures that the platform configuration settings are properly synchronized across Dakota and your application.
Chapter 6

Performing Function Evaluations

Performing function evaluations is one of the most critical functions of the Dakota software. It can also be one of the most complicated, as a variety of scheduling approaches and parallelism levels are supported. This complexity manifests itself in the code through a series of cascaded member functions, from the top level model evaluation functions, through various scheduling routines, to the low level details of performing a system call, fork, or direct function invocation. This section provides an overview of the primary classes and member functions involved.

### 6.1 Synchronous function evaluations

For a synchronous (i.e., blocking) mapping of parameters to responses, an iterator invokes `Model::compute_response()` to perform a function evaluation. This function is all that is seen from the iterator level, as underlying complexities are isolated. The binding of this top level function with lower level functions is as follows:

- `Model::compute_response()` utilizes `Model::derived_compute_response()` for portions of the response computation specific to derived model classes.
- `Model::derived_compute_response()` directly or indirectly invokes `Interface::map()`.
- `Interface::map()` utilizes `ApplicationInterface::derived_map()` for portions of the mapping specific to derived application interface classes.

### 6.2 Asynchronous function evaluations

For an asynchronous (i.e., nonblocking) mapping of parameters to responses, an iterator invokes `Model::asynch_compute_response()` multiple times to queue asynchronous jobs and then invokes either `Model::synchronize()` or `Model::synchronize_nowait()` to schedule the queued jobs in blocking or nonblocking fashion. Again, these functions are all that is seen from the iterator level, as underlying complexities are isolated. The binding of these top level functions with lower level functions is as follows:

- `Model::asynch_compute_response()` utilizes `Model::derived_asynch_compute_response()` for portions of the response computation specific to derived model classes.
- This derived model class function directly or indirectly invokes `Interface::map()` in asynchronous mode, which adds the job to a scheduling queue.
- `Model::synchronize()` or `Model::synchronize_nowait()` utilize `Model::derived_synchronize()` or `Model::derived_synchronize_nowait()` for portions of the scheduling process specific to derived model classes.
• These derived model class functions directly or indirectly invoke Interface::synch() or Interface::synch_nowait().

• For application interfaces, these interface synchronization functions are responsible for performing evaluation scheduling in one of the following modes: master dynamic, peer dynamic or peer static.

• NOTE: The Interface evaluation scheduling in Dakota was refactored for releases 5.4 and 6.0. Discussion of the new Interface-related functions is currently missing here.

6.3 Analyses within each function evaluation

NOTE: The Interface evaluation scheduling in Dakota was refactored for releases 5.4 and 6.0. Discussion of the new Interface-related functions for analyses is currently missing here.

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 to support synchronous local, asynchronous local, message passing, and hybrid modes. Not all of the schedulers are elevated to the ApplicationInterface level since the system call and direct function interfaces do not yet support nonblocking local analyses (and therefore support synchronous local and message passing modes, but not asynchronous local or hybrid modes). Fork interfaces, however, support all modes of analysis parallelism.
Chapter 7

Working with Variable Containers and Views

Variable views control the subset of variable types that are active and inactive within a particular iterative study. For design optimization and uncertainty quantification (UQ), for example, the active variables view consists of design or uncertain types, respectively, and any other variable types are carried along invisible to the iterative algorithm being employed. For parameter studies and design of experiments, however, a variable subset view is not imposed and all variables are active. Selected UQ methods can also be toggled into an “All” view using the active all variables input specification. When not in an All view, finer gradations within the uncertain variable sets are also relevant: probabilistic methods (reliability, stochastic expansion) view aleatory uncertain variables as active, nonprobabilistic methods (interval, evidence) view epistemic uncertain variables as active, and a few UQ methods (sampling) view both as active. In a more advanced NestedModel use case such as optimization under uncertainty, design variables are active in the outer optimization context and the uncertain variables are active in the inner UQ context, with an additional requirement on the inner UQ level to return derivatives with respect to its “inactive” variables (i.e., the design variables) for use in the outer optimization loop.

For efficiency, contiguous arrays of data store variable information for each of the domain types (continuous, discrete integer, and discrete real), but active and inactive views into them permit selecting subsets in a given context. This management is encapsulated into the Variables and SharedVariablesData classes. This page clarifies concepts of relaxed (formerly merged) vs. mixed, fine-grained vs. aggregated types, domain types, and views into contiguous arrays.

We begin with an overview of the storage and management concept, for which the following two sections describe the storage of variable values and meta-data about their organization, used in part to manage views. They are intended to communicate rationale to maintainers of Variables and SharedVariablesData classes. The final section provides a discussion of active and inactive views.

7.1 Storage in Variables

As described in the Main Page Variables, a Variables object manages variable types (design, aleatory uncertain, epistemic uncertain, and state) and domain types (continuous, discrete integer, and discrete real) and supports different approaches to either distinguishing among these types or aggregating them. Two techniques are used in cooperation to accomplish this management: (1) class specialization (RelaxedVariables or MixedVariables) and (2) views into contiguous variable arrays. The latter technique is used whenever it can satisfy the requirement, with fallback to class specialization when it cannot. In particular, aggregation or separation of variable types can be accomplished with views, but for aggregation or separation of variable domains, we must resort to class
specialization in order to relax discrete domain types. In this class specialization, a RelaxedVariables object combines continuous and discrete types (relaxing integers to reals) whereas a MixedVariables object maintains the integer/real distinction throughout.

The core data for a Variables instance is stored in a set of three contiguous arrays, corresponding to the domain types: allContinuousVars, allDiscreteIntVars, and allDiscreteRealVars, unique to each Variables instance.

Within the core variable data arrays, data corresponding to different aggregated variable types are stored in sequence for each domain type:

- continuous: [design, aleatory uncertain, epistemic uncertain, state]
- discrete integer: [design, aleatory uncertain, (epistemic uncertain), state]
- discrete real: [design, aleatory uncertain, (epistemic uncertain), state]

Note there are currently no epistemic discrete variables. This domain type ordering (continuous, discrete integer, discrete real) and aggregated variable type ordering (design, aleatory uncertain, epistemic uncertain, state) is preserved whenever distinct types are flattened into single contiguous arrays. Note that the aleatory and epistemic uncertain variables contain sub-types for different distributions (e.g., normal, uniform, histogram, poisson), and discrete integer types include both integer ranges and integer set sub-types. All sub-types are ordered according to their order of appearance in dakota.input.nspec.

When relaxing in MixedVariables, the allContinuousVars will also aggregate the discrete types, such that they contain ALL design, then ALL uncertain, then ALL state variables, each in aggregated type order; the allDiscreteIntVars and allDiscreteRealVars arrays are empty.

### 7.2 Storage in SharedVariablesData

Each Variables instance contains a reference-counted SharedVariablesData object that stores information on the variables configuration. This configuration data includes counts, types, IDs, and labels, which are often the same across many Variables instances. Thus, SharedVariablesData is intended to reduce the memory footprint by allowing the sharing of a single copy of redundant information among different Variables instances.

One of the purposes of this shared information is to support mappings between variable types, IDs, and indices into the storage arrays. Variable "types" refer to the fine-grained variable types a user would specify in an input file, as enumerated in DataVariables.hpp, e.g., CONTINUOUS, DESIGN, WEIBULL, UNCERTAIN, DISCRETE, STATE, RANGE, etc. variablesComponents is a map from these variable types to counts of how many are present.

In contrast, the variablesCompsTotals array stores total counts of each "aggregated type" (design, aleatory uncertain, epistemic uncertain, state) which might be selected to be active in a given view. Thus this array has length 12 to track the combinations of three domain type storage arrays with four possible aggregated variable types: \{continuous, discrete integer, discrete real\} x \{design, aleatory uncertain, epistemic uncertain, state\}. For example, the first entry of this array stores the number of continuous design variables, the second the number of discrete integer design (including both discrete design range and discrete design set integer types), and the last the number of discrete real state variables.

The arrays allContinuousTypes, allDiscreteIntTypes, and allDiscreteRealTypes are sized to match the corresponding core domain type storage arrays. They track the fine-grained variable type stored in that entry of the data array (since when relaxed, the continuous array may be storing data corresponding to discrete data).

Finally allContinuousIds stores the 1-based IDs of the variables stored in the allContinuousVars array, i.e., the variable number of all the problem variables considered as a single contiguous set, in aggregate type order. For relaxed (formerly merged) views, relaxedDiscreteIds stores the 1-based IDs of the variables which have been relaxed into the continuous array.
These counts, types, and IDs are most commonly used within the Model classes for mappings between variables objects at different levels of a model recursion. See, for example, the variable mappings in the NestedModel constructor.

7.3 Active and inactive views

The pair SharedVariablesDataRep::variablesView tracks the active and inactive views of the data, with values taken from the enum in DataVariables.hpp. The valid values include EMPTY and the combinations \{relaxed, mixed\} x \{all, design, aleatory uncertain, epistemic uncertain, uncertain, state\}. The ALL cases indicate aggregation of the design, aleatory uncertain, epistemic uncertain, and state types, whereas the DISTINCT cases indicate either no aggregation (design, aleatory uncertain, epistemic uncertain, state) or reduced aggregation (aleatory+epistemic uncertain). The active view is determined by the algorithm in use, managed in Variables::getView(). Any inactive view is set based on higher level iteration within a model recursion (e.g., a NestedModel), which enables lower level iteration to return derivatives with respect to variables that are active at the higher level. In the case where there is no higher level iteration, then the inactive view will remain EMPTY. It is important to stress that “inactive” at one level corresponds to active at another, and therefore the inactive set of variables should not be interpreted as the strict complement of the active set of variables; rather, active and inactive are both subsets whose union may still be a subset of the total container (more precise terminology might involve “primary” active and “secondary” active or similar). An active complement view could potentially be supported in the future, should the need arise, although this view would require management of non-contiguous portions of the aggregated arrays.

Given these groupings (views), the active and inactive subsets of the allContinuousVars, allDiscreteIntVars, and allDiscreteRealVars arrays are always contiguous, permitting vector views of the underlying data using either Teuchos::View (for numerical vectors) or Boost.MultiArray (for book-keeping arrays) views.

When a Variables envelope is constructed, its letter is initialized to either a RelaxedVariables or MixedVariables object depending on the active view. The derived classes size the contiguous storage arrays to accommodate all the problem variables, and then initialize active views into them, which could involve either subsets (DISTINCT active views) or views of the full arrays (ALL active views). Inactive views, on the other hand, are initialized during construction of a model recursion (e.g., a call to Model::inactiveView() in the NestedModel constructor). Thus, active variable subsets are always available but inactive variable subsets will be EMPTY prior to them being initialized within a Model recursion.

Accessors for continuous variables include:

- `continuous_variables()`: returns the active view which might return all (ALL views) or a subset (DISTINCT views) such as design, uncertain, only aleatory uncertain, etc.

- `inactive_continuous_variables()`: returns the inactive view which is either a subset or empty

- `all_continuous_variables()`: returns the full vector allContinuousVars

and this pattern is followed for active/inactive/all access to discrete_int_variables() and discrete_real_variables() as well as for labels, IDs, and types in SharedVariablesData and variable bounds in Constraints.

Member SurfpackApproximation::build()

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.

Member SurfpackApproximation::hessian(const Variables &vars)

Make this acceptably efficient
Chapter 8

Namespace Index

8.1 Namespace List

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

- **Dakota**
  - The primary namespace for DAKOTA ........................................... 67

- **SIM**
  - A sample namespace for derived classes that use assign_rep() to plug facilities into DAKOTA 282
# Chapter 9

## Hierarchical Index

### 9.1 Class Hierarchy

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

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActiveSet</td>
<td>283</td>
</tr>
<tr>
<td>Approximation</td>
<td>306</td>
</tr>
<tr>
<td>Approximation</td>
<td></td>
</tr>
<tr>
<td>GaussProcApproximation</td>
<td>445</td>
</tr>
<tr>
<td>PecosApproximation</td>
<td>719</td>
</tr>
<tr>
<td>SurfpackApproximation</td>
<td>823</td>
</tr>
<tr>
<td>TANA3Approximation</td>
<td>845</td>
</tr>
<tr>
<td>TaylorApproximation</td>
<td>847</td>
</tr>
<tr>
<td>APPSEvalMgr</td>
<td>317</td>
</tr>
<tr>
<td>BaseConstructor</td>
<td>322</td>
</tr>
<tr>
<td>callback_data</td>
<td>323</td>
</tr>
<tr>
<td>COLINApplication</td>
<td>323</td>
</tr>
<tr>
<td>CommandShell</td>
<td>333</td>
</tr>
<tr>
<td>Constraints</td>
<td>343</td>
</tr>
<tr>
<td>MixedVarConstraints</td>
<td>517</td>
</tr>
<tr>
<td>RelaxedVarConstraints</td>
<td>759</td>
</tr>
<tr>
<td>DataEnvironment</td>
<td>352</td>
</tr>
<tr>
<td>DataEnvironmentRep</td>
<td>353</td>
</tr>
<tr>
<td>DataInterface</td>
<td>366</td>
</tr>
<tr>
<td>DataMethod</td>
<td>367</td>
</tr>
<tr>
<td>DataMethodRep</td>
<td>368</td>
</tr>
<tr>
<td>DataModel</td>
<td>380</td>
</tr>
<tr>
<td>DataModelRep</td>
<td>381</td>
</tr>
<tr>
<td>DataResponses</td>
<td>385</td>
</tr>
<tr>
<td>DataResponsesRep</td>
<td>387</td>
</tr>
<tr>
<td>DataVariables</td>
<td>390</td>
</tr>
<tr>
<td>DataVariablesRep</td>
<td>391</td>
</tr>
<tr>
<td>DiscrepancyCorrection</td>
<td>410</td>
</tr>
<tr>
<td>JEGAOptimizer::Driver</td>
<td>417</td>
</tr>
<tr>
<td>Environment</td>
<td>426</td>
</tr>
<tr>
<td>ExecutableEnvironment</td>
<td>438</td>
</tr>
<tr>
<td>LibraryEnvironment</td>
<td>498</td>
</tr>
<tr>
<td>Class/Method/Interface</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>NomadOptimizer::Evaluator</td>
<td>430</td>
</tr>
<tr>
<td>JEGAOptimizer::Evaluator</td>
<td>432</td>
</tr>
<tr>
<td>JEGAOptimizer::EvaluatorCreator</td>
<td>437</td>
</tr>
<tr>
<td>ExperimentData</td>
<td>439</td>
</tr>
<tr>
<td>GetLongOpt</td>
<td>450</td>
</tr>
<tr>
<td>CommandLineHandler</td>
<td>332</td>
</tr>
<tr>
<td>Graphics</td>
<td>453</td>
</tr>
<tr>
<td>Interface</td>
<td>461</td>
</tr>
<tr>
<td>ApplicationInterface</td>
<td>291</td>
</tr>
<tr>
<td>DirectApplicInterface</td>
<td>406</td>
</tr>
<tr>
<td>MatlabInterface</td>
<td>500</td>
</tr>
<tr>
<td>PythonInterface</td>
<td>748</td>
</tr>
<tr>
<td>ScilabInterface</td>
<td>781</td>
</tr>
<tr>
<td>TestDriverInterface</td>
<td>848</td>
</tr>
<tr>
<td>ParallelDirectApplicInterface</td>
<td>699</td>
</tr>
<tr>
<td>SerialDirectApplicInterface</td>
<td>787</td>
</tr>
<tr>
<td>ProcessApplicInterface</td>
<td>732</td>
</tr>
<tr>
<td>ProcessHandleApplicInterface</td>
<td>736</td>
</tr>
<tr>
<td>ForkApplicInterface</td>
<td>440</td>
</tr>
<tr>
<td>SpawnApplicInterface</td>
<td>822</td>
</tr>
<tr>
<td>SysCallApplicInterface</td>
<td>842</td>
</tr>
<tr>
<td>GridApplicInterface</td>
<td>456</td>
</tr>
<tr>
<td>ApproximationInterface</td>
<td>312</td>
</tr>
<tr>
<td>Iterator</td>
<td>470</td>
</tr>
<tr>
<td>Analyzer</td>
<td>285</td>
</tr>
<tr>
<td>NonD</td>
<td>580</td>
</tr>
<tr>
<td>EfficientSubspaceMethod</td>
<td>421</td>
</tr>
<tr>
<td>NonDCalibration</td>
<td>600</td>
</tr>
<tr>
<td>NonDBayesCalibration</td>
<td>598</td>
</tr>
<tr>
<td>NonDDREAMBayesCalibration</td>
<td>605</td>
</tr>
<tr>
<td>NonDGPMSABayesCalibration</td>
<td>627</td>
</tr>
<tr>
<td>NonDQUESOBayesCalibration</td>
<td>666</td>
</tr>
<tr>
<td>NonDExpansion</td>
<td>609</td>
</tr>
<tr>
<td>NonDPolynomialChaos</td>
<td>659</td>
</tr>
<tr>
<td>NonDStochCollocation</td>
<td>680</td>
</tr>
<tr>
<td>NonDIntegration</td>
<td>631</td>
</tr>
<tr>
<td>NonDCubature</td>
<td>603</td>
</tr>
<tr>
<td>NonDQuadrature</td>
<td>663</td>
</tr>
<tr>
<td>NonDSparseGrid</td>
<td>677</td>
</tr>
<tr>
<td>NonDInterval</td>
<td>634</td>
</tr>
<tr>
<td>NonDGlobalInterval</td>
<td>617</td>
</tr>
<tr>
<td>NonDGlobalEvidence</td>
<td>615</td>
</tr>
<tr>
<td>NonDGlobalSingleInterval</td>
<td>622</td>
</tr>
<tr>
<td>NonDLHSInterval</td>
<td>637</td>
</tr>
<tr>
<td>NonDLHSEvidence</td>
<td>636</td>
</tr>
<tr>
<td>NonDLHSSingleInterval</td>
<td>641</td>
</tr>
<tr>
<td>NonDLocalInterval</td>
<td>643</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>NonDLocalEvidence</td>
<td>642</td>
</tr>
<tr>
<td>NonDLocalSingleInterval</td>
<td>655</td>
</tr>
<tr>
<td>NonDPOFDarts</td>
<td>657</td>
</tr>
<tr>
<td>NonDReliability</td>
<td>669</td>
</tr>
<tr>
<td>NonDGlobalReliability</td>
<td>620</td>
</tr>
<tr>
<td>NonDLocalReliability</td>
<td>646</td>
</tr>
<tr>
<td>NonDSampling</td>
<td>671</td>
</tr>
<tr>
<td>NonDAdaptImpSampling</td>
<td>590</td>
</tr>
<tr>
<td>NonDAdaptiveSampling</td>
<td>594</td>
</tr>
<tr>
<td>NonDGPImpSampling</td>
<td>624</td>
</tr>
<tr>
<td>NonDIncrementLHSSampling</td>
<td>630</td>
</tr>
<tr>
<td>NonDLHSSampling</td>
<td>639</td>
</tr>
<tr>
<td>PStudyDACE</td>
<td>744</td>
</tr>
<tr>
<td>DDACEDesignCompExp</td>
<td>402</td>
</tr>
<tr>
<td>FSUDesignCompExp</td>
<td>442</td>
</tr>
<tr>
<td>ParamStudy</td>
<td>714</td>
</tr>
<tr>
<td>PSUADEDesignCompExp</td>
<td>746</td>
</tr>
<tr>
<td>Verification</td>
<td>863</td>
</tr>
<tr>
<td>RichExtrapVerification</td>
<td>779</td>
</tr>
<tr>
<td>MetaIterator</td>
<td>502</td>
</tr>
<tr>
<td>CollabHybridMetaIterator</td>
<td>330</td>
</tr>
<tr>
<td>ConcurrentMetaIterator</td>
<td>334</td>
</tr>
<tr>
<td>EmbedHybridMetaIterator</td>
<td>425</td>
</tr>
<tr>
<td>SeqHybridMetaIterator</td>
<td>784</td>
</tr>
<tr>
<td>Minimizer</td>
<td>503</td>
</tr>
<tr>
<td>LeastSq</td>
<td>494</td>
</tr>
<tr>
<td>NL2SOLLeastSq</td>
<td>568</td>
</tr>
<tr>
<td>NLSSOLLeastSq</td>
<td>575</td>
</tr>
<tr>
<td>SNLLLeastSq</td>
<td>810</td>
</tr>
<tr>
<td>Optimizer</td>
<td>690</td>
</tr>
<tr>
<td>APPSOptimizer</td>
<td>320</td>
</tr>
<tr>
<td>COLINOptimizer</td>
<td>326</td>
</tr>
<tr>
<td>CONMINOptimizer</td>
<td>336</td>
</tr>
<tr>
<td>DOTOptimizer</td>
<td>413</td>
</tr>
<tr>
<td>JEGAOptimizer</td>
<td>488</td>
</tr>
<tr>
<td>NCSUOptimizer</td>
<td>553</td>
</tr>
<tr>
<td>NLPQLPOptimizer</td>
<td>571</td>
</tr>
<tr>
<td>NomadOptimizer</td>
<td>578</td>
</tr>
<tr>
<td>NonlinearCGOptimizer</td>
<td>682</td>
</tr>
<tr>
<td>NPSOLOptimizer</td>
<td>685</td>
</tr>
<tr>
<td>OptDartsOptimizer</td>
<td>687</td>
</tr>
<tr>
<td>SNLLOptimizer</td>
<td>813</td>
</tr>
<tr>
<td>SurrBasedMinimizer</td>
<td>834</td>
</tr>
<tr>
<td>EffGlobalMinimizer</td>
<td>419</td>
</tr>
<tr>
<td>SurrBasedGlobalMinimizer</td>
<td>826</td>
</tr>
<tr>
<td>SurrBasedLocalMinimizer</td>
<td>827</td>
</tr>
<tr>
<td>IteratorScheduler</td>
<td>483</td>
</tr>
<tr>
<td>Model</td>
<td>520</td>
</tr>
<tr>
<td>Class Name</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>NestedModel</td>
<td>556</td>
</tr>
<tr>
<td>RecastModel</td>
<td>751</td>
</tr>
<tr>
<td>SingleModel</td>
<td>805</td>
</tr>
<tr>
<td>SurrogateModel</td>
<td>838</td>
</tr>
<tr>
<td>DataFitSurfModel</td>
<td>355</td>
</tr>
<tr>
<td>HierarchSurfModel</td>
<td>457</td>
</tr>
<tr>
<td>MPIManager</td>
<td>548</td>
</tr>
<tr>
<td>MPIPackBuffer</td>
<td>549</td>
</tr>
<tr>
<td>MPIUnpackBuffer</td>
<td>551</td>
</tr>
<tr>
<td>NL2Res</td>
<td>568</td>
</tr>
<tr>
<td>NoDBBaseConstructor</td>
<td>577</td>
</tr>
<tr>
<td>OutputManager</td>
<td>695</td>
</tr>
<tr>
<td>ParallelConfiguration</td>
<td>697</td>
</tr>
<tr>
<td>ParallelLevel</td>
<td>700</td>
</tr>
<tr>
<td>ParallelLibrary</td>
<td>702</td>
</tr>
<tr>
<td>ParamResponsePair</td>
<td>711</td>
</tr>
<tr>
<td>partial_prp_equality</td>
<td>718</td>
</tr>
<tr>
<td>partial_prp_hash</td>
<td>719</td>
</tr>
<tr>
<td>ProblemDescDB</td>
<td>724</td>
</tr>
<tr>
<td>NIDRProblemDescDB</td>
<td>564</td>
</tr>
<tr>
<td>ProgramOptions</td>
<td>739</td>
</tr>
<tr>
<td>RecastBaseConstructor</td>
<td>750</td>
</tr>
<tr>
<td>Response</td>
<td>762</td>
</tr>
<tr>
<td>ResponseRep</td>
<td>766</td>
</tr>
<tr>
<td>ResultsDBAny</td>
<td>772</td>
</tr>
<tr>
<td>ResultsEntry&lt;StoredType&gt;</td>
<td>774</td>
</tr>
<tr>
<td>ResultsID</td>
<td>775</td>
</tr>
<tr>
<td>ResultsManager</td>
<td>776</td>
</tr>
<tr>
<td>ResultsNames</td>
<td>778</td>
</tr>
<tr>
<td>SensAnalysisGlobal</td>
<td>782</td>
</tr>
<tr>
<td>SharedApproxData</td>
<td>789</td>
</tr>
<tr>
<td>SharedPecosApproxData</td>
<td>793</td>
</tr>
<tr>
<td>SharedSurfpackApproxData</td>
<td>796</td>
</tr>
<tr>
<td>SharedVariablesData</td>
<td>798</td>
</tr>
<tr>
<td>SharedVariablesDataRep</td>
<td>802</td>
</tr>
<tr>
<td>SNLLBase</td>
<td>807</td>
</tr>
<tr>
<td>SNLLLeastSq</td>
<td>810</td>
</tr>
<tr>
<td>SNLLOptimizer</td>
<td>813</td>
</tr>
<tr>
<td>SOLBase</td>
<td>819</td>
</tr>
<tr>
<td>NLSSOLLastSq</td>
<td>575</td>
</tr>
<tr>
<td>NPSOLOptimizer</td>
<td>685</td>
</tr>
<tr>
<td>TrackerHTTP</td>
<td>853</td>
</tr>
<tr>
<td>Variables</td>
<td>855</td>
</tr>
<tr>
<td>MixedVariables</td>
<td>518</td>
</tr>
<tr>
<td>RelaxedVariables</td>
<td>760</td>
</tr>
</tbody>
</table>
Chapter 10

Class Index

10.1 Class List

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

- **ActiveSet**
  Container class for active set tracking information. Contains the active set request vector and the derivative variables vector . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 283

- **Analyzer**
  Base class for NonD, DACE, and ParamStudy branches of the iterator hierarchy . . . . . . 285

- **ApplicationInterface**
  Derived class within the interface class hierarchy for supporting interfaces to simulation codes 291

- **Approximation**
  Base class for the approximation class hierarchy . . . . . . . . . . . . . . . . . . . . . . . . 306

- **ApproximationInterface**
  Derived class within the interface class hierarchy for supporting approximations to simulation-based results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 312

- **APPSEvalMgr**
  Evaluation manager class for APPSPACK . . . . . . . . . . . . . . . . . . . . . . . . . . . 317

- **APPSOptimizer**
  Wrapper class for HOPSPACK . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 320

- **BaseConstructor**
  Dummy struct for overloading letter-envelope constructors . . . . . . . . . . . . . . . . . . . 322

- **callback_data**
  Dumb struct for overloading letter-envelope constructors . . . . . . . . . . . . . . . . . . . 322

- **COLINApplication**
  Utility class for managing command line inputs to DAKOTA . . . . . . . . . . . . . . . . . . 323

- **COLINOptimizer**
  Wrapper class for optimizers defined using COLIN . . . . . . . . . . . . . . . . . . . . . . . . 326

- **CollabHybridMetaIterator**
  Meta-iterator for hybrid iteration using multiple collaborating optimization and nonlinear least squares methods . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 330

- **CommandLineHandler**
  Utility class for managing command line inputs to DAKOTA . . . . . . . . . . . . . . . . . . . 332

- **CommandShell**
  Utility class which defines convenience operators for spawning processes with system calls . 333

- **ConcurrentMetaIterator**
  Meta-iterator for multi-start iteration or pareto set optimization . . . . . . . . . . . . . . . . . 334
CHAPTER 10. CLASS INDEX

CONMINOptimizer
  Wrapper class for the CONMIN optimization library ........................................ 336

Constraints
  Base class for the variable constraints class hierarchy .................................. 343

DataEnvironment
  Handle class for environment specification data ............................................ 352

DataEnvironmentRep
  Body class for environment specification data ............................................. 353

DataFitSurrModel
  Derived model class within the surrogate model branch for managing data fit surrogates (global and local) ................................................................. 355

DataInterface
  Handle class for interface specification data .............................................. 366

DataMethod
  Handle class for method specification data ............................................... 367

DataMethodRep
  Body class for method specification data ................................................ 368

DataModel
  Handle class for model specification data ................................................. 380

DataModelRep
  Body class for model specification data ................................................ 381

DataResponses
  Handle class for responses specification data ......................................... 385

DataResponsesRep
  Body class for responses specification data ............................................. 387

DataVariables
  Handle class for variables specification data ........................................... 390

DataVariablesRep
  Body class for variables specification data ............................................ 391

DDACEDesignCompExp
  Wrapper class for the DDACE design of experiments library .......................... 402

DirectApplicInterface
  Derived application interface class which spawns simulation codes and testers using direct procedure calls ................................................................. 406

DiscrepancyCorrection
  Base class for discrepancy corrections ..................................................... 410

DOTOptimizer
  Wrapper class for the DOT optimization library ........................................ 413

JEGAOptimizer::Driver
  A subclass of the JEGA front end driver that exposes the individual protected methods to execute the algorithm ................................................................. 417

EffGlobalMinimizer
  Implementation of Efficient Global Optimization/Least Squares algorithms .... 419

EfficientSubspaceMethod
  Efficient Subspace Method (ESM), as proposed by Hany S. Abdel-Khalik ........... 421

EmbedHybridMetaIterator
  Meta-iterator for closely-coupled hybrid iteration, typically involving the embedding of local search methods within global search methods ..................... 425
### 10.1. CLASS LIST

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Environment</strong></td>
<td>Base class for the environment class hierarchy</td>
</tr>
<tr>
<td><strong>NomadOptimizer::Evaluator</strong></td>
<td>NOMAD-based Evaluator class</td>
</tr>
<tr>
<td><strong>JEGAOptimizer::Evaluator</strong></td>
<td>An evaluator specialization that knows how to interact with Dakota</td>
</tr>
<tr>
<td><strong>JEGAOptimizer::EvaluatorCreator</strong></td>
<td>A specialization of the JEGA::FrontEnd::EvaluatorCreator that creates a new instance of a Evaluator</td>
</tr>
<tr>
<td><strong>ExecutableEnvironment</strong></td>
<td>Environment corresponding to execution as a stand-alone application</td>
</tr>
<tr>
<td><strong>ExperimentData</strong></td>
<td></td>
</tr>
<tr>
<td><strong>ForkApplicInterface</strong></td>
<td>Derived application interface class which spawns simulation codes using fork/execvp/waitpid</td>
</tr>
<tr>
<td><strong>FSUDesignCompExp</strong></td>
<td>Wrapper class for the FSUDace QMC/CVT library</td>
</tr>
<tr>
<td><strong>GaussProcApproximation</strong></td>
<td>Derived approximation class for Gaussian Process implementation</td>
</tr>
<tr>
<td><strong>GetLongOpt</strong></td>
<td>GetLongOpt is a general command line utility from S. Manoharan (Advanced Computer Research Institute, Lyon, France)</td>
</tr>
<tr>
<td><strong>Graphics</strong></td>
<td>Single interface to 2D (motif) and 3D (PLPLOT) graphics as well as tabular cataloguing of data for post-processing with Matlab, Tecplot, etc</td>
</tr>
<tr>
<td><strong>GridApplicInterface</strong></td>
<td>Derived application interface class which spawns simulation codes using grid services such as Condor or Globus</td>
</tr>
<tr>
<td><strong>HierarchSurrModel</strong></td>
<td>Derived model class within the surrogate model branch for managing hierarchical surrogates (models of varying fidelity)</td>
</tr>
<tr>
<td><strong>Interface</strong></td>
<td>Base class for the interface class hierarchy</td>
</tr>
<tr>
<td><strong>Iterator</strong></td>
<td>Base class for the iterator class hierarchy</td>
</tr>
<tr>
<td><strong>IteratorScheduler</strong></td>
<td>Environment corresponding to execution as a stand-alone application</td>
</tr>
<tr>
<td><strong>JEGAOptimizer</strong></td>
<td>A version of Dakota::Optimizer for instantiation of John Eddy’s Genetic Algorithms (JEGA)</td>
</tr>
<tr>
<td><strong>LeastSq</strong></td>
<td>Base class for the nonlinear least squares branch of the iterator hierarchy</td>
</tr>
<tr>
<td><strong>LibraryEnvironment</strong></td>
<td>Environment corresponding to execution as an embedded library</td>
</tr>
<tr>
<td><strong>MatlabInterface</strong></td>
<td></td>
</tr>
<tr>
<td><strong>MetaIterator</strong></td>
<td>Base class for meta-iterator</td>
</tr>
<tr>
<td><strong>Minimizer</strong></td>
<td>Base class for the optimizer and least squares branches of the iterator hierarchy</td>
</tr>
</tbody>
</table>
MixedVarConstraints
Derived class within the Constraints hierarchy which separates continuous and discrete variables (no domain type array merging) ................................................................. 517

MixedVariables
Derived class within the Variables hierarchy which separates continuous and discrete variables (no domain type array merging) ................................................................. 518

Model
Base class for the model class hierarchy ................................................................. 520

MPIManager
Class MPIManager to manage Dakota’s MPI world, which may be a subset of MPI_COMM_WORLD ................................................................. 548

MPIPackBuffer
Class for packing MPI message buffers ................................................................. 549

MPIUnpackBuffer
Class for unpacking MPI message buffers ................................................................. 551

NCSUOptimizer
Wrapper class for the NCSU DIRECT optimization library ................................................................. 553

NestedModel
Derived model class which performs a complete sub-iterator execution within every evaluation of the model ................................................................. 556

NIDRProblemDescDB
The derived input file database utilizing the new IDR parser ................................................................. 564

NL2Res
Auxiliary information passed to calcr and calcj via ur ................................................................. 568

NL2SOLLeastSq
Wrapper class for the NL2SOL nonlinear least squares library ................................................................. 568

NLPQLPOptimizer
Wrapper class for the NLPQLP optimization library, Version 2.0 ................................................................. 571

NLSSOLLeastSq
Wrapper class for the NLSSOL nonlinear least squares library ................................................................. 575

NoDBBaseConstructor
Dummy struct for overloading constructors used in on-the-fly instantiations ................................................................. 577

NomadOptimizer
Wrapper class for NOMAD Optimizer ................................................................. 578

NonD
Base class for all nondeterministic iterators (the DAKOTA/UQ branch) ................................................................. 580

NonDAdaptImpSampling
Class for the Adaptive Importance Sampling methods within DAKOTA ................................................................. 590

NonDAdaptiveSampling
Class for testing various Adaptively sampling methods using geometric, statistical, and topological information of the surrogate ................................................................. 594

NonDBayesCalibration
Base class for Bayesian inference: generates posterior distribution on model parameters given experimental data ................................................................. 598

NonDCalibration ................................................................. 600

NonDCubature
Derived nondeterministic class that generates N-dimensional numerical cubature points for evaluation of expectation integrals ................................................................. 603
<table>
<thead>
<tr>
<th>NonDDREAMBayesCalibration</th>
<th>Bayesian inference using the DREAM approach ............................................ 605</th>
</tr>
</thead>
<tbody>
<tr>
<td>NonDExpansion</td>
<td>Base class for polynomial chaos expansions (PCE) and stochastic collocation (SC) 609</td>
</tr>
<tr>
<td>NonDGlobalEvidence</td>
<td>Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ ........... 615</td>
</tr>
<tr>
<td>NonDGlobalInterval</td>
<td>Class for using global nongradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification ........................................... 617</td>
</tr>
<tr>
<td>NonDGlobalReliability</td>
<td>Class for global reliability methods within DAKOTA/UQ ................................... 620</td>
</tr>
<tr>
<td>NonDGlobalSingleInterval</td>
<td>Class for using global nongradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification ........................................... 622</td>
</tr>
<tr>
<td>NonDGPImpSampling</td>
<td>Class for the Gaussian Process-based Importance Sampling method ......................... 624</td>
</tr>
<tr>
<td>NonDGPMBayesCalibration</td>
<td>Generates posterior distribution on model parameters given experiment data ........... 627</td>
</tr>
<tr>
<td>NonDIntegration</td>
<td>Derived nondeterministic class that generates N-dimensional numerical integration points for evaluation of expectation integrals ........................................... 631</td>
</tr>
<tr>
<td>NonDInterval</td>
<td>Base class for interval-based methods within DAKOTA/UQ .................................. 634</td>
</tr>
<tr>
<td>NonDLHSEvidence</td>
<td>Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ ........... 636</td>
</tr>
<tr>
<td>NonDLHSInterval</td>
<td>Class for the LHS-based interval methods within DAKOTA/UQ ................................. 637</td>
</tr>
<tr>
<td>NonDLHSSampling</td>
<td>Performs LHS and Monte Carlo sampling for uncertainty quantification .................... 639</td>
</tr>
<tr>
<td>NonDLHSSingleInterval</td>
<td>Class for pure interval propagation using LHS .................................................. 641</td>
</tr>
<tr>
<td>NonDLocalEvidence</td>
<td>Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ ........... 642</td>
</tr>
<tr>
<td>NonDLocalInterval</td>
<td>Class for using local gradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification ........................................... 643</td>
</tr>
<tr>
<td>NonDLocalReliability</td>
<td>Class for the reliability methods within DAKOTA/UQ ......................................... 646</td>
</tr>
<tr>
<td>NonDLocalSingleInterval</td>
<td>Class for using local gradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification ........................................... 655</td>
</tr>
<tr>
<td>NonDPOFDarts</td>
<td>Base class for POF Dart methods within DAKOTA/UQ ......................................... 657</td>
</tr>
<tr>
<td>NonDPolynomialChaos</td>
<td>Nonintrusive polynomial chaos expansion approaches to uncertainty quantification .......... 659</td>
</tr>
</tbody>
</table>
NonDQuadrature
Derived nondeterministic class that generates N-dimensional numerical quadrature points for evaluation of expectation integrals over uncorrelated standard normals/uniforms/exponentials/betas/gammas

NonDQUESOBayesCalibration
Bayesian inference using the QUESO library from UT Austin

NonDReliability
Base class for the reliability methods within DAKOTA/UQ

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

NonDSparseGrid
Derived nondeterministic class that generates N-dimensional Smolyak sparse grids for numerical evaluation of expectation integrals over independent standard random variables

NonDStochCollocation
Nonintrusive stochastic collocation approaches to uncertainty quantification

NonlinearCGOptimizer
Wrapper class for the NPSOL optimization library

OptDartsOptimizer
Wrapper class for OptDarts Optimizer

Optimizer
Base class for the optimizer branch of the iterator hierarchy

OutputManager
Class to manage redirection of stdout/stderr, and keep track of current redir state, and manage rank 0 output

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

ParallelDirectApplicInterface
Sample derived interface class for testing parallel simulator plug-ins using assign_rep()

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

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

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

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

partial_prp_equality
Predicate for comparing ONLY the interfaceId and Vars attributes of PRPair

partial_prp_hash
Wrapper to delegate to the ParamResponsePair hash_value function

PecosApproximation
Derived approximation class for global basis polynomials

ProblemDescDB
The database containing information parsed from the DAKOTA input file
10.1. CLASS LIST

ProcessApplicInterface
Derived application interface class that spawns a simulation code using a separate process and communicates with it through files .............................................................. 732

ProcessHandleApplicInterface
Derived application interface class that spawns a simulation code using a separate process, receives a process identifier, and communicates with the spawned process through files ...... 736

ProgramOptions
ProgramOptions stores options whether from the CLH or from library user; initially valid only on worldRank = 0, but then broadcast in ParallelLibrary::manage_outputs_restart ........... 739

PStudyDACE
Base class for managing common aspects of parameter studies and design of experiments methods .......................................................... 744

PSUADEDesignCompExp
Wrapper class for the PSUADE library ................................................ 746

PythonInterface ................................................................. 748

RecastBaseConstructor
Dummy struct for overloading constructors used in on-the-fly Model instantiations ...... 750

RecastModel
Derived model class which provides a thin wrapper around a sub-model in order to recast the form of its inputs and/or outputs .......................................................... 751

RelaxedVarConstraints
Derived class within the Constraints hierarchy which employs relaxation of discrete variables 759

RelaxedVariables
Derived class within the Variables hierarchy which employs the relaxation of discrete variables 760

Response
Container class for response functions and their derivatives. Response provides the handle class .................................................................................. 762

ResponseRep
Container class for response functions and their derivatives. ResponseRep provides the body class ............................................................................. 766

ResultsDBAny ................................................................. 772

ResultsEntry<StoredType>
Class to manage in-core vs. file database lookups ........................................ 774

ResultsID
Get a globally unique 1-based execution number for a given iterator name (combination of methodName and methodID) for use in results DB. Each Iterator::run() call creates or increments this count for its string identifier ........................................................................... 775

ResultsManager
Results manager for iterator final data .......................................................... 776

ResultsNames
List of valid names for iterator results .......................................................... 778

RichExtrapVerification
Class for Richardson extrapolation for code and solution verification .................. 779

ScilabInterface ................................................................. 781

SensAnalysisGlobal
Class for a utility class containing correlation calculations and variance-based decomposition 782

SeqHybridMetaIterator
Method for sequential hybrid iteration using multiple optimization and nonlinear least squares methods on multiple models of varying fidelity ........................................ 784
<table>
<thead>
<tr>
<th>Class Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SerialDirectApplicInterface</td>
<td>Sample derived interface class for testing serial simulator plug-ins using assign_rep()</td>
<td>787</td>
</tr>
<tr>
<td>SharedApproxData</td>
<td>Base class for the shared approximation data class hierarchy</td>
<td>789</td>
</tr>
<tr>
<td>SharedPecosApproxData</td>
<td>Derived approximation class for global basis polynomials</td>
<td>793</td>
</tr>
<tr>
<td>SharedSurfpackApproxData</td>
<td>Derived approximation class for Surfpack approximation classes. Interface between Surfpack and Dakota</td>
<td>796</td>
</tr>
<tr>
<td>SharedVariablesData</td>
<td>Container class encapsulating variables data that can be shared among a set of Variables instances</td>
<td>798</td>
</tr>
<tr>
<td>SharedVariablesDataRep</td>
<td>The representation of a SharedVariablesData instance. This representation, or body, may be shared by multiple SharedVariablesData handle instances</td>
<td>802</td>
</tr>
<tr>
<td>SingleModel</td>
<td>Derived model class which utilizes a single interface to map variables into responses</td>
<td>805</td>
</tr>
<tr>
<td>SNLLBase</td>
<td>Base class for OPT++ optimization and least squares methods</td>
<td>807</td>
</tr>
<tr>
<td>SNLLLeastSq</td>
<td>Wrapper class for the OPT++ optimization library</td>
<td>810</td>
</tr>
<tr>
<td>SNLLOptimizer</td>
<td>Wrapper class for the OPT++ optimization library</td>
<td>813</td>
</tr>
<tr>
<td>SOLBase</td>
<td>Base class for Stanford SOL software</td>
<td>819</td>
</tr>
<tr>
<td>SpawnApplicInterface</td>
<td>Derived application interface class which spawns simulation codes using spawnvp</td>
<td>822</td>
</tr>
<tr>
<td>SurfpackApproximation</td>
<td>Derived approximation class for Surfpack approximation classes. Interface between Surfpack and Dakota</td>
<td>823</td>
</tr>
<tr>
<td>SurrBasedGlobalMinimizer</td>
<td>The global surrogate-based minimizer which sequentially minimizes and updates a global surrogate model without trust region controls</td>
<td>826</td>
</tr>
<tr>
<td>SurrBasedLocalMinimizer</td>
<td>Class for provably-convergent local surrogate-based optimization and nonlinear least squares</td>
<td>827</td>
</tr>
<tr>
<td>SurrBasedMinimizer</td>
<td>Base class for local/global surrogate-based optimization/least squares</td>
<td>834</td>
</tr>
<tr>
<td>SurrogateModel</td>
<td>Base class for surrogate models (DataFitSurrModel and HierarchSurrModel)</td>
<td>838</td>
</tr>
<tr>
<td>SysCallApplicInterface</td>
<td>Derived application interface class which spawns simulation codes using system calls</td>
<td>842</td>
</tr>
<tr>
<td>TANA3Approximation</td>
<td>Derived approximation class for TANA-3 two-point exponential approximation (a multipoint approximation)</td>
<td>845</td>
</tr>
<tr>
<td>TaylorApproximation</td>
<td>Derived approximation class for first- or second-order Taylor series (a local approximation)</td>
<td>847</td>
</tr>
<tr>
<td>TestDriverInterface</td>
<td></td>
<td>848</td>
</tr>
<tr>
<td>TrackerHTTP</td>
<td>TrackerHTTP: a usage tracking module that uses HTTP/HTTPS via the curl library</td>
<td>853</td>
</tr>
</tbody>
</table>
### Class List

**Variables**
- Base class for the variables class hierarchy ......................................................... 855

**Verification**
- Base class for managing common aspects of verification studies ....................... 863
Chapter 11

File Index

11.1 File List

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

- **dakota_dll_api.cpp**
  This file contains a DakotaRunner class, which launches DAKOTA .................................. 865
- **dakota_dll_api.h**
  API for DLL interactions ................................................................. 866
- **dakota_tabular_io.hpp**
  Utility functions for reading and writing tabular data files Emerging utilities for tabular file
  I/O. For now, just extraction of capability from separate contexts to facilitate rework. These
  augment (and leverage) those in data_util.h ........................................... 867
- **dll_tester.cpp**
  Test the DLL with a DAKOTA input file ................................................. 868
- **JEGAOptimizer.cpp**
  Contains the implementation of the JEGAOptimizer class ............................ 869
- **JEGAOptimizer.hpp**
  Contains the definition of the JEGAOptimizer class ..................................... 869
- **library_mode.cpp**
  File containing a mock simulator main for testing Dakota in library mode ............ 870
- **library_split.cpp**
  File containing a mock simulator main for testing DAKOTA in library mode on a split com-
  municator ................................................................................................. 873
- **main.cpp**
  File containing the main program for DAKOTA ............................................. 874
- **restart_util.cpp**
  File containing the DAKOTA restart utility main program ............................... 875
Chapter 12

Namespace Documentation

12.1 Dakota Namespace Reference

The primary namespace for DAKOTA.

Classes

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

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

- class APPSEvalMgr
  
  Evaluation manager class for APPSPACK.

- class APPSOptimizer
  
  Wrapper class for HOPSPACK.

- class COLINAApplication

- class COLINOptimizer
  
  Wrapper class for optimizers defined using COLIN.

- class CollabHybridMetaIterator
  
  Meta-iterator for hybrid iteration using multiple collaborating optimization and nonlinear least squares methods.

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

- class CommandLineHandler
  
  Utility class for managing command line inputs to DAKOTA.

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

- class ConcurrentMetaIterator
  
  Meta-iterator for multi-start iteration or pareto set optimization.

- class CONMINOptimizer
  
  Wrapper class for the CONMIN optimization library.

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

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

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

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

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

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

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

- **class Environment**
  Base class for the environment class hierarchy.

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

- **class Interface**
  Base class for the interface class hierarchy.

- **class Iterator**
  Base class for the iterator class hierarchy.

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

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

- **class Model**
  Base class for the model class hierarchy.

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

- **class Optimizer**
  Base class for the optimizer branch of the iterator hierarchy.

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

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

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

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

- **class Verification**
  Base class for managing common aspects of verification studies.
• class **DataEnvironmentRep**
  Body class for environment specification data.
• class **DataEnvironment**
  Handle class for environment specification data.
• class **DataFitSurrModel**
  Derived model class within the surrogate model branch for managing data fit surrogates (global and local)
• class **DataInterface**
  Handle class for interface specification data.
• class **DataMethodRep**
  Body class for method specification data.
• class **DataMethod**
  Handle class for method specification data.
• class **DataModelRep**
  Body class for model specification data.
• class **DataModel**
  Handle class for model specification data.
• class **DataResponsesRep**
  Body class for responses specification data.
• class **DataResponses**
  Handle class for responses specification data.
• class **DataVariablesRep**
  Body class for variables specification data.
• class **DataVariables**
  Handle class for variables specification data.
• class **DDACEDesignCompExp**
  Wrapper class for the DDACE design of experiments library.
• class **DirectApplicInterface**
  Derived application interface class which spawns simulation codes and testers using direct procedure calls.
• class **DiscrepancyCorrection**
  Base class for discrepancy corrections.
• class **DOTOptimizer**
  Wrapper class for the DOT optimization library.
• class **EffGlobalMinimizer**
  Implementation of Efficient Global Optimization/Least Squares algorithms.
• class **EfficientSubspaceMethod**
  Efficient Subspace Method (ESM), as proposed by Hany S. Abdel-Khalik.
• class **EmbedHybridMetaIterator**
  Meta-iterator for closely-coupled hybrid iteration, typically involving the embedding of local search methods within global search methods.
• class **ExecutableEnvironment**
  Environment corresponding to execution as a stand-alone application.
• class **ExperimentData**
• class **ForkApplicInterface**
Derived application interface class which spawns simulation codes using fork/execvp/waitpid.

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

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

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

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

- **class IteratorScheduler**
  Environment corresponding to execution as a stand-alone application.

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

- **class LibraryEnvironment**
  Environment corresponding to execution as an embedded library.

- **class MatlabInterface**
- **class MetaIterator**
  Base class for meta-iterators.

- **class MixedVarConstraints**
  Derived class within the Constraints hierarchy which separates continuous and discrete variables (no domain type array merging).

- **class MixedVariables**
  Derived class within the Variables hierarchy which separates continuous and discrete variables (no domain type array merging).

- **class MPIManager**
  Class MPIManager to manage Dakota’s MPI world, which may be a subset of MPI_COMM_WORLD.

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

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

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

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

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

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

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

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

- **class NLSSOLLeastSq**
12.1. **DAKOTA NAMESPACE REFERENCE**

Wrapper class for the NLSSOL nonlinear least squares library.

- **class NomadOptimizer**
  Wrapper class for NOMAD Optimizer.

- **class NonDAAdaptImpSampling**
  Class for the Adaptive Importance Sampling methods within DAKOTA.

- **class NonDAAdaptiveSampling**
  Class for testing various Adaptively sampling methods using geometric, statistical, and topological information of the surrogate.

- **class NonDBayesCalibration**
  Base class for Bayesian inference: generates posterior distribution on model parameters given experimental data.

- **class NonDCalibration**
- **class NonDCubature**
  Derived nondeterministic class that generates N-dimensional numerical cubature points for evaluation of expectation integrals.

- **class NonDDREAMBayesCalibration**
  Bayesian inference using the DREAM approach.

- **class NonDExpansion**
  Base class for polynomial chaos expansions (PCE) and stochastic collocation (SC)

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

- **class NonDGGlobalInterval**
  Class for using global nongradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification.

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

- **class NonDGGlobalSingleInterval**
  Class for using global nongradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification.

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

- **class NonDGPMsABayesCalibration**
  Generates posterior distribution on model parameters given experimental data.

- **class NonDIncLHSSampling**
  Performs incremental LHS sampling for uncertainty quantification.

- **class NonDIntegration**
  Derived nondeterministic class that generates N-dimensional numerical integration points for evaluation of expectation integrals.

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

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

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

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

- **class NonDLHSSingleInterval**
  
  Class for pure interval propagation using LHS.

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

- **class NonDLocalInterval**
  
  Class for using local gradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification.

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

- **class NonDLocalSingleInterval**
  
  Class for using local gradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification.

- **class NonDPOFDarts**
  
  Base class for POF Dart methods within DAKOTA/UQ.

- **class NonDPolynomialChaos**
  
  Nonintrusive polynomial chaos expansion approaches to uncertainty quantification.

- **class NonDQuadrature**
  
  Derived nondeterministic class that generates N-dimensional numerical quadrature points for evaluation of expectation integrals over uncorrelated standard normals/uniforms/exponentials/betas/gammas.

- **class NonDQUESOBayesCalibration**
  
  Bayesian inference using the QUESO library from UT Austin.

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

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

- **class NonDSparseGrid**
  
  Derived nondeterministic class that generates N-dimensional Smolyak sparse grids for numerical evaluation of expectation integrals over independent standard random variables.

- **class NonDStochCollocation**
  
  Nonintrusive stochastic collocation approaches to uncertainty quantification.

- **class NonlinearCGOptimizer**

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

- **class OptDartsOptimizer**
  
  Wrapper class for OptDarts Optimizer.

- **class OutputManager**
  
  Class to manage redirection of stdout/stderr, and keep track of current redir state, and manage rank 0 output.

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

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

- **class ParallelLibrary**
Class for partitioning multiple levels of parallelism and managing message passing within these levels.

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

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

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

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

- **class** ProcessApplicInterface
  Derived application interface class that spawns a simulation code using a separate process and communicates with it through files.

- **class** ProcessHandleApplicInterface
  Derived application interface class that spawns a simulation code using a separate process, receives a process identifier, and communicates with the spawned process through files.

- **class** ProgramOptions
  `ProgramOptions` stores options whether from the CLH or from library user; initially valid only on worldRank = 0, but then broadcast in `ParallelLibrary::manage_outputs_restart`.

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

- **struct** partial_prp_equality
  predicate for comparing ONLY the interfaceId and Vars attributes of PRPair

- **class** PSUADESDesignCompExp
  Wrapper class for the PSUADE library.

- **class** PythonInterface

- **class** RecastModel
  Derived model class which provides a thin wrapper around a sub-model in order to recast the form of its inputs and/or outputs.

- **class** RelaxedVarConstraints
  Derived class within the `Constraints` hierarchy which employs relaxation of discrete variables.

- **class** RelaxedVariables
  Derived class within the `Variables` hierarchy which employs the relaxation of discrete variables.

- **class** ResultsDBAny

- **class** ResultsID
  Get a globally unique 1-based execution number for a given iterator name (combination of methodName and methodID) for use in results DB. Each `Iterator::run()` call creates or increments this count for its string identifier.

- **class** ResultsNames
  List of valid names for iterator results.

- **class** ResultsManager
  Results manager for iterator final data.

- **class** ResultsEntry
  Class to manage in-core vs. file database lookups.

- **class** RichExtrapVerification
  Class for Richardson extrapolation for code and solution verification.
• class ScilabInterface
• class SensAnalysisGlobal
  Class for a utility class containing correlation calculations and variance-based decomposition.
• class SeqHybridMetaIterator
  Method for sequential hybrid iteration using multiple optimization and nonlinear least squares methods on multiple
  models of varying fidelity.
• class SharedApproxData
  Base class for the shared approximation data class hierarchy.
• class SharedPecosApproxData
  Derived approximation class for global basis polynomials.
• class SharedSurfpackApproxData
  Derived approximation class for Surfpack approximation classes. Interface between Surfpack and Dakota.
• class SharedVariablesDataRep
  The representation of a SharedVariablesData instance. This representation, or body, may be shared by multiple
  SharedVariablesData handle instances.
• class SharedVariablesData
  Container class encapsulating variables data that can be shared among a set of Variables instances.
• class SingleModel
  Derived model class which utilizes a single interface to map variables into responses.
• class SNLLBase
  Base class for OPT++ optimization and least squares methods.
• class SNLLLeastSq
  Wrapper class for the OPT++ optimization library.
• class SNLLOptimizer
  Wrapper class for the OPT++ optimization library.
• class SOLBase
  Base class for Stanford SOL software.
• class SpawnApplicInterface
  Derived application interface class which spawns simulation codes using spawnvp.
• class SurfpackApproximation
  Derived approximation class for Surfpack approximation classes. Interface between Surfpack and Dakota.
• class SurrBasedGlobalMinimizer
  The global surrogate-based minimizer which sequentially minimizes and updates a global surrogate model without
  trust region controls.
• class SurrBasedLocalMinimizer
  Class for provably-convergent local surrogate-based optimization and nonlinear least squares.
• class SurrBasedMinimizer
  Base class for local/global surrogate-based optimization/least squares.
• class SurrogateModel
  Base class for surrogate models (DataFitSurrModel and HierarchSurrModel).
• class SysCallApplicInterface
  Derived application interface class which spawns simulation codes using system calls.
• class TANA3Approximation
  Derived approximation class for TANA-3 two-point exponential approximation (a multipoint approximation).
12.1. DAKOTA NAMESPACE REFERENCE

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

- **class TestDriverInterface**

- **class TrackerHTTP**
  
  *TrackerHTTP: a usage tracking module that uses HTTP/HTTPS via the curl library.*

**Typedefs**

- typedef double **Real**
- typedef std::string **String**
- typedef Teuchos::SerialDenseVector< int, Real > **RealVector**
- typedef Teuchos::SerialDenseMatrix< int, Real > **RealMatrix**
- typedef Teuchos::SerialSymDenseMatrix< int, Real > **RealSymMatrix**
- typedef Teuchos::SerialDenseVector< int, int > **IntVector**
- typedef Teuchos::SerialDenseMatrix< int, int > **IntMatrix**
- typedef std::deque< bool > **BoolDeque**
- typedef boost::dynamic_bitset< unsigned long > **BitArray**
- typedef std::vector< BoolDeque > **BoolDequeArray**
- typedef std::vector< Real > **RealArray**
- typedef std::vector< RealArray > **Real2DArray**
- typedef std::vector< int > **IntArray**
- typedef std::vector< IntArray > **Int2DArray**
- typedef std::vector< short > **ShortArray**
- typedef std::vector< unsigned > **UShortArray**
- typedef std::vector< UShortArray > **UShort2DArray**
- typedef std::vector< UShort2DArray > **UShort3DArray**
- typedef std::vector< size_t > **SizetArray**
- typedef std::vector< SizetArray > **Sizet2DArray**
- typedef std::vector< String > **StringArray**
- typedef std::vector< StringArray > **String2DArray**
- typedef boost::multi_array_types::index_range **idx_range**
- typedef boost::multi_array< String > **StringMultiArray**
typedef StringMultiArray::array_view < 1 >::type StringMultiArrayView
typedef StringMultiArray::const_array_view < 1 >::type StringMultiArrayConstView
typedef boost::multi_array < unsigned short, 1 >::type UShortMultiArray
typedef UShortMultiArray::array_view < 1 >::type UShortMultiArrayView
typedef UShortMultiArray::const_array_view < 1 >::type UShortMultiArrayConstView
typedef boost::multi_array < size_t, 1 >::type SizetMultiArray
typedef SizetMultiArray::array_view < 1 >::type SizetMultiArrayView
typedef SizetMultiArray::const_array_view < 1 >::type SizetMultiArrayConstView
typedef std::vector < RealVectorArray >::type RealVectorArray
typedef std::vector < RealVector2DArray >::type RealVector2DArray
typedef std::vector < RealMatrixArray >::type RealMatrixArray
typedef std::vector < RealSymMatrixArray >::type RealSymMatrixArray
typedef std::vector < IntVectorArray >::type IntVectorArray
typedef std::vector < VariablesArray >::type VariablesArray
typedef std::vector < ResponseArray >::type ResponseArray
typedef std::vector < PRPArray >::type PRPArray
typedef std::vector < PRP2DArray >::type PRP2DArray
typedef std::vector < ModelArray >::type ModelArray
typedef std::vector < IteratorArray >::type IteratorArray
typedef std::list < BoolList >::type BoolList
typedef std::list < IntList >::type IntList
typedef std::list < SizetList >::type SizetList
typedef std::list < RealList >::type RealList
typedef std::list < RealVectorList >::type RealVectorList
typedef std::list < StringList >::type StringList
typedef std::list < VariablesList >::type VariablesList
typedef std::list < InterfaceList >::type InterfaceList
typedef std::list < ResponseList >::type ResponseList
typedef std::list < ModelList >::type ModelList
typedef std::list < IteratorList >::type IteratorList
12.1. DAKOTA NAMESPACE REFERENCE

- typedef std::pair<int, int> IntIntPair
- typedef std::pair<int, String> IntStringPair
- typedef std::pair<Real, Real> RealRealPair
- typedef std::pair<int, Response> IntResponsePair
- typedef std::set<Real> RealSet
- typedef std::set<int> IntSet
- typedef std::set<unsigned short> UShortSet
- typedef std::set<size_t> SizetSet
- typedef std::vector<RealSet> RealSetArray
- typedef std::vector<IntSet> IntSetArray
- typedef std::vector<UShortSet> UShortSetArray
- typedef std::map<int, int> IntIntMap
- typedef std::map<int, short> IntShortMap
- typedef std::map<int, Real> IntRealMap
- typedef std::map<Real, Real> RealRealMap
- 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> IntArraySizetMap
- typedef std::multimap<RealRealPair, ParamResponsePair> RealPairPRPMultiMap
- typedef IntList::iterator ILIter
- typedef IntList::const_iterator ILCIter
- typedef SizetList::iterator StLIter
- typedef SizetList::const_iterator StLCIter
- typedef RealList::iterator RLIter
- typedef RealList::const_iterator RLCIter
- typedef RealVectorList::iterator RVLIter
- typedef RealVectorList::const_iterator RVLCIter
- typedef StringList::iterator StringLIter
- typedef StringList::const_iterator StringLCIter
- typedef VariablesList::iterator VarsLIter
- typedef InterfaceList::iterator InterLIter
- typedef ResponseList::iterator RespLIter
- typedef ModelList::iterator ModelLIter
- typedef ModelList::reverse_iterator ModelLRevIter
- typedef IteratorList::iterator IterLIter
- typedef std::list<ParallelLevel>::iterator ParLevLIter
• typedef std::list
  < ParallelConfiguration >
  ::iterator ParConfigLIter
• typedef IntSet::iterator ISIter
• typedef IntSet::const_iterator ISClIter
• typedef RealSet::iterator RSIIter
• typedef RealSet::const_iterator RSCIter
• typedef IntIntMap::iterator IntIntMIter
• typedef IntIntMap::const_iterator IntIntMCIter
• typedef IntShortMap::iterator IntShMIter
• typedef IntShortMap::const_iterator IntShMCIter
• typedef IntRealMap::iterator IRMIter
• typedef IntRealMap::const_iterator IRMCIter
• typedef RealRealMap::iterator RRMIter
• typedef RealRealMap::const_iterator RRMCIter
• typedef 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(*)(const char *, const struct stat *, int ftype, int depth, void *) ftw
• typedef struct dirent dirent
• typedef struct Dakota::Cbuf Cbuf
• typedef struct Dakota::Buf Buf
• typedef struct Dakota::Finfo Finfo
• typedef boost::tuple
  < std::string, std::string, size_t, std::string > ResultsKeyType
          Data type for results key (instance name / id, unique run, label), where data_key is a valid colon-delimited string from ResultsNames tuple<method_name, method_id, execution_number, data_key>
• typedef std::string MetaDataKeyType
          Data type for metadata key.
• typedef std::vector< std::string > MetaDataValueType
          Data type for metadata value.
• typedef std::map< MetaDataKeyType, MetaDataValueType > MetaDataType
          A single MetaData entry is map<string, vector<string>> Example: pair( "Column labels", ["Mean", "Std Dev", "Skewness", "Kurtosis"] )
• typedef boost::tuple
  < std::string, std::string, size_t > StrStrSizet
12.1. DAKOTA NAMESPACE REFERENCE

**Iterator unique ID:** `<method_name, method_id, exec_num>`

- `typedef void(* dl_find_optimum_t)(void *, Optimizer1 *, char *)`
- `typedef void(* dl_destructor_t)(void **)`
- `typedef Teuchos::SerialDenseSolver < int, Real > RealSolver`
- `typedef Teuchos::SerialSpdDenseSolver < int, Real > RealSpdSolver`
- `typedef int(* start_grid_computing_t )(char *analysis_driver_script, char *params_file, char *results_file)`
  
  *definition of start grid computing type (function pointer)*
- `typedef int(* perform_analysis_t )(char *iteration_num)`
  
  *definition of perform analysis type (function pointer)*
- `typedef int (*)(* get_jobs_completed_t )()`
  
  *definition of get completed jobs type (function pointer)*
- `typedef int (*)(* stop_grid_computing_t )()`
  
  *definition of stop grid computing type (function pointer)*
- `typedef int MPI_Comm`
- `typedef void * MPI_Request`
- `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 (*)(* Vf )()`
- `typedef void (*)(* DbCallbackFunctionPtr )(Dakota::ProblemDescDB *db, void *data_ptr)`
- `typedef bmi::multi_index_container < Dakota::ParamResponsePair,`
  
  bmi::indexed_by`
  
  < bmi::tag < ordered >`
  
  , bmi::const_mem_fun`
  
  < Dakota::ParamResponsePair, const IntStringPair &,&Dakota::ParamResponsePair::eval_interface_ids >`
  
  >, bmi::hashed_non_unique`
  
  < bmi::tag < hashed >`
  
  , bmi::identity`
  
  < Dakota::ParamResponsePair >`
  
  , partial_prp_hash, partial_prp Equality > > > PRPMultiIndexCache`

  *Boost Multi-Index Container for globally caching ParamResponsePairs.*

- `typedef PRPMultiIndexCache PRPCache`
• typedef PRPCache::index_iterator
  < ordered >::type PRPCacheOIter

• typedef
  PRPCache::index_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

• typedef
  bmi::multi_index_container
  < Dakota::ParamResponsePair,
    bmi::indexed_by
    < bmi::ordered_unique
    < bmi::tag< ordered >
      , bmi::const_mem_fun
      < Dakota::ParamResponsePair,
        int,&Dakota::ParamResponsePair::eval_id
      >, bmi::hashed_non_unique
    < bmi::tag< hashed >
      , bmi::identity
      < Dakota::ParamResponsePair
        , partial_prp_hash,
        partial_prp_equality > > > PRPMultiIndexQueue

  Boost Multi-Index Container for locally queueing ParamResponsePairs.

• typedef PRPMultiIndexQueue PRPQueue

• typedef
  PRPQueue::index_iterator
  < ordered >::type PRPQueueOIter

• typedef
  PRPQueue::index_const_iterator
  < ordered >::type PRPQueueOCIter

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

• typedef
  PRPQueue::index_const_iterator
  < hashed >::type PRPQueueHCIter

• typedef PRPQueueOIter PRPQueueIter
• typedef PRPQueueOCIter PRPQueueCIter

• typedef std::pair< boost::any,
  MetaDataType > ResultsValueType

  Core data storage type: boost::any, with optional metadata (see other types in results_types.hpp)
Enumerations

- `enum` {
  COBYLA, DIRECT, EA, MS,
  PS, SW, BETA }

- `enum` {
  sFTW_F, sFTW_SL, sFTW_D, sFTW_DP,
  sFTW_DNR, sFTW_O, sFTW_NS }

- `enum` {
  sFTWret_OK, sFTWret_quit, sFTWret_skipdir, sFTWret_Follow,
  sFTWret_mallocfailure }  

- `enum` { OBJECTIVE, INEQUALITY_CONSTRAINT, EQUALITY_CONSTRAINT }

  define algebraic function types

- `enum` {
  DEFAULT_METHOD =0, HYBRID =META_BIT, PARETO_SET, MULTI_START,
  BRANCH_AND_BOUND, RICHARDSON_EXTRAP = (ANALYZER_BIT | VERIF_BIT), CENTERED_PARAMETER_STUDY = (ANALYZER_BIT | PSTUDYDACE_BIT), LIST_PARAMETER_STUDY,
  MULTIDIM_PARAMETER_STUDY, VECTOR_PARAMETER_STUDY, DACE, FSU_CVT,
  FSU_HALTON, FSU_HAMMERSLEY, PSUADE_MOAT, LOCAL_RELIABILITY = (ANALYZER_BIT | NOND_BIT),
  GLOBAL_RELIABILITY, POLYNOMIAL_CHAOS, STOCH_COLLOCATION, CUBATURE_INTEGRATION,
  SPARSE_GRID_INTEGRATION, QUADRATURE_INTEGRATION, BAYES_CALIBRATION, GPAIS,
  POF_DARTS, EFFICIENT_SUBSPACE, IMPORTANCE_SAMPLING, ADAPTIVE_SAMPLING,
  RANDOM_SAMPLING, LOCAL_INTERVAL_EST, LOCAL_EVIDENCE, GLOBAL_INTERVAL_EST,
  GLOBAL_EVIDENCE, SURROGATE_BASED_LOCAL = (MINIMIZER_BIT | SURRBASED_BIT),
  SURROGATE_BASED_GLOBAL, EFFICIENT_GLOBAL,
  NL2SOL = (MINIMIZER_BIT | LEASTSQ_BIT), NLSSOL_SQP, OPTPP_G_NEWTON, ASYNCH_PATTERN_SEARCH = (MINIMIZER_BIT | OPTIMIZER_BIT),
  OPTPP_PDS, COLINY_BETA, COLINY_COBYLA, COLINY_DIRECT,
  COLINY_MULTI_START, COLINY_EA, COLINY_PATTERN_SEARCH, COLINY_SOLIS_WETS,
  MOGA, SOGA, NCSU_DIRECT, MESH_ADAPTIVE_SEARCH,
  GENIE_OPT_DARTS, GENIE_DIRECT, NONLINEAR_CG, OPTPP_CG,
  OPTPP_Q_NEWTON, OPTPP_FD_NEWTON, OPTPP_NEWTON, NPSOL_SQP,
  NLQL_SQP, DOT_BFGS, DOT_FRCG, DOT_MMFD,
  DOT_SLP, DOT_SQP, CONMIN_FRCG, CONMIN_MFD,
  DL_SOLVER }

- `enum` {
  SUBMETHOD_DEFAULT =0, SUBMETHOD_COLLABORATIVE, SUBMETHOD_EMBEDDED, S-
  SUBMETHOD_SEQUENTIAL,
  SUBMETHOD_LHS, SUBMETHOD_RANDOM, SUBMETHOD_INCREMENTAL_LHS, SUBME-
  THOD_INCREMENTAL_RANDOM,
  SUBMETHOD_BOX_BEHNKEN, SUBMETHOD_CENTRAL_COMPOSITE, SUBMETHOD_GRI-
D, SUBMETHOD_OA_LHS,
SUBMETHOD_OAS, SUBMETHOD_DREAM, SUBMETHOD_GPMSA, SUBMETHOD_QUESO,
SUBMETHOD_NIP, SUBMETHOD_SQP, SUBMETHOD_EA, SUBMETHOD_EGO,
SUBMETHOD_SBO, SUBMETHOD_CONVERGE_ORDER, SUBMETHOD_CONVERGE_QOI, S-
SUBMETHOD_ESTIMATE_ORDER }

Sub-methods, including sampling, inference algorithm, opt algorithm types.

• enum { 
SILENT_OUTPUT, QUIET_OUTPUT, NORMAL_OUTPUT, VERBOSE_OUTPUT,
DEBUG_OUTPUT }
• enum { 
DEFAULT_SCHEDULING, MASTER_SCHEDULING, PEER_SCHEDULING, PEER_DYNAMIC-
_SCHEDULING,
PEER_STATIC_SCHEDULING, DYNAMIC_SCHEDULING, STATIC_SCHEDULING }
• enum { DEFAULT_CONFIG, PUSH_DOWN, PUSH_UP }
• enum { STD_NORMAL_U, STD_UNIFORM_U, ASKEY_U, EXTENDED_U }
• enum { DEFAULT_COVARIANCE, NO_COVARIANCE, DIAGONAL_COVARIANCE, FULL_CO-
VARIANCE }
• enum { NO_INT_REFINE =0, IS, AIS, MMAIS }
• enum { PROBABILITIES, RELIABILITIES, GEN_RELIABILITIES }
• enum { COMPONENT =0, SYSTEM_SERIES, SYSTEM_PARALLEL }
• enum { CUMULATIVE, COMPLEMENTARY }
• enum { DEFAULT_LS =0, SVD_LS, EQ_CON_LS }
• enum { NO_EMULATOR, PCE_EMULATOR, SC_EMULATOR, GP_EMULATOR,
KRIGING_EMULATOR }
• enum { IGNORE_RANKS, SET_RANKS, GET_RANKS, SET_GET_RANKS }
• enum { 
UNCERTAIN, UNCERTAIN_UNIFORM, ALEATORY_UNCERTAIN, ALEATORY_UNCERTAI-
N_UNIFORM,
EPISTEMIC_UNCERTAIN, EPISTEMIC_UNCERTAIN_UNIFORM, ACTIVE, ACTIVE_UNIFO-
RM,
ALL, ALL_UNIFORM }
• enum { 
MV =0, AMV_X, AMV_U, AMV_PLUS_X,
AMV_PLUS_U, TANA_X, TANA_U, NO_APPROX,
EGRA_X, EGRA_U }
• enum { BREITUNG, HOHENRACK, HONG }
• enum { ORIGINAL_PRIMARY, SINGLE_OBJECTIVE, LAGRANGIAN_OBJECTIVE, AUGME-
NTED_LAGRANGIAN_OBJECTIVE }
• enum { NOCONSTRAINTS, LINEARIZED_CONSTRAINTS, ORIGINAL_CONSTRAINTS }
• enum { NO_RELAX, HOMOTOPY, COMPOSITE_STEP }
• enum { PENALTY MERIT, ADAPTIVE_PENALTY MERIT, LAGRANGIAN MERIT,
AUGME-
NTED_LAGRANGIAN_MERIT }
• enum { FILTER, TR_RATIO }
• enum { SCALE_NONE, SCALE_VALUE, SCALE_LOG }
• enum { CDV, LINEAR, NONLIN, FN_LSQ }
• enum { DISALLOW, TARGET, BOUNDS }
• enum { DEFAULT_POINTS, MINIMUM_POINTS, RECOMMENDED_POINTS, TOTAL_POINTS }

  define special values for pointsManagement

• enum {
  NO_SURROGATE =0, UNCORRECTED_SURROGATE, AUTO_CORRECTED_SURROGATE, BYPASS_SURROGATE,
  MODEL_DISCREPANCY }

  define special values for SurrogateModel::responseMode

• enum { NO_CORRECTION =0, ADDITIVE_CORRECTION, MULTIPLICATIVE_CORRECTION, COMBINED_CORRECTION }

  define special values for approxCorrectionType

• enum { DEFAULT_DOMAIN =0, RELAXED_DOMAIN, MIXED_DOMAIN }

• enum {
  DEFAULT_VIEW =0, ALL_VIEW, DESIGN_VIEW, UNCERTAIN_VIEW,
  ALEATORY_UNCERTAIN_VIEW, EPISTEMIC_UNCERTAIN_VIEW, STATE_VIEW }

• enum {
  EMPTY =0, RELAXED_ALL, MIXED_ALL, RELAXED_DESIGN,
  RELAXED_UNCERTAIN, RELAXED_ALEATORY_UNCERTAIN, RELAXED_EPISTEMIC_UNCERTAIN,
  RELAXED_STATE, MIXED_DESIGN, MIXED_UNCERTAIN, MIXED_ALEATORY_UNCERTAIN, MIXED_EPSTEMIC_UNCERTAIN,
  MIXED_STATE }

• enum {
  CONTINUOUS_DESIGN =1, DISCRETE_DESIGN_RANGE, DISCRETE_DESIGN_SET_INT, DISCRETE_DESIGN_SET_REAL,
  NORMAL_UNCERTAIN, LOGNORMAL_UNCERTAIN, UNIFORM_UNCERTAIN, LOGUNIFORM_UNCERTAIN,
  TRIANGULAR_UNCERTAIN, EXPONENTIAL_UNCERTAIN, BETA_UNCERTAIN, GAMMA_UNCERTAIN,
  GUMBEL_UNCERTAIN, FRECHET_UNCERTAIN, WEIBULL_UNCERTAIN, HISTOGRAM_BIN_UNCERTAIN,
  POISSON_UNCERTAIN, BINOMIAL_UNCERTAIN, NEGATIVE_BINOMIAL_UNCERTAIN, GEOMETRIC_UNCERTAIN,
  HYPERGEOMETRIC_UNCERTAIN, HISTOGRAM_POINT_UNCERTAIN, CONTINUOUS_INTERVAL_UNCERTAIN,
  DISCRETE_INTERVAL_UNCERTAIN, DISCRETE_UNCERTAIN_SET_INT, DISCRETE_UNCERTAIN_SET_REAL, CONTINUOUS_STATE,
  DISCRETE_STATE_RANGE, DISCRETE_STATE_SET_INT, DISCRETE_STATE_SET_REAL }

• enum var_t {
  VAR_x1, VAR_x2, VAR_x3, VAR_b,
  VAR_h, VAR_P, VAR_M, VAR_Y,
  VAR_w, VAR_t, VAR_R, VAR_E,
  VAR_X, VAR_Fs, VAR_P1, VAR_P2,
  VAR_P3, VAR_B, VAR_D, VAR_H,
  VAR_F0, VAR_d, VAR_MForm }

  enumeration of possible variable types (to index to names)
CHAPTER 12. NAMESPACE DOCUMENTATION

- enum driver_t {
  NO_DRIVER = 0, CANTILEVER_BEAM, MOD_CANTILEVER_BEAM, CYLINDER_HEAD, 
  EXTENDED_ROSENROCK, GENERALIZED_ROSENROCK, LF_ROSENROCK, MF_ROSENROCK, 
  ROSENROCK, GERSTNER, SCALABLE GERSTNER, LOGNORMAL_RATIO, 
  MULTIMODAL, PLUGIN_ROSENROCK, PLUGIN_TEXT_BOOK, SHORT_COLUMN, 
  LF_SHORT_COLUMN, MF_SHORT_COLUMN, SIDE_IMPACT_COST, SIDE_IMPACT_PERFORMANCE, 
  SOBOL_RATIONAL, SOBOL_G_FUNCTION, SOBOL_ISHIGAMI, STEEL_COLUMN_COST, 
  STEEL_COLUMN_PERFORMANCE, TEXT_BOOK, TEXT_BOOK1, TEXT_BOOK2, 
  TEXT_BOOK3, TEXT_BOOK_OUU, SCALABLE_TEXT_BOOK, SCALABLE_MONOMIALS, 
  HERBIE, SMOOTH_Herbie, SHUBERT, SALINAS, 
  MODELCENTER, GENZ }

  enumeration of possible direct driver types (to index to names)

- enum local_data_t { VARIABLES_MAP = 1, VARIABLES_VECTOR = 2 }

  enumeration for how local variables are stored (values must employ a bit representation)

- enum sigtype { NO_SIGMA, SCALAR_SIGMA, COVARIANCE MATRIX }

  special values for sigma type

- enum edtype { SCALAR_DATA, FUNCTIONAL_DATA }

  special values for experimental data type

- 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_frecchet = 9, CAUVar_weibull = 10, CAUVar_histogram_bin = 11,
  CAUVar_Nkinds = 12 }

- enum {
  DAUVar_poisson = 0, DAUVar_binomial = 1, DAUVar_negative_binomial = 2, DAUVar_geometric = 3,
  DAUVar_hypergeometric = 4, DAUVar_Nkinds = 5 }

- enum { DAUVar_histogram_point = 0, DAUVar_Nkinds = 1 }

- enum { CEUVar_interval = 0, CEUVar_Nkinds = 1 }

- enum { DEUVar_interval = 0, DEUVar_set_int = 1, DEUVar_Nkinds = 2 }

- enum { DEURVar_set_real = 0, DEURVar_Nkinds = 1 }

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

- enum { N_VLR = 4 }

- enum { N_VLI = 2 }

- enum { FULL_TENSOR, FILTERED_TENSOR, RANDOM_TENSOR }

- enum CG_UPDATETYPE {
  CG_STEEPEST, CG_FLETCHER_REEVE, CG_POLAK_RIBIERE, CG_POLAK_RIBIERE_PLU-
  S, 
  CGHESTENES_STIEFEL }

  NonlinearCG update options.
• enum \texttt{CG	extsc{lineSearchType}} \{ \texttt{CG\_FIXED\_STEP}, \texttt{CG\_LS\_SIMPLE}, \texttt{CG\_LS\_BRENT}, \texttt{CG\_LS\_WOLFE} \}

  Nonlinear CG linesearch options.

• enum \texttt{EvalType} \{ \texttt{NLFEvaluator}, \texttt{CONEvaluator} \}

  enumeration for the type of evaluator function

• enum \{ 

  \texttt{TH\_SILENT\_OUTPUT}, \texttt{TH\_QUIET\_OUTPUT}, \texttt{TH\_NORMAL\_OUTPUT}, \texttt{TH\_VERBOS\_OUTPUT}, 

  \texttt{TH\_DEBUG\_OUTPUT} \}

\section*{Functions}

• \texttt{CommandShell} \& \texttt{flush} (\texttt{CommandShell} \&\texttt{shell})
  
  convenient shell manipulator function to “flush” the shell

• \texttt{bool} \texttt{nearby} (const \texttt{RealVector} \&\texttt{rv1}, const \texttt{RealVector} \&\texttt{rv2}, \texttt{Real rel\_tol})

  tolerance-based equality operator for \texttt{RealVector}

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

  equality operator for \texttt{ShortArray}

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

  equality operator for \texttt{StringArray}

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

  equality operator for \texttt{SizetArray} and \texttt{SizetMultiArrayConstView}

• \texttt{Real rel\_change\_L2} (const \texttt{RealVector} \&\texttt{curr\_rv}, const \texttt{RealVector} \&\texttt{prev\_rv})

  Computes relative change between \texttt{RealVectors} using Euclidean L2 norm.

• \texttt{Real rel\_change\_L2} (const \texttt{RealVector} \&\texttt{curr\_rv1}, const \texttt{RealVector} \&\texttt{prev\_rv1}, const \texttt{IntVector} \&\texttt{curr\_iv}, const \texttt{IntVector} \&\texttt{prev\_iv}, const \texttt{RealVector} \&\texttt{curr\_rv2}, const \texttt{RealVector} \&\texttt{prev\_rv2})

  Computes relative change between Real/int/Real vector triples using Euclidean L2 norm.

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

  equality operator for \texttt{IntArray}

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

  inequality operator for \texttt{IntArray}

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

  inequality operator for \texttt{ShortArray}

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

  inequality operator for \texttt{StringArray}

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

  inequality operator for \texttt{StringArray}

• \texttt{std::string} \texttt{strtolower} (const \texttt{std::string} \&\texttt{s})

  Return lowercase copy of string s.

• \texttt{bool} \texttt{strbegins} (const \texttt{std::string} \&\texttt{input}, const \texttt{std::string} \&\texttt{test})

  Return true if input string begins with string test.

• \texttt{bool} \texttt{streends} (const \texttt{std::string} \&\texttt{input}, const \texttt{std::string} \&\texttt{test})

  Return true if input string ends with string test.

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

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

- **void build_labels** (StringArray &label_array, const String &root_label)
  create an array of labels by tagging root_label for each entry in label_array. Uses build_label().

- **void build_labels** (StringMultiArray &label_array, const String &root_label)
  create an array of labels by tagging root_label for each entry in label_array. Uses build_label().

- **void build_labels_partial** (StringArray &label_array, const String &root_label, size_t start_index, size_t num_items)
  create a partial array of labels by tagging root_label for a subset of entries in label_array. Uses build_label().

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

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

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

- **template<typename OrdinalType, typename ScalarType> void copy_data** (const Teuchos::SerialDenseVector<OrdinalType, ScalarType> &sdv, Teuchos::SerialDenseVector<OrdinalType, ScalarType> &sdv2)
  copy Teuchos::SerialDenseVector<OT,ST> to Teuchos::SerialDenseVector<OT,ST>

- **template<typename T> void copy_data** (const std::list<T> &dl, std::vector<T> &da)
  copy std::list<T> to std::vector<T>

- **template<typename T> void copy_data** (const std::list<T> &dl, std::vector<std::vector<T>> &d2a, size_t num_a, size_t a_len)
  copy std::list<T> to std::vector<std::vector<T> > (unroll vecOfvecs into vector)

- **template<typename T> void copy_data** (const std::map<int, T> &im, std::vector<T> &da)
  copy map<int, T> to std::vector<T> (discard integer keys)

- **template<typename OrdinalType, typename ScalarType> void copy_data** (const Teuchos::SerialDenseVector<OrdinalType, ScalarType> &sdv1, Teuchos::SerialDenseVector<OrdinalType, ScalarType> &sdv2)
  copy Teuchos::SerialDenseVector<OrdinalType, ScalarType> to same (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, std::vector<T> &da)
template<typename OrdinalType, typename ScalarType>
void copy_data (const std::vector<ScalarType> &da, Teuchos::SerialDenseVector<OrdinalType, ScalarType> &sdv)

void copy(Array<ScalarType>& da, Teuchos::SerialDenseVector<OrdinalType, ScalarType> &sdv)

void copy(data(const Teuchos::SerialDenseVector<OrdinalType, ScalarType>& sdv, std::vector<Teuchos::SerialDenseVector<OrdinalType, ScalarType>> &sdva, OrdinalType num_vec, OrdinalType vec_len)

copy portion of first SerialDenseVector to all of second SerialDenseVector

copy all of first SerialDenseVector to portion of second SerialDenseVector

copy all of first SerialDenseVector to portion of second SerialDenseVector

copy all of first SerialDenseVector to portion of second SerialDenseVector

copy all of first Array<T> to all of second Array<T>

copy all of first Array<T> to portion of second Array<T>

copy all of first Array<T> to portion of boost::multi_array<T, 1>& bma, size_t start_index2)
template<typename T>
void copy_data_partial (const std::vector<T> &da1, size_t start_index1, size_t num_items, std::vector<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_vec, RealVector &m_vec, size_t start_index, size_t num_items)
merge a discrete integer vector into a single continuous vector

void merge_data_partial (const IntVector &d_vec, RealArray &m_array, size_t start_index, size_t num_items)
merge a discrete integer vector into a single continuous 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

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 KeyType, typename ValueType>
const KeyType & map_index_to_key (OrdinalType index, const std::map<KeyType, ValueType> &pairs)
retrieve the set value corresponding to the passed index

template<typename OrdinalType, typename KeyType, typename ValueType>
const ValueType & map_index_to_value (OrdinalType index, const std::map<KeyType, ValueType> &pairs)
retrieve the set value corresponding to the passed index

template<typename KeyType, typename ValueType>
void map_keys_to_set (const std::map<KeyType, ValueType> &source_map, std::set<KeyType> &target_set)
calculate the map index corresponding to the passed key

size_t map_key_to_index (const KeyType &key, const std::map<KeyType, ValueType> &pairs)
calculate the map index corresponding to the passed key

template<typename OrdinalType, typename MultiArrayType, typename DakArrayType>
void copy_data (const MultiArrayType &ma, DakArrayType &da)
generic copy (inactive)

template<typename T>
size_t find_index (const boost::multi_array<T, 1> &bma, const T &search_data)
compute the index of an entry within a boost::multi_array

size_t find_index (SizetMultiArrayConstView bmacv, size_t search_data)
compute the index of an entry within a boost::multi_array view

size_t find_index (StringMultiArrayConstView bmacv, const String &search_data)
compute the index of an entry within a boost::multi_array view
12.1. DAKOTA NAMESPACE REFERENCE

- template<typename ListT >
  size_t find_index (const ListT &l, const typename ListT::value_type &val)
  compute the index of an entry within a std::list
- void copy_data (SizetMultiArrayConstView ma, SizetArray &da)
  copy boost::multi_array view to Array
- void copy_data (StringMultiArrayConstView ma, StringArray &da)
  copy boost::multi_array view to Array
- template<typename ListT >
  ListT::const_iterator find_if (const ListT &c, bool(*test_fn)(const typename ListT::value_type &, const std::string &), const std::string &test_fn_data)
  return an iterator to the first list element satisfying the predicate test_fn w.r.t. the passed test_fn_data; end if not found
- template<typename DakContainerType >
  bool contains (const DakContainerType &v, const typename DakContainerType::value_type &val)
  return true if the item val appears in container v
- void dak_sicatch (int sig)
- void start_dakota_heartbeat (int seconds)
- int sftw (const char *name, int(*fn)(const char *file, const struct stat *, int ftype, int depth, void *v), void *v)
- static int ftw1 (char *name, size_t namelen, size_t namemaxlen, ftw_fn fn, int, void *v)
- static int compar (const void *a, const void *b)
- static int dodir (DIR *dir, char *name, size_t namelen, size_t namemaxlen, ftw_fn fn, int depth, void *v, struct stat *sb)
- int sftw (const char *name, ftw_fn fn, void *v)
- static int Symlink (const char *from, const char *to)
- static int my_recrm (const char *file, const struct stat *sb, int ftype, int depth, void *v)
- int rec_rmdir (const char *name)
- static void buf_incr (Buf *b, size_t Lt)
- int my_cp (const char *file, const struct stat *sb, int ftype, int depth, void *v)
- int rec_cp (const char *from, const char *todir, int copy, int flatten, int replace)
- static char * pathsimp (char *t0)
- void get_npath (int appdrive, std::string *pnpath)
- void workdir_adjust (const std::string &workdir)
- std::string get_cwd ()
  Portability adapter for getcwd.
- std::vector< std::string > get_pathext ()
  Utility function for executable file search algorithms.
- void putenv_impl (const char *name_and_value)
  Utility function from borrowed from boost/test.
- void abort_handler (int code)
  global function which handles serial or parallel aborts
- void register_signal_handlers ()
  Tie various signal handlers to Dakota’s abort_handler function.
- void mpi_debug_hold ()
  Global function to hold Dakota processes to help with MPI debugging.
CHAPTER 12. NAMESPACE DOCUMENTATION

- template<typename T>
  
  T abort_handler_1 (int code)

- ResultsKeyType make_key (const StrStrSize &iterator_id, const std::string &data_name)
  
  Make a full ResultsKeyType from the passed iterator_id and data_name.

- MetaDataValueType make_metadatavalue (StringMultiArrayConstView labels)
  
  create MetaDataValueType from the passed strings

- MetaDataValueType make_metadatavalue (StringMultiArrayConstView cv_labels, StringMultiArrayConstView div_labels, StringMultiArrayConstView drv_labels, const StringArray &resp_labels)
  
  create MetaDataValueType from the passed strings

- MetaDataValueType make_metadatavalue (const StringArray &resp_labels)
  
  create MetaDataValueType from the passed strings

- MetaDataValueType make_metadatavalue (const std::string &
  
  create MetaDataValueType from the passed strings

- MetaDataValueType make_metadatavalue (const std::string &, const std::string &
  
  create MetaDataValueType from the passed strings

- MetaDataValueType make_metadatavalue (const std::string &, const std::string &, const std::string &
  
  create MetaDataValueType from the passed strings

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

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

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

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

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

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

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

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

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

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

- bool operator== (const Model &m1, const Model &m2)
  
  equality operator for Envelope is true if same letter instance

- bool operator!= (const Model &m1, const Model &m2)
inequality operator for Envelope is true if different letter instance

- `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&).`
- `MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, Response &response)`
  - `MPIUnpackBuffer extraction operator for Response. Calls read(MPIUnpackBuffer&).`
- `MPIPackBuffer & operator<<(MPIPackBuffer &s, const Response &response)`
  - `MPIPackBuffer insertion operator for Response. Calls write(MPIPackBuffer&).`
- `bool operator==(const Response &resp1, const Response &resp2)`
  - equality operator for Response
- `bool operator!=(const Response &resp1, const Response &resp2)`
  - inequality operator for Response
- `std::string re_match (const std::string &token, const boost::regex &re)`
  - Global utility function to ease migration from CtelRegExp to Boost.Regex.
- `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.`
- `MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, Variables &vars)`
  - `MPIUnpackBuffer extraction operator for Variables.`
- `MPIPackBuffer & operator<<(MPIPackBuffer &s, const Variables &vars)`
  - `MPIPackBuffer insertion operator for Variables.`
- `bool operator!=(const Variables &vars1, const Variables &vars2)`
  - inequality operator for Variables
- `template<typename OrdinalType, typename ScalarType1, typename ScalarType2, typename ScalarType3>
  void write_ordered (std::ostream &s, const SizetArray &comp_totals, const Teuchos::SerialDenseVector<
      OrdinalType, ScalarType1> &c_vector, const Teuchos::SerialDenseVector< OrdinalType, ScalarType2 >
      &di_vector, const Teuchos::SerialDenseVector< OrdinalType, ScalarType3 > &dr_vector)`
  - ScalarType1 will be Real, ScalarType2 will be int, and ScalarType3 may be int or Real, but written for arbitrary types.
- `template<typename ScalarType>
  void write_ordered (std::ostream &s, const SizetArray &comp_totals, const std::vector<
      ScalarType > &c_vector, const std::vector< ScalarType > &di_vector, const std::vector< ScalarType > &dr_vector)`
  - ScalarType1 will be Real, ScalarType2 will be int, and ScalarType3 may be int or Real, but written for arbitrary types.
- `MPIPackBuffer & operator<<(MPIPackBuffer &s, const DataEnvironment &data)`
  - `MPIPackBuffer insertion operator for DataEnvironment.`
- `MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, DataEnvironment &data)`
  - `MPIUnpackBuffer extraction operator for DataEnvironment.`
• std::ostream & operator<<(std::ostream &s, const DataEnvironment &data)
  std::ostream insertion operator for DataEnvironment
• MPIPackBuffer & operator<<(MPIPackBuffer &s, const DataInterface &data)
  MPIPackBuffer insertion operator for DataInterface.
• MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, DataInterface &data)
  MPIUnpackBuffer extraction operator for DataInterface.
• std::ostream & operator<<(std::ostream &s, const DataInterface &data)
  std::ostream insertion operator for DataInterface
• MPIPackBuffer & operator<<(MPIPackBuffer &s, const DataMethod &data)
  MPIPackBuffer insertion operator for DataMethod.
• MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, DataMethod &data)
  MPIUnpackBuffer extraction operator for DataMethod.
• std::ostream & operator<<(std::ostream &s, const DataMethod &data)
  std::ostream insertion operator for DataMethod
• MPIPackBuffer & operator<<(MPIPackBuffer &s, const DataModel &data)
  MPIPackBuffer insertion operator for DataModel.
• MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, DataModel &data)
  MPIUnpackBuffer extraction operator for DataModel.
• std::ostream & operator<<(std::ostream &s, const DataModel &data)
  std::ostream insertion operator for DataModel
• MPIPackBuffer & operator<<(MPIPackBuffer &s, const DataResponses &data)
  MPIPackBuffer insertion operator for DataResponses.
• MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, DataResponses &data)
  MPIUnpackBuffer extraction operator for DataResponses.
• std::ostream & operator<<(std::ostream &s, const DataResponses &data)
  std::ostream insertion operator for DataResponses
• MPIPackBuffer & operator<<(MPIPackBuffer &s, const DataVariables &data)
  MPIPackBuffer insertion operator for DataVariables.
• MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, DataVariables &data)
  MPIUnpackBuffer extraction operator for DataVariables.
• std::ostream & operator<<(std::ostream &s, const DataVariables &data)
  std::ostream insertion operator for DataVariables
• int dlsolver_option (Opt_Info *)
• RealVector const * continuous_lower_bounds (Optimizer1 *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_ineq_constraint_coeffs (Optimizer1 *o)
• RealMatrix const * linear_eq_constraint_coeffs (Optimizer1 *o)
• void ComputeResponses (Optimizer1 *o, int mode, int n, double *x)
12.1. DAKOTA NAMESPACE REFERENCE

- void GetFuncs (Optimizer1 *o, int m0, int m1, double *f)
- void GetGrads (Optimizer1 *o, int m0, int m1, int n, int is, int js, double *g)
- void GetContVars (Optimizer1 *o, int n, double *x)
- void SetBestContVars (Optimizer1 *o, int n, double *x)
- void SetBestRespFns (Optimizer1 *o, int n, double *x)
- void *dl_constructor (Optimizer1 *, Dakotafuncs *, 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_eq_constraint_targets1 (Optimizer1 *o)
- static RealMatrix const *linear_ineq_constraint_coeffs1 (Optimizer1 *o)
- static RealMatrix const *linear_eq_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 GetReal1 (Optimizer1 *o, const char *name)
- static int GetInt1 (Optimizer1 *o, const char *name)
- static bool GetBool1 (Optimizer1 *o, const char *name)
- DOTOptimizer *new_DOTOptimizer (ProblemDescDB &problem_db)
- DOTOptimizer *new_DOTOptimizer (Model &model)
- DOTOptimizer *new_DOTOptimizer (ProblemDescDB &problem_db, Model &model)
- void read_historical_data (const std::string &expDataFileName, const std::string &context_message, size_t numExperiments, IntVector &numReplicates, size_t numExpConfigVars, size_t numFunctions, size_t numExpStdDeviationsRead, bool expDataFileAnnotated, bool calc_sigma_from_data, RealMatrix &xObsData, RealMatrixArray &yObsData, RealMatrixArray &yStdData)

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

- Real getdist (const RealVector &x1, const RealVector &x2)
- Real mindist (const RealVector &x, const RealMatrix &xset, int except)
- Real mindistindx (const RealVector &x, const RealMatrix &xset, const IntArray &indx)
- Real getRmax (const RealMatrix &xset)
- int start_grid_computing (char *analysis_driver_script, char *params_file, char *results_file)
- int stop_grid_computing ()
- int perform_analysis (char *iteration_num)
- template<typename T>
  string asstring (const T &val)
  
  *Creates a string from the argument val using an ostringstream.*

- PACKBUF (int, MPI_INT) PACKBUF(u_int)
- MPI_UNSIGNED PACKBUF (long, MPI_LONG) PACKBUF(u_long)
CHAPTER 12. NAMESPACE DOCUMENTATION

- MPI_UNSIGNED MPI_UNSIGNED_LONG PACKBUF (short, MPI_SHORT) PACKBUF(u_short)
- MPI_UNSIGNED MPI_UNSIGNED_LONG
  MPI_UNSIGNED_SHORT PACKBUF (char, MPI_CHAR) PACKBUF(u_char)
- MPI_UNSIGNED MPI_UNSIGNED_LONG
  MPI_UNSIGNED_SHORT
  MPI_UNSIGNED_CHAR PACKBUF (double, MPI_DOUBLE) PACKBUF(float)
- UNPACKBUF (int, MPI_INT) UNPACKBUF(u_int)
- MPI_UNSIGNED UNPACKBUF (long, MPI_LONG) UNPACKBUF(u_long)
- MPI_UNSIGNED MPI_UNSIGNED_LONG UNPACKBUF (short, MPI_SHORT) UNPACKBUF(u_short)
- MPI_UNSIGNED MPI_UNSIGNED_LONG
  MPI_UNSIGNED_SHORT
  MPI_UNSIGNED_CHAR UNPACKBUF (double, MPI_DOUBLE) UNPACKBUF(float)
- PACKSIZE (int, MPI_INT) PACKSIZE(u_int)
- MPI_UNSIGNED PACKSIZE (long, MPI_LONG) PACKSIZE(u_long)
- MPI_UNSIGNED MPI_UNSIGNED_LONG PACKSIZE (short, MPI_SHORT) PACKSIZE(u_short)
- MPI_UNSIGNED MPI_UNSIGNED_LONG
  MPI_UNSIGNED_SHORT
  MPI_UNSIGNED_CHAR PACKSIZE (double, MPI_DOUBLE) PACKSIZE(float)
- MPI_UNSIGNED MPI_UNSIGNED_LONG
  MPI_UNSIGNED_SHORT
  MPI_UNSIGNED_CHAR MPI_FLOAT
  int MPIPackSize (const bool &data, const int num=1)
    return packed size of a bool
- 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
12.1. DAKOTA NAMESPACE REFERENCE

- **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

- template <class ContainerT>
  void container_read (ContainerT &c, MPIUnpackBuffer &s)
  Read a generic container (vector<T>, list<T>) from MPIUnpackBuffer, s.

- template <class ContainerT>
  void container_write (const ContainerT &c, MPIPackBuffer &s)
  Write a generic container to MPIPackBuffer, s.

- template <class ContainerT>
  MPIUnpackBuffer & operator >> (MPIUnpackBuffer &s, ContainerT &data)
  global MPIUnpackBuffer extraction operator for generic container

- template <class ContainerT>
  MPIPackBuffer & operator << (MPIPackBuffer &s, const ContainerT &data)
  global MPIPackBuffer insertion operator for generic container

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

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

- int MPIPackSize (const long &data, const int num=1)
• \texttt{int MPIPackSize (const \_u\_long &data, const int num=1)}
  \textit{return packed size of a \_u\_long}
• \texttt{int MPIPackSize (const \_short &data, const int num=1)}
  \textit{return packed size of a \_short}
• \texttt{int MPIPackSize (const \_u\_short &data, const int num=1)}
  \textit{return packed size of a \_u\_short}
• \texttt{int MPIPackSize (const \_char &data, const int num=1)}
  \textit{return packed size of a \_char}
• \texttt{int MPIPackSize (const \_u\_char &data, const int num=1)}
  \textit{return packed size of a \_u\_char}
• \texttt{int MPIPackSize (const \_double &data, const int num=1)}
  \textit{return packed size of a \_double}
• \texttt{int MPIPackSize (const \_float &data, const int num=1)}
  \textit{return packed size of a \_float}
• \texttt{int nidr\_parse (const char ∗, FILE ∗)}
• \texttt{int not\_executable (const char ∗driver\_name, const char ∗tdir)}
• \texttt{static void scale\_chk (StringArray ∗ST, RealVector ∗S, const char ∗what, const char ∗∗univ)}
• \texttt{static void BuildLabels (StringArray ∗sa, size\_t nsa, size\_t n1, size\_t n2, const char ∗stub)}
• \texttt{static int mixed\_check (IntSet ∗S, int n, IntArray ∗iv, const char ∗what)}
• \texttt{static void mixed\_check2 (size\_t n, IntArray ∗iv, const char ∗what)}
• \texttt{static int wronglen (size\_t n, RealVector ∗V, const char ∗what)}
• \texttt{static int wronglen (size\_t n, IntVector ∗V, const char ∗what)}
• \texttt{static void Vcopyup (RealVector ∗V, RealVector ∗M, size\_t i, size\_t n)}
• \texttt{static void Set\_rv (RealVector ∗V, double d, size\_t n)}
• \texttt{static void Set\_iv (IntVector ∗V, int d, size\_t n)}
• \texttt{static void wrong\_number (const char ∗what, const char ∗kind, size\_t nsv, size\_t m)}
• \texttt{static void too\_small (const char ∗kind)}
• \texttt{static void not\_div (const char ∗kind, size\_t nsv, size\_t m)}
• \texttt{static void suppressed (const char ∗kind, int ndup, int ∗ip, Real ∗rp)}
• \texttt{static void bad\_initial\_value (const char ∗kind, int val)}
• \texttt{static void bad\_initial\_value (const char ∗kind, Real val)}
• \texttt{static void Vgen\_ContinuousDes (DataVariablesRep ∗dv, size\_t offset)}
• \texttt{static void Vgen\_DiscreteDesRange (DataVariablesRep ∗dv, size\_t offset)}
• \texttt{static void Vgen\_ContinuousState (DataVariablesRep ∗dv, size\_t offset)}
• \texttt{static void Vgen\_DiscreteStateRange (DataVariablesRep ∗dv, size\_t offset)}
• \texttt{static void Vchk\_NormalUnc (DataVariablesRep ∗dv, size\_t offset, Var\_Info ∗vi)}
• \texttt{static void Vgen\_NormalUnc (DataVariablesRep ∗dv, size\_t offset)}
• \texttt{static void Vchk\_LognormalUnc (DataVariablesRep ∗dv, size\_t offset, Var\_Info ∗vi)}
• \texttt{static void Vgen\_LognormalUnc (DataVariablesRep ∗dv, size\_t offset)}
• \texttt{static void Vchk\_UniformUnc (DataVariablesRep ∗dv, size\_t offset, Var\_Info ∗vi)}
• \texttt{static void Vgen\_UniformUnc (DataVariablesRep ∗dv, size\_t offset)}
• \texttt{static void Vchk\_LoguniformUnc (DataVariablesRep ∗dv, size\_t offset, Var\_Info ∗vi)}
• \texttt{static void Vgen\_LoguniformUnc (DataVariablesRep ∗dv, size\_t offset)}
• \texttt{static void Vchk\_TriangularUnc (DataVariablesRep ∗dv, size\_t offset, Var\_Info ∗vi)}
12.1. DAKOTA NAMESPACE REFERENCE

- static void Vgen_TriangularUnc (DataVariablesRep *dv, size_t offset)
- static void Vchk_ExponentialUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_ExponentialUnc (DataVariablesRep *dv, size_t offset)
- static void Vchk_BetaUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_BetaUnc (DataVariablesRep *dv, size_t offset)
- static void Vchk_GammaUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_GammaUnc (DataVariablesRep *dv, size_t offset)
- static void Vchk_GumbelUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_GumbelUnc (DataVariablesRep *dv, size_t offset)
- static void Vchk_FrechetUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_FrechetUnc (DataVariablesRep *dv, size_t offset)
- static void Vchk_WeibullUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_WeibullUnc (DataVariablesRep *dv, size_t offset)
- static void Vchk_HistogramBinUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_HistogramBinUnc (DataVariablesRep *dv, size_t offset)
- static void Vchk_PoissonUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_PoissonUnc (DataVariablesRep *dv, size_t offset)
- static void Vchk_BinomialUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_BinomialUnc (DataVariablesRep *dv, size_t offset)
- static void Vchk_NegBinomialUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_NegBinomialUnc (DataVariablesRep *dv, size_t offset)
- static void Vchk_GeometricUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_GeometricUnc (DataVariablesRep *dv, size_t offset)
- static void Vchk_HyperGeomUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_HyperGeomUnc (DataVariablesRep *dv, size_t offset)
- static void Vchk_HistogramPtUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_HistogramPtUnc (DataVariablesRep *dv, size_t offset)
- static void Vchk_ContinuousIntervalUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_ContinuousIntervalUnc (DataVariablesRep *dv, size_t offset)
- static void Vchk_DiscreteIntervalUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_DiscreteIntervalUnc (DataVariablesRep *dv, size_t offset)
- static bool check_set_keys (size_t num_v, size_t ds_len, const char *kind, IntArray *input_nds, int &avg_-num_ds)
- static void Vchk_DLset (size_t num_v, const char *kind, IntArray *input_ndsi, IntVector *input_dsi, IntSetArray &dsi_all, IntVector &dsi_init_pt)
- static void Vchk_DLset (size_t num_v, const char *kind, IntArray *input_ndsi, IntVector *input_dsi, RealVector *input_dsp, IntRealMapArray &dsi_vals_probs, IntVector &dsi_init_pt)
- static void Vchk_DLSet (size_t num_v, const char *kind, IntArray *input_ndsr, RealVector *input_dsr, RealSetArray &dsr_all, RealVector &dsr_init_pt)
- static void Vchk_DLSet (size_t num_v, const char *kind, IntArray *input_ndsr, RealVector *input_dsr, RealVector *input_dsp, RealMapArray &dsr_vals_probs, RealVector &dsr_init_pt)
- static bool check_LUV_size (size_t num_v, IntVector &L, IntVector &U, IntVector &V, bool aggregate_L_LUV, size_t offset)
- static bool check_LUV_size (size_t num_v, RealVector &L, RealVector &U, RealVector &V, bool aggregate_L_LUV, size_t offset)
- static void Vgen_DLset (size_t num_v, IntSetArray &sets, IntVector &L, IntVector &U, IntVector &V, bool aggregate_L_LUV=false, size_t offset=0)
static void Vgen_DLset (size_t num_v, IntRealMapArray &vals_probs, IntVector &IP, IntVector &L, IntVector &U, IntVector &V, bool aggregate, LUV=false, size_t offset=0)
static void Vgen_DLset (size_t num_v, RealSetArray &sets, RealVector &L, RealVector &U, RealVector &V, bool aggregate, LUV=false, size_t offset=0)
static void Vgen_DLset (size_t num_v, RealMapArray &vals_probs, RealVector &IP, RealVector &L, RealVector &U, RealVector &V, RealVector &V, bool aggregate, LUV=false, size_t offset=0)
static void Vgen_DLset (size_t num_v, RealVector &L, RealVector &U, RealVector &V, RealVector &V, bool aggregate, LUV=false, size_t offset=0)
static void Vchk_DiscrDisSetInt (DataVariablesRep *dv, size_t offset, Var_Info *vi)
static void Vchk_DiscrDisSetInt (DataVariablesRep *dv, size_t offset, Var_Info *vi)
static void Vchk_DiscrDisSetReal (DataVariablesRep *dv, size_t offset, Var_Info *vi)
static void Vchk_DiscrDisSetReal (DataVariablesRep *dv, size_t offset, Var_Info *vi)
static void Vchk_DiscrUncSetInt (DataVariablesRep *dv, size_t offset, Var_Info *vi)
static void Vchk_DiscrUncSetInt (DataVariablesRep *dv, size_t offset, Var_Info *vi)
static void Vchk_DiscrUncSetReal (DataVariablesRep *dv, size_t offset, Var_Info *vi)
static void Vchk_DiscrUncSetReal (DataVariablesRep *dv, size_t offset, Var_Info *vi)
static void Vchk_DiscrStateSetInt (DataVariablesRep *dv, size_t offset, Var_Info *vi)
static void Vchk_DiscrStateSetInt (DataVariablesRep *dv, size_t offset, Var_Info *vi)
static void Vchk_DiscrStateSetReal (DataVariablesRep *dv, size_t offset, Var_Info *vi)
static void Vchk_DiscrStateSetReal (DataVariablesRep *dv, size_t offset, Var_Info *vi)
static void Vchk_DiscrValue (DataVariablesRep *dv, size_t offset, Var_Info *vi)
static void Vchk_DiscrValue (DataVariablesRep *dv, size_t offset, Var_Info *vi)
static void Vchk_DiscrValue (DataVariablesRep *dv, size_t offset, Var_Info *vi)
static void Vchk_DiscrValue (DataVariablesRep *dv, size_t offset, Var_Info *vi)
static void flatten_num_rv (RealVectorArray *rv, IntArray **pia)
static void flatten_num_rva (RealVectorArray *rva, IntVector **piv)
static void flatten_num_rsa (RealSetArray *rsa, IntVector **piv)
static void flatten_num_rsm (RealSymMatrix *rsm, RealVector **piv)
static void flatten_num_rsa (RealSetArray *rsa, RealVector **piv)
static void flatten_num_isa (IntSetArray *isa, IntVector **piv)
static void flatten_num_irma (IntRealMapArray *irma, IntArray **pia)
static void flatten_num_rma (RealRealMapArray *rma, IntArray **piv)
static void flatten_num_rma_keys (RealRealMapArray *rma, RealVector **piv)
static void flatten_num_rma_values (RealRealMapArray *rma, RealVector **piv)
static void flatten_irma (IntRealMapArray *irma, IntVector **piv)
static void flatten_irma_keys (IntRealMapArray *irma, IntVector **piv)
static void flatten_irma_values (IntRealMapArray *irma, RealVector **piv)
static void var_iubv (const char *keyname, Values *val, VarLabel *vl)
static Iface_mp_Rlit MP3 (failAction, recoveryFnVals, recover)
static Iface_mp_llit MP2 (failAction, retryLimit, retry)
static Iface_mp_llit MP2 (failAction, abort)
static Iface_mp_llit MP2 (failAction, continuation)
static Iface_mp_llit MP2 (interfaceSynchronization, asynchronous)
static Iface_mp_llit MP2 (interfaceType, direct)
static Iface_mp_llit MP2 (interfaceType, forK)
static Iface_mp_llit MP2 (interfaceType, grid)
static Iface_mp_llit MP2 (interfaceType, matlab)
static Iface_mp_llit MP2 (interfaceType, python)
• static Iface_mp_lit MP2 (interfaceType, scilab)
• static Iface_mp_lit MP2 (interfaceType, system)
• static Iface_mp_type MP2 (analysisScheduling, MASTER_SCHEDULING)
• static Iface_mp_type MP2 (analysisScheduling, PEER_SCHEDULING)
• static Iface_mp_type MP2 (evalScheduling, MASTER_SCHEDULING)
• static Iface_mp_type MP2 (evalScheduling, PEER_DYNAMIC_SCHEDULING)
• static Iface_mp_type MP2 (evalScheduling, PEER_STATIC_SCHEDULING)
• static Iface_mp_type MP2 (asynchLocalEvalScheduling, DYNAMIC_SCHEDULING)
• static Iface_mp_type MP2 (asynchLocalEvalScheduling, STATIC_SCHEDULING)
• static String MP (algebraicMappings)
• static String MP (idInterface)
• static String MP (inputFilter)
• static String MP (outputFilter)
• static String MP (parametersFile)
• static String MP (resultsFile)
• static String MP (templateDir)
• static String MP (workDir)
• static String2DArray MP (analysisComponents)
• static StringArray MP (analysisDrivers)
• static StringArray MP (templateFiles)
• static bool MP (activeSetVectorFlag)
• static bool MP (allowExistingResultsFlag)
• static bool MP (apreproFlag)
• static bool MP (dirSave)
• static bool MP (dirTag)
• static bool MP (evalCacheFlag)
• static bool MP (fileSaveFlag)
• static bool MP (fileTagFlag)
• static bool MP (nearbyEvalCacheFlag)
• static bool MP (numpyFlag)
• static bool MP (restartFileFlag)
• static bool MP (templateCopy)
• static bool MP (templateReplace)
• 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 int MP (procsPerEval)
• static Real MP (nearbyEvalCacheTol)
• static IntVector MP (primeBase)
• static IntVector MP (sequenceLeap)
• static IntVector MP (sequenceStart)
• static IntVector MP (stepsPerVariable)
• static Method_mpilit2 MP3 (replacementType, numberRetained, chc)
• static Method_mpilit2 MP3 (replacementType, numberRetained, elitist)
• static Method_mpilit2 MP3 (replacementType, numberRetained, random)
• static Method_mpilit2z MP3 (crossoverType, numCrossPoints, multi_point_binary)
• static Method_mpilit2z MP3 (crossoverType, numCrossPoints, multi_point_parameterized_binary)
• static Method_mpilit2z MP3 (crossoverType, numCrossPoints, multi_point_real)
• static Method_mpilit MP2 (batchSelectionType, naive)
• static Method_mpilit MP2 (batchSelectionType, distance_penalty)
• static Method_mpilit MP2 (batchSelectionType, topology)
• static Method_mpilit MP2 (batchSelectionType, constant_liar)
• static Method_mpilit MP2 (boxDivision, all_dimensions)
• static Method_mpilit MP2 (boxDivision, major_dimension)
• static Method_mpilit MP2 (convergenceType, average_fitness_tracker)
• static Method_mpilit MP2 (convergenceType, best_fitness_tracker)
• static Method_mpilit MP2 (convergenceType, metric_tracker)
• static Method_mpilit MP2 (crossoverType, blend)
• static Method_mpilit MP2 (crossoverType, two_point)
• static Method_mpilit MP2 (crossoverType, uniform)
• static Method_mpilit MP2 (evalSynchronize, blocking)
• static Method_mpilit MP2 (evalSynchronize, nonblocking)
• static Method_mpilit MP2 (expansionSampleType, incremental_lhs)
• static Method_mpilit MP2 (exploratoryMoves, adaptive)
• static Method_mpilit MP2 (exploratoryMoves, multi_step)
• static Method_mpilit MP2 (exploratoryMoves, simple)
• static Method_mpilit MP2 (fitnessType, domination_count)
• static Method_mpilit MP2 (fitnessType, layer_rank)
• static Method_mpilit MP2 (fitnessType, linear_rank)
• static Method_mpilit MP2 (fitnessType, merit_function)
• static Method_mpilit MP2 (fitnessType, proportional)
• static Method_mpilit MP2 (fitnessMetricType, predicted_variance)
• static Method_mpilit MP2 (fitnessMetricType, distance)
• static Method_mpilit MP2 (fitnessMetricType, gradient)
• static Method_mpilit MP2 (initializationType, random)
• static Method_mpilit MP2 (initializationType, unique_random)
• static Method_mpilit MP2 (meritFunction, merit_max)
• static Method_mpilit MP2 (meritFunction, merit_max_smooth)
• static Method_mpilit MP2 (meritFunction, merit1)
• static Method_mpilit MP2 (meritFunction, merit1_smooth)
• static Method_mpilit MP2 (meritFunction, merit2)
• static Method_mpilit MP2 (meritFunction, merit2_smooth)
• static Method_mpilit MP2 (meritFunction, merit2_squared)
• static Method_mpilit MP2 (mcmcType, dram)
• static Method_mpilit MP2 (mcmcType, multilevel)
• static Method_mpilit MP2 (metropolisType, adaptive)
• static Method_mpilit MP2 (metropolisType, hastings)
• static Method_mpilit 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 (patternBasis, coordinate)
static Method\_mp\_lit MP2 (patternBasis, simplex)
static Method\_mp\_lit MP2 (pointReUse, all)
static Method\_mp\_lit MP2 (rejectionType, standard)
static Method\_mp\_lit MP2 (rejectionType, delayed)
static Method\_mp\_lit MP2 (reliabilityIntegration, first\_order)
static Method\_mp\_lit MP2 (reliabilityIntegration, second\_order)
static Method\_mp\_lit MP2 (replacementType, elitist)
static Method\_mp\_lit MP2 (replacementType, favor\_feasible)
static Method\_mp\_lit MP2 (replacementType, roulette\_wheel)
static Method\_mp\_lit MP2 (replacementType, unique\_roulette\_wheel)
static Method\_mp\_lit MP2 (rngName, mt19937)
static Method\_mp\_lit MP2 (rngName, rnum2)
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\_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\_litrv MP3 (nichingType, nicheVector, distance)
static Method\_mp\_litrv MP3 (nichingType, nicheVector, max\_designs)
static Method\_mp\_litrv MP3 (nichingType, nicheVector, radial)
static Method\_mp\_litrv MP3 (postProcessorType, distanceVector, distance\_postprocessor)
static Method\_mp\_slit2 MP3 (initializationType, flatFile, flat\_file)
static Method\_mp\_utype\_lit MP3s (methodName, dlDetails, DL\_SOLVER)
static Real MP\_ (absConvTol)
static Real MP\_ (centeringParam)
static Real MP\_ (collocationRatio)
static Real MP\_ (collocRatioTermsOrder)
static Real MP\_ (constraintPenalty)
static Real MP\_ (constrPenalty)
static Real MP\_ (constraintTolerance)
static Real MP\_ (contractFactor)
static Real MP\_ (contractStepLength)
static Real MP_ (convergenceTolerance)
static Real MP_ (crossoverRate)
static Real MP_ (falseConvTol)
static Real MP_ (functionPrecision)
static Real MP_ (globalBalanceParam)
static Real MP_ (gradientTolerance)
static Real MP_ (hybridLSProb)
static Real MP_ (grThreshold)
static Real MP_ (initDelta)
static Real MP_ (initStepLength)
static Real MP_ (initTRRadius)
static Real MP_ (likelihoodScale)
static Real MP_ (lineSearchTolerance)
static Real MP_ (localBalanceParam)
static Real MP_ (maxBoxSize)
static Real MP_ (maxStep)
static Real MP_ (minBoxSize)
static Real MP_ (mutationRate)
static Real MP_ (mutationScale)
static Real MP_ (refinementRate)
static Real MP_ (regressionL2Penalty)
static Real MP_ (shrinkagePercent)
static Real MP_ (singConvTol)
static Real MP_ (singRadius)
static Real MP_ (smoothFactor)
static Real MP_ (solnTarget)
static Real MP_ (stepLenToBoundary)
static Real MP_ (surrBasedLocalTRContract)
static Real MP_ (surrBasedLocalTRContractTrigger)
static Real MP_ (surrBasedLocalTRExpand)
static Real MP_ (surrBasedLocalTRExpandTrigger)
static Real MP_ (surrBasedLocalTRInitSize)
static Real MP_ (surrBasedLocalTRMinSize)
static Real MP_ (threshDelta)
static Real MP_ (threshStepLength)
static Real MP_ (vbdDropTolerance)
static Real MP_ (volBoxSize)
static Real MP_ (vns)
static Real MP_ (xConvTol)
static RealVector MP_ (anisoDimPref)
static RealVector MP_ (concurrentParameterSets)
static RealVector MP_ (finalPoint)
static RealVector MP_ (lineEqConstraintCoeffs)
static RealVector MP_ (lineEqScales)
static RealVector MP_ (lineEqTargets)
static RealVector MP_ (linearIneqConstraintCoeffs)
• static RealVector MP_ (linearIneqLowerBnds)
• static RealVector MP_ (linearIneqUpperBnds)
• static RealVector MP_ (linearIneqScales)
• static RealVector MP_ (listOfPoints)
• static RealVector MP_ (proposalCovScale)
• static RealVector MP_ (regressionNoiseTol)
• static RealVector MP_ (stepVector)
• static RealVectorArray MP_ (genReliabilityLevels)
• static RealVectorArray MP_ (probabilityLevels)
• static RealVectorArray MP_ (reliabilityLevels)
• static RealVectorArray MP_ (responseLevels)
• static unsigned short MP_ (cubIntOrder)
• static unsigned short MP_ (vbdOrder)
• static SizetArray MP_ (collocationPoints)
• static SizetArray MP_ (expansionSamples)
• static UShortArray MP_ (expansionOrder)
• static UShortArray MP_ (quadratureOrder)
• static UShortArray MP_ (sparseGridLevel)
• static UShortArray MP_ (tensorGridOrder)
• static UShortArray MP_ (varPartitions)
• static String MP_ (approxExportFile)
• static String MP_ (approxImportFile)
• static String MP_ (betaSolverName)
• static String MP_ (displayFormat)
• static String MP_ (expansionExportFile)
• static String MP_ (expansionImportFile)
• static String MP_ (historyFile)
• static String MP_ (hybridGlobalMethodName)
• static String MP_ (hybridGlobalMethodPointer)
• static String MP_ (hybridGlobalModelPointer)
• static String MP_ (hybridLocalMethodName)
• static String MP_ (hybridLocalModelPointer)
• static String MP_ (hybridLocalModelPointer)
• static String MP_ (idMethod)
• static String MP_ (logFile)
• static String MP_ (modelPointer)
• static String MP_ (pstudyFilename)
• static String MP_ (subMethodName)
• static String MP_ (subMethodPointer)
• static String MP_ (subModelPointer)
• static StringArray MP_ (hybridMethodNames)
• static StringArray MP_ (hybridMethodPointers)
• static StringArray MP_ (hybridModelPointers)
• static StringArray MP_ (linearEqScaleTypes)
• static StringArray MP_ (linearIneqScaleTypes)
• static bool MP_ (approxExportAnnotated)
• static bool MP_ (approxImportAnnotated)
• static bool MP_ (calibrateSigmaFlag)
• static bool MP_ (constantPenalty)
• static bool MP_ (crossValidation)
• static bool MP_ (expansionFlag)
• static bool MP_ (fixedSeedFlag)
• static bool MP_ (fixedSequenceFlag)
• static bool MP_ (latinizeFlag)
• static bool MP_ (mainEffectsFlag)
• static bool MP_ (methodScaling)
• static bool MP_ (methodUseDerivsFlag)
• static bool MP_ (mutationAdaptive)
• static bool MP_ (normalizedCoeffs)
• static bool MP_ (printPopFlag)
• static bool MP_ (pstudyFileAnnotated)
• static bool MP_ (randomizeOrderFlag)
• static bool MP_ (regressDiag)
• static bool MP_ (showAllEval)
• static bool MP_ (showMiscOptions)
• static bool MP_ (speculativeFlag)
• static bool MP_ (tensorGridFlag)
• static bool MP_ (surrBasedGlobalReplacePts)
• static bool MP_ (surrBasedLocalLayerBypass)
• static bool MP_ (vbdFlag)
• static bool MP_ (volQualityFlag)
• static short MP_ (expansionType)
• static short MP_ (nestingOverride)
• static short MP_ (refinementType)
• static int MP_ (batchSize)
• static int MP_ (concurrentRandomJobs)
• static int MP_ (contractAfterFail)
• static int MP_ (covarianceType)
• static int MP_ (crossoverChainPairs)
• static int MP_ (emulatorSamples)
• static int MP_ (expandAfterSuccess)
• static int MP_ (iteratorServers)
• static int MP_ (jumpStep)
• static int MP_ (maxFunctionEvaluations)
• static int MP_ (maxIterations)
• static int MP_ (mutationRange)
• static int MP_ (newSolnsGenerated)
• static int MP_ (numChains)
• static int MP_ (numCR)
• static int MP_ (numSamples)
• static int MP_ (numSteps)
• static int MP_{numSymbols}
• static int MP_{numTrials}
• static int MP_{populationSize}
• static int MP_{previousSamples}
• static int MP_{procsPerIterator}
• static int MP_{randomSeed}
• static int MP_{refineSamples}
• static int MP_{searchSchemeSize}
• static int MP_{surrBasedLocalSoftConvLimit}
• static int MP_{totalPatternSize}
• static int MP_{verifyLevel}
• static size_t MP_{numDesigns}
• static size_t MP_{numFinalSolutions}
• static size_t MP_{numGenerations}
• static size_t MP_{numOffspring}
• static size_t MP_{numParents}
• static Method_mp_type MP2s_{covarianceControl, DIAGONAL_COVARIANCE}
• static Method_mp_type MP2s_{covarianceControl, FULL_COVARIANCE}
• static Method_mp_type MP2s_{distributionType, COMPLEMENTARY}
• static Method_mp_type MP2s_{distributionType, CUMULATIVE}
• static Method_mp_type MP2s_{emulatorType, GP_EMULATOR}
• static Method_mp_type MP2s_{emulatorType, KRIGING_EMULATOR}
• static Method_mp_type MP2s_{emulatorType, PCE_EMULATOR}
• static Method_mp_type MP2s_{emulatorType, SC_EMULATOR}
• static Method_mp_type MP2p_{expansionBasisType, ADAPTED_BASIS}
• static Method_mp_type MP2p_{expansionBasisType, HIERARCHICAL_INTERPOLANT}
• static Method_mp_type MP2p_{expansionBasisType, NODAL_INTERPOLANT}
• static Method_mp_type MP2p_{expansionBasisType, TENSOR_PRODUCT_BASIS}
• static Method_mp_type MP2p_{expansionBasisType, TOTAL_ORDER_BASIS}
• static Method_mp_type MP2s_{expansionType, ASKEY_U}
• static Method_mp_type MP2s_{expansionType, STD_NORMAL_U}
• static Method_mp_type MP2p_{growthOverride, RESTRICTED}
• static Method_mp_type MP2p_{growthOverride, UNRESTRICTED}
• static Method_mp_type MP2s_{iterScheduling, MASTER_SCHEDULING}
• static Method_mp_type MP2s_{iterScheduling, PEER_SCHEDULING}
• static Method_mp_type MP2s_{isRegressionType, EQ_CON_LS}
• static Method_mp_type MP2s_{isRegressionType, SVD_LS}
• static Method_mp_type MP2o_{meritFn, ArgaezTapia}
• static Method_mp_type MP2o_{meritFn, NormFmu}
• static Method_mp_type MP2o_{meritFn, VanShanno}
• static Method_mp_type MP2s_{methodOutput, DEBUG_OUTPUT}
• static Method_mp_type MP2s_{methodOutput, NORMAL_OUTPUT}
• static Method_mp_type MP2s_{methodOutput, QUIET_OUTPUT}
• static Method_mp_type MP2s_{methodOutput, SILENT_OUTPUT}
• static Method_mp_type MP2s_{methodOutput, VERBOSE_OUTPUT}
• static Method_mp_type MP2p_{nestingOverride, NESTED}
• static Method\_mp\_type \texttt{MP2p} (nestingOverride, NON\_NESTED)
• static Method\_mp\_type \texttt{MP2p} (refinementControl, DIMENSION\_ADAPTIVE\_CONTROL\_GENERALIZED)
• static Method\_mp\_type \texttt{MP2p} (refinementControl, DIMENSION\_ADAPTIVE\_CONTROL\_DECAY)
• static Method\_mp\_type \texttt{MP2p} (refinementControl, DIMENSION\_ADAPTIVE\_CONTROL\_SOBOL)
• static Method\_mp\_type \texttt{MP2p} (refinementControl, LOCAL\_ADAPTIVE\_CONTROL)
• static Method\_mp\_type \texttt{MP2p} (refinementControl, UNIFORM\_CONTROL)
• static Method\_mp\_type \texttt{MP2p} (refinementType, P\_REFINEMENT)
• static Method\_mp\_type \texttt{MP2p} (refinementType, H\_REFINEMENT)
• static Method\_mp\_type \texttt{MP2s} (responseLevelTarget, GEN\_RELIABILITIES)
• static Method\_mp\_type \texttt{MP2s} (responseLevelTarget, PROBABILITIES)
• static Method\_mp\_type \texttt{MP2s} (responseLevelTarget, RELIABILITIES)
• static Method\_mp\_type \texttt{MP2s} (responseLevelTargetReduce, SYSTEM\_PARALLEL)
• static Method\_mp\_type \texttt{MP2s} (responseLevelTargetReduce, SYSTEM\_SERIES)
• static Method\_mp\_type \texttt{MP2s} (surrBasedLocalAcceptLogic, FILTER)
• static Method\_mp\_type \texttt{MP2s} (surrBasedLocalAcceptLogic, TR\_RATIO)
• static Method\_mp\_type \texttt{MP2s} (surrBasedLocalConstrRelax, HOMOTOPY)
• static Method\_mp\_type \texttt{MP2s} (surrBasedLocalMeritFn, ADAPTIVE\_PENALTY\_MERIT)
• static Method\_mp\_type \texttt{MP2s} (surrBasedLocalMeritFn, AUGMENTED\_LAGRANGIAN\_MERIT)
• static Method\_mp\_type \texttt{MP2s} (surrBasedLocalMeritFn, LAGRANGIAN\_MERIT)
• static Method\_mp\_type \texttt{MP2s} (surrBasedLocalMeritFn, PENALTY\_MERIT)
• static Method\_mp\_type \texttt{MP2s} (surrBasedLocalSubProbCon, LINEARIZED\_CONSTRAINTS)
• static Method\_mp\_type \texttt{MP2s} (surrBasedLocalSubProbCon, NO\_CONSTRAINTS)
• static Method\_mp\_type \texttt{MP2s} (surrBasedLocalSubProbCon, ORIGINAL\_CONSTRAINTS)
• static Method\_mp\_type \texttt{MP2s} (surrBasedLocalSubProbObj, AUGMENTED\_LAGRANGIAN\_OBJECTIVE)
• static Method\_mp\_type \texttt{MP2s} (surrBasedLocalSubProbObj, LAGRANGIAN\_OBJECTIVE)
• static Method\_mp\_type \texttt{MP2s} (surrBasedLocalSubProbObj, ORIGINAL\_PRIMARY)
• static Method\_mp\_type \texttt{MP2s} (surrBasedLocalSubProbObj, SINGLE\_OBJECTIVE)
• static Method\_mp\_type \texttt{MP2s} (integrationRefine, AIS)
• static Method\_mp\_type \texttt{MP2s} (integrationRefine, IS)
• static Method\_mp\_type \texttt{MP2s} (integrationRefine, MMAIS)
• static Method\_mp\_type \texttt{MP2s} (methodName, ASYNCH\_PATTERN\_SEARCH)
• static Method\_mp\_type \texttt{MP2s} (methodName, COLINY\_BETA)
• static Method\_mp\_type \texttt{MP2s} (methodName, COLINY\_COBYLA)
• static Method\_mp\_type \texttt{MP2s} (methodName, COLINY\_DIRECT)
• static Method\_mp\_type \texttt{MP2s} (methodName, COLINY\_EA)
• static Method\_mp\_type \texttt{MP2s} (methodName, COLINY\_PATTERN\_SEARCH)
• static Method\_mp\_type \texttt{MP2s} (methodName, COLINY\_SOLIS\_WETS)
• static Method `mp_utype` MP2s (methodName, CONMIN_FRCG)
• static Method `mp_utype` MP2s (methodName, CONMIN_MFD)
• static Method `mp_utype` MP2s (methodName, DACE)
• static Method `mp_utype` MP2s (methodName, DOT_BFGS)
• static Method `mp Utpe` MP2s (methodName, DOT_FRCG)
• static Method `mp utpe` MP2s (methodName, DOT_MMFD)
• static Method `mp utpe` MP2s (methodName, DOT_SLP)
• static Method `mp utpe` MP2s (methodName, DOT_SQP)
• static Method `mp utpe` MP2s (methodName, EFFICIENT_GLOBAL)
• static Method `mp utpe` MP2s (methodName, FSU_CVT)
• static Method `mp utpe` MP2s (methodName, FSU_HALTON)
• static Method `mp utpe` MP2s (methodName, FSU_HAMMERSLEY)
• static Method `mp utpe` MP2s (methodName, HYBRID)
• static Method `mp utpe` MP2s (methodName, MESH_ADAPTIVE_SEARCH)
• static Method `mp utpe` MP2s (methodName, MOGA)
• static Method `mp utpe` MP2s (methodName, MULTI_START)
• static Method `mp utpe` MP2s (methodName, NCSU_DIRECT)
• static Method `mp utpe` MP2s (methodName, NL2SOL)
• static Method `mp utpe` MP2s (methodName, NLPQL_SQP)
• static Method `mp utpe` MP2s (methodName, NLSSOL_SQP)
• static Method `mp utpe` MP2s (methodName, ADAPTIVE_SAMPLING)
• static Method `mp utpe` MP2s (methodName, BAYES_CALIBRATION)
• static Method `mp utpe` MP2s (methodName, EFFICIENT_SUBSPACE)
• static Method `mp utpe` MP2s (methodName, GENIE_DIRECT)
• static Method `mp utpe` MP2s (methodName, GENIE_OPT_DARTS)
• static Method `mp utpe` MP2s (methodName, GPAIS)
• static Method `mp utpe` MP2s (methodName, GLOBAL_EVIDENCE)
• static Method `mp utpe` MP2s (methodName, GLOBAL_INTERVAL_EST)
• static Method `mp utpe` MP2s (methodName, GLOBAL_RELIABILITY)
• static Method `mp utpe` MP2s (methodName, IMPORTANCE_SAMPLING)
• static Method `mp utpe` MP2s (methodName, LOCAL_EVIDENCE)
• static Method `mp utpe` MP2s (methodName, LOCAL_INTERVAL_EST)
• static Method `mp utpe` MP2s (methodName, LOCAL_RELIABILITY)
• static Method `mp utpe` MP2s (methodName, POF_DARTS)
• static Method `mp utpe` MP2s (methodName, POLYNOMIAL_CHAOS)
• static Method `mp utpe` MP2s (methodName, RANDOM_SAMPLING)
• static Method `mp utpe` MP2s (methodName, STOCH_COLLOCATION)
• static Method `mp utpe` MP2s (methodName, NONLINEAR_CG)
• static Method `mp utpe` MP2s (methodName, NPSOL_SQP)
• static Method `mp utpe` MP2s (methodName, OPTPP_CG)
• static Method `mp utpe` MP2s (methodName, OPTPP_FD_NEWTON)
• static Method `mp utpe` MP2s (methodName, OPTPP_G_NEWTON)
• static Method `mp utpe` MP2s (methodName, OPTPP_NEWTON)
• static Method `mp utpe` MP2s (methodName, OPTPP_PDS)
• static Method `mp utpe` MP2s (methodName, OPTPP_Q_NEWTON)
• static Method `mp utpe` MP2s (methodName, PARETO_SET)
• static Method\_mp\_utype MP2s (methodName, PSU\_QE, MO\_AT)
• static Method\_mp\_utype MP2s (methodName, RICH\_SON, EXTRAP)
• static Method\_mp\_utype MP2s (methodName, SO\_GA)
• static Method\_mp\_utype MP2s (methodName, SUR\_RO\_GATE\_BASE\_AD\_ED, GLOBAL)
• static Method\_mp\_utype MP2s (methodName, SUR\_RO\_GATE\_BASE\_AD\_ED, LOCAL)
• static Method\_mp\_utype MP2s (methodName, VECTOR\_PARAMETER\_STUDY)
• static Method\_mp\_utype MP2s (methodName, LIST\_PARAMETER\_STUDY)
• static Method\_mp\_utype MP2s (methodName, CENTERED\_PARAMETER\_STUDY)
• static Method\_mp\_utype MP2s (methodName, MULTIDIM\_PARAMETER\_STUDY)
• static Method\_mp\_utype MP2s (reliabilitySearchType, AMV\_PLUS\_U)
• static Method\_mp\_utype MP2s (reliabilitySearchType, AMV\_PLUS\_X)
• static Method\_mp\_utype MP2s (reliabilitySearchType, AMV\_U)
• static Method\_mp\_utype MP2s (reliabilitySearchType, AMV\_X)
• static Method\_mp\_utype MP2s (reliabilitySearchType, EGRA\_U)
• static Method\_mp\_utype MP2s (reliabilitySearchType, EGRA\_X)
• static Method\_mp\_utype MP2s (reliabilitySearchType, NO\_APPROX)
• static Method\_mp\_utype MP2s (reliabilitySearchType, TANA\_U)
• static Method\_mp\_utype MP2s (reliabilitySearchType, TANA\_X)
• static Method\_mp\_utype MP2s (sampleType, SUBMETHOD\_INCREMENTAL\_LHS)
• static Method\_mp\_utype MP2s (sampleType, SUBMETHOD\_INCREMENTAL\_RANDOM)
• static Method\_mp\_utype MP2s (sampleType, SUBMETHOD\_LHS)
• static Method\_mp\_utype MP2s (sampleType, SUBMETHOD\_RANDOM)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_COLLABORATIVE)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_EMBEDDED)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_SEQUENTIAL)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_DREAM)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_GPMSA)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_QUESO)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_NIP)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_SBO)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_SBO)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_LHS)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_LHS)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_OA\_LHS)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_OA\_LHS)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_BOX\_BEHNKEN)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_CENTRAL\_COMPOSITE)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_GRID)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_CONVERGE\_ORDER)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_CONVERGE\_QOI)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_ESTIMATE\_ORDER)
• static IntSet MP\_surrogateFnIndices
• static Model\_mp\_lit MP2 (approxPoint\_Reuse, all)
• static Model\_mp\_lit MP2 (approxPoint\_Reuse, none)
• static Model_mp_lit MP2 (approxPointReuse, region)
• static Model_mp_lit MP2 (marsInterpolation, linear)
• static Model_mp_lit MP2 (marsInterpolation, cubic)
• static Model_mp_lit MP2 (modelType, nested)
• static Model_mp_lit MP2 (modelType, single)
• static Model_mp_lit MP2 (modelType, surrogate)
• static Model_mp_lit MP2 (surrogateType, hierarchical)
• static Model_mp_lit MP2 (surrogateType, global_gaussian)
• static Model_mp_lit MP2 (surrogateType, global_kriging)
• static Model_mp_lit MP2 (surrogateType, global_mars)
• static Model_mp_lit MP2 (surrogateType, global_moving_least_squares)
• static Model_mp_lit MP2 (surrogateType, global_neural_network)
• static Model_mp_lit MP2 (surrogateType, global_radial_basis)
• static Model_mp_lit MP2 (surrogateType, local_taylor)
• static Model_mp_lit MP2 (surrogateType, multipoint_tana)
• static Model_mp_lit MP2 (trendOrder, constant)
• static Model_mp_lit MP2 (trendOrder, linear)
• static Model_mp_lit MP2 (trendOrder, reduced_quadratic)
• static Model_mp_lit MP2 (trendOrder, quadratic)
• static Model_mp_ord MP2s (approxCorrectionOrder, 0)
• static Model_mp_ord MP2s (approxCorrectionOrder, 1)
• static Model_mp_ord MP2s (approxCorrectionOrder, 2)
• static Model_mp_ord MP2s (polynomialOrder, 1)
• static Model_mp_ord MP2s (polynomialOrder, 2)
• static Model_mp_ord MP2s (polynomialOrder, 3)
• static Model_mp_type MP2s (approxCorrectionType, ADDITIVE_CORRECTION)
• static Model_mp_type MP2s (approxCorrectionType, COMBINED_CORRECTION)
• static Model_mp_type MP2s (approxCorrectionType, MULTIPLICATIVE_CORRECTION)
• static Model_mp_type MP2s (pointsManagement, MINIMUM_POINTS)
• static Model_mp_type MP2s (pointsManagement, RECOMMENDED_POINTS)
• static Model_mp_type MP2s (subMethodScheduling, MASTER_SCHEDULING)
• static Model_mp_type MP2s (subMethodScheduling, PEER_SCHEDULING)
• static Real MP_ (annRange)
• static Real MP_ (krigingNugget)
• static Real MP_ (percentFold)
• static RealVector MP_ (krigingCorrelations)
• static RealVector MP_ (primaryRespCoeffs)
• static RealVector MP_ (secondaryRespCoeffs)
• static String MP_ (approxChallengeFile)
• static String MP_ (approxExportModelFile)
• static String MP_ (idModel)
• static String MP_ (interfacePointer)
• static String MP_ (krigingOptMethod)
• static String MP_ (lowFidelityModelPointer)
• static String MP_ (optionalInterfRespPointer)
• static String $\text{MP}_r$ (responsesPointer)
• static String $\text{MP}_r$ (truthModelPointer)
• static String $\text{MP}_r$ (variablesPointer)
• static StringArray $\text{MP}_r$ (primaryVarMaps)
• static StringArray $\text{MP}_r$ (secondaryVarMaps)
• static StringArray $\text{MP}_r$ (diagMetrics)
• static bool $\text{MP}_r$ (approxChallengeAnnotated)
• static bool $\text{MP}_r$ (crossValidateFlag)
• static bool $\text{MP}_r$ (hierarchicalTags)
• static bool $\text{MP}_r$ (modelUseDerivsFlag)
• static bool $\text{MP}_r$ (pressFlag)
• static bool $\text{MP}_r$ (pointSelection)
• static short $\text{MP}_r$ (annNodes)
• static short $\text{MP}_r$ (annRandomWeight)
• static short $\text{MP}_r$ (krigingFindNugget)
• static short $\text{MP}_r$ (krigingMaxTrials)
• static short $\text{MP}_r$ (marsMaxBases)
• static short $\text{MP}_r$ (mlsPolyOrder)
• static short $\text{MP}_r$ (mlsWeightFunction)
• static short $\text{MP}_r$ (rbfBases)
• static short $\text{MP}_r$ (rbfMaxPts)
• static short $\text{MP}_r$ (rbfMaxSubsets)
• static short $\text{MP}_r$ (rbfMinPartition)
• static int $\text{MP}_r$ (numFolds)
• static int $\text{MP}_r$ (pointsTotal)
• static int $\text{MP}_r$ (subMethodProcs)
• static int $\text{MP}_r$ (subMethodServers)
• static IntSet $\text{MP}_r$ (idAnalyticGrads)
• static IntSet $\text{MP}_r$ (idAnalyticHessians)
• static IntSet $\text{MP}_r$ (idNumericalGrads)
• static IntSet $\text{MP}_r$ (idNumericalHessians)
• static IntSet $\text{MP}_r$ (idQuasiHessians)
• static IntVector $\text{MP}_r$ (numReplicates)
• static RealVector $\text{MP}_r$ (expConfigVars)
• static RealVector $\text{MP}_r$ (expObservations)
• static RealVector $\text{MP}_r$ (expStdDeviations)
• static RealVector $\text{MP}_r$ (primaryRespFnWeights)
• static RealVector $\text{MP}_r$ (nonlinearEqTargets)
• static RealVector $\text{MP}_r$ (nonlinearIneqLowerBnds)
• static RealVector $\text{MP}_r$ (nonlinearIneqUpperBnds)
• static RealVector $\text{MP}_r$ (fdGradStepSize)
• static RealVector $\text{MP}_r$ (fdHessStepSize)
• static RealVector $\text{MP}_r$ (primaryRespFnScales)
• static RealVector $\text{MP}_r$ (nonlinearEqScales)
• static RealVector $\text{MP}_r$ (nonlinearIneqScales)
• static Resp $\text{mp}_2$ (gradientType, analytic)
• static Resp_mp_lit MP2 (gradientType, mixed)
• static Resp_mp_lit MP2 (gradientType, none)
• static Resp_mp_lit MP2 (gradientType, numerical)
• static Resp_mp_lit MP2 (hessianType, analytic)
• static Resp_mp_lit MP2 (hessianType, mixed)
• static Resp_mp_lit MP2 (hessianType, none)
• static Resp_mp_lit MP2 (hessianType, numerical)
• static Resp_mp_lit MP2 (hessianType, quasi)
• static Resp_mp_lit MP2 (intervalType, central)
• static Resp_mp_lit MP2 (intervalType, forward)
• static Resp_mp_lit MP2 (methodSource, dakota)
• static Resp_mp_lit MP2 (methodSource, vendor)
• static Resp_mp_lit MP2 (fdGradStepType, absolute)
• static Resp_mp_lit MP2 (fdGradStepType, bounds)
• static Resp_mp_lit MP2 (fdGradStepType, relative)
• static Resp_mp_lit MP2 (fdHessStepType, absolute)
• static Resp_mp_lit MP2 (fdHessStepType, bounds)
• static Resp_mp_lit MP2 (fdHessStepType, relative)
• static Resp_mp_lit MP2 (quasiHessianType, bfgs)
• static Resp_mp_lit MP2 (quasiHessianType, damped_bfgs)
• static Resp_mp_lit MP2 (quasiHessianType, sr1)
• static String MP_ (expDataFileName)
• static String MP_ (idResponses)
• static StringArray MP_ (nonlinearEqScaleTypes)
• static StringArray MP_ (nonlinearIneqScaleTypes)
• static StringArray MP_ (primaryRespFnScaleTypes)
• static StringArray MP_ (primaryRespFnSense)
• static StringArray MP_ (responseLabels)
• static bool MP_ (centralHess)
• static bool MP_ (expDataFileAnnotated)
• static bool MP_ (ignoreBounds)
• static size_t MP_ (numExpStdDeviations)
• static size_t MP_ (numExpConfigVars)
• static size_t MP_ (numExperiments)
• static size_t MP_ (numLeastSqTerms)
• static size_t MP_ (numNonlinearEqConstraints)
• static size_t MP_ (numNonlinearIneqConstraints)
• static size_t MP_ (numObjectiveFunctions)
• static size_t MP_ (numResponseFunctions)
• static String MP_ (resultsOutputFile)
• static String MP_ (tabularDataFile)
• static String MP_ (topMethodPointer)
• static bool MP_ (graphicsFlag)
• static bool MP_ (resultsOutputFlag)
• static bool MP_ (tabularDataFlag)
• static int MP_ (outputPrecision)
- static size_t  \texttt{MP\_}(\text{numBetaUncVars})
- static size_t  \texttt{MP\_}(\text{numBinomialUncVars})
- static size_t  \texttt{MP\_}(\text{numContinuousDesVars})
- static size_t  \texttt{MP\_}(\text{numContinuousIntervalUncVars})
- static size_t  \texttt{MP\_}(\text{numContinuousStateVars})
- static size_t  \texttt{MP\_}(\text{numDiscreteDesRangeVars})
- static size_t  \texttt{MP\_}(\text{numDiscreteDesSetIntVars})
- static size_t  \texttt{MP\_}(\text{numDiscreteDesSetRealVars})
- static size_t  \texttt{MP\_}(\text{numDiscreteIntervalUncVars})
- static size_t  \texttt{MP\_}(\text{numDiscreteStateRangeVars})
- static size_t  \texttt{MP\_}(\text{numDiscreteStateSetIntVars})
- static size_t  \texttt{MP\_}(\text{numDiscreteStateSetRealVars})
- static size_t  \texttt{MP\_}(\text{numDiscreteUncSetIntVars})
- static size_t  \texttt{MP\_}(\text{numDiscreteUncSetRealVars})
- static size_t  \texttt{MP\_}(\text{numExponentialUncVars})
- static size_t  \texttt{MP\_}(\text{numFrechetUncVars})
- static size_t  \texttt{MP\_}(\text{numGammaUncVars})
- static size_t  \texttt{MP\_}(\text{numGeometricUncVars})
- static size_t  \texttt{MP\_}(\text{numGumbelUncVars})
- static size_t  \texttt{MP\_}(\text{numHistogramBinUncVars})
- static size_t  \texttt{MP\_}(\text{numHistogramPtUncVars})
- static size_t  \texttt{MP\_}(\text{numHyperGeomUncVars})
- static size_t  \texttt{MP\_}(\text{numLognormalUncVars})
- static size_t  \texttt{MP\_}(\text{numLoguniformUncVars})
- static size_t  \texttt{MP\_}(\text{numNegBinomialUncVars})
- static size_t  \texttt{MP\_}(\text{numNormalUncVars})
- static size_t  \texttt{MP\_}(\text{numPoissonUncVars})
- static size_t  \texttt{MP\_}(\text{numTriangularUncVars})
- static size_t  \texttt{MP\_}(\text{numUniformUncVars})
- static size_t  \texttt{MP\_}(\text{numWeibullUncVars})
- static IntVector  \texttt{VP\_}(\text{ddsi})
- static IntVector  \texttt{VP\_}(\text{DIlb})
- static IntVector  \texttt{MP\_}(\text{discreteDesignRangeLowerBnds})
- static IntVector  \texttt{MP\_}(\text{discreteDesignRangeUpperBnds})
- static IntVector  \texttt{MP\_}(\text{discreteDesignRangeVars})
- static IntVector  \texttt{MP\_}(\text{discreteDesignSetIntVars})
- static IntVector  \texttt{MP\_}(\text{discreteIntervalUncVars})
- static IntVector  \texttt{MP\_}(\text{discreteStateRangeLowerBnds})
- static IntVector  \texttt{MP\_}(\text{discreteStateRangeUpperBnds})
- static IntVector  \texttt{MP\_}(\text{discreteStateRangeVars})
- static IntVector  \texttt{MP\_}(\text{discreteStateSetIntVars})
- static IntVector  \texttt{MP\_}(\text{discreteUncSetIntVars})
- static IntVector  \texttt{VP\_}(\text{Dlub})
- static IntVector  \texttt{VP\_}(\text{dssi})
- static IntVector  \texttt{VP\_}(\text{dusi})
- static IntArray  \texttt{VP\_}(\text{nddsi})
- static IntArray VP_ (nddsr)
- static IntArray VP_ (ndssi)
- static IntArray VP_ (ndssr)
- static IntArray VP_ (ndusi)
- static IntArray VP_ (ndusr)
- static IntArray VP_ (nhbp)
- static IntArray VP_ (nhpp)
- static IntArray VP_ (nCI)
- static IntArray VP_ (nDI)
- static RealVector MP_ (betaUncLowerBnds)
- static RealVector MP_ (betaUncUpperBnds)
- static RealVector MP_ (betaUncVars)
- static RealVector MP_ (binomialUncProbPerTrial)
- static RealVector MP_ (continuousDesignLowerBnds)
- static RealVector MP_ (continuousDesignUpperBnds)
- static RealVector MP_ (continuousDesignVars)
- static RealVector MP_ (continuousDesignScales)
- static RealVector MP_ (continuousIntervalUncVars)
- static RealVector MP_ (continuousStateLowerBnds)
- static RealVector MP_ (continuousStateUpperBnds)
- static RealVector MP_ (continuousStateVars)
- static RealVector MP_ (discreteDesignSetRealVars)
- static RealVector MP_ (discreteStateSetRealVars)
- static RealVector MP_ (discreteUncSetRealVars)
- static RealVector MP_ (frechetUncBetas)
- static RealVector MP_ (frechetUncVars)
- static RealVector MP_ (geometricUncProbPerTrial)
- static RealVector MP_ (gumbelUncBetas)
- static RealVector MP_ (gumbelUncVars)
- static RealVector MP_ (histogramBinUncVars)
- static RealVector MP_ (histogramPointUncVars)
- static RealVector MP_ (negBinomialUncProbPerTrial)
- static RealVector MP_ (normalUncLowerBnds)
- static RealVector MP_ (normalUncMeans)
- static RealVector MP_ (normalUncUpperBnds)
- static RealVector MP_ (normalUncVars)
- static RealVector MP_ (triangularUncModes)
- static RealVector MP_ (triangularUncVars)
- static RealVector MP_ (uniformUncVars)
- static RealVector MP_ (weibullUncVars)
- static RealVector VP_ (ddsr)
- static RealVector VP_ (dssr)
- static RealVector VP_ (dusr)
- static RealVector VP_ (CIlb)
- static RealVector VP_ (CIB)
- static RealVector VP_ (CIP)
• static RealVector VP_(DIp)
• static RealVector VP_(DSIp)
• static RealVector VP_(DSRp)
• static RealVector VP_(hba)
• static RealVector VP_(hbo)
• static RealVector VP_(hbc)
• static RealVector VP_(hpa)
• static RealVector VP_(hpc)
• static RealVector VP_(ucm)
• static String MP_(idVariables)
• static StringArray MP_(continuousDesignLabels)
• static StringArray MP_(continuousDesignScaleTypes)
• static StringArray MP_(continuousStateLabels)
• static StringArray MP_(discreteDesignRangeLabels)
• static StringArray MP_(discreteDesignSetIntLabels)
• static StringArray MP_(discreteDesignSetRealLabels)
• static StringArray MP_(discreteStateRangeLabels)
• static StringArray MP_(discreteStateSetIntLabels)
• static StringArray MP_(discreteStateSetRealLabels)
• static Var brv MP2s (betaUncAlphas, 0.)
• static Var brv MP2s (betaUncBetas, 0.)
• static Var brv MP2s (exponentialUncBetas, 0.)
• static Var brv MP2s (exponentialUncVars, 0.)
• static Var brv MP2s (frechetUncAlphas, 2.)
• static Var brv MP2s (gammaUncAlphas, 0.)
• static Var brv MP2s (gammaUncBetas, 0.)
• static Var brv MP2s (gammaUncVars, 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)
• static Var brv MP2s (lognormalUncVars, 0.)
• static Var brv MP2s (lognormalUncZetas, 0.)
• static Var brv MP2s (loguniformUncLowerBnds, 0.)
• static Var brv MP2s (loguniformUncUpperBnds, DBL_MAX)
• static Var brv MP2s (loguniformUncVars, 0.)
• static Var brv MP2s (loguniformUncLambdas, 0.)
• static Var brv MP2s (normalUncStdDevs, 0.)
• static Var brv MP2s (poissonUncLambdas, 0.)
• static Var brv MP2s (triangularUncLowerBnds, -DBL_MAX)
• static Var brv MP2s (triangularUncUpperBnds, DBL_MAX)
• static Var brv MP2s (uniformUncLowerBnds, -DBL_MAX)
• static Var brv MP2s (uniformUncUpperBnds, DBL_MAX)
• static Var brv MP2s (weibullUncAlphas, 0.)
• static Var_brv MP2s (weibullUncBetas, 0)
• static Var_biv MP2s (binomialUncNumTrials, 0)
• static Var_biv MP2s (binomialUncVars, 0)
• static Var_biv MP2s (geometricUncVars, 0)
• static Var_biv MP2s (hyperGeomUncNumDrawn, 0)
• static Var_biv MP2s (hyperGeomUncSelectedPop, 0)
• static Var_biv MP2s (hyperGeomUncTotalPop, 0)
• static Var_biv MP2s (hyperGeomUncVars, 0)
• static Var_biv MP2s (negBinomialUncNumTrials, 0)
• static Var_biv MP2s (negBinomialUncVars, 0)
• static Var_biv MP2s (poissonUncVars, 0)
• static Var_mp_type Vtype (varsDomain, MIXED_DOMAIN)
• static Var_mp_type Vtype (varsDomain, RELAXED_DOMAIN)
• static Var_mp_type Vtype (varsView, ALL_VIEW)
• static Var_mp_type Vtype (varsView, DESIGN_VIEW)
• static Var_mp_type Vtype (varsView, UNCERTAIN_VIEW)
• static Var_mp_type Vtype (varsView, ALEATORY_UNCERTAIN_VIEW)
• static Var_mp_type Vtype (varsView, EPISTEMIC_UNCERTAIN_VIEW)
• static Var_mp_type Vtype (varsView, STATE_VIEW)
• void dn2f_ (int *, int *, Real *, Calcrj, int *, lv, Real *, int *, v, void *, ur, Vf)
• void dn2fb_ (int *, int *, Real *, x, Calcrj, int *, lv, Real *, int *, ui, void *, ur, Vf)
• void dn2g_ (int *, int *, Real *, x, Calcrj, Calcrj, int *, lv, Real *, int *, ui, void *, ur, Vf)
• void dn2gb_ (int *, int *, Real *, x, Real *, b, Calcrj, Calcrj, int *, lv, Real *, int *, ui, void *

12.1. DAKOTA NAMESPACE REFERENCE
• **NPSOLOptimizer** *new_NPSOLOptimizer* (const RealVector &initial_point, const RealVector &var_lower_bnds, const RealVector &var_upper_bnds, const RealMatrix &lin_ineq_coeffs, const RealVector &lin_ineq_lower_bnds, const RealVector &lin_ineq_upper_bnds, const RealMatrix &lin_eq_coeffs, const RealVector &lin_eq_targets, const RealVector &nonlin_ineq_lower_bnds, const RealVector &nonlin_ineq_upper_bnds, const RealVector &nonlin_eq_targets, void(*user_obj_eval)(int &, int &, double *, double &, double *, int &), void(*user_con_eval)(int &, int &, int &, int &, int &, int *, double *, double *, double *, int &), const Real &conv_tol)

• 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

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

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

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

• bool operator!=(const ParamResponsePair &pair1, const ParamResponsePair &pair2)
  inequality operator for ParamResponsePair

• static void *binsearch (void *kw, size_t ksize, 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)

• MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, ProgramOptions &progopt)
  MPIUnpackBuffer extraction operator.

• MPIPackBuffer & operator<<(MPIPackBuffer &s, const ProgramOptions &progopt)
  MPIPackBuffer insertion operator.

• bool set_compare (const ParamResponsePair &database_pr, const ActiveSet &search_set)
  search function for a particular ParamResponsePair within a PRPList based on ActiveSet content (request vector and derivative variables vector)

• bool id_vars_exact_compare (const ParamResponsePair &database_pr, const ParamResponsePair &search_pr)
  search function for a particular ParamResponsePair within a PRPMultiIndex

• std::size_t hash_value (const ParamResponsePair &prp)
  hash value for ParamResponsePairs stored in a PRPMultiIndex

• PRPCacheHIter hashedCacheBegin (PRPCache &prp_cache)
  hashed definition of cache begin

• PRPCacheHIter hashedCacheEnd (PRPCache &prp_cache)
  hashed definition of cache end

• PRPQueueHIter hashedQueueBegin (PRPQueue &prp_queue)
  hashed definition of queue begin

• PRPQueueHIter hashedQueueEnd (PRPQueue &prp_queue)
hashed definition of queue end

- **PRPCacheHIter lookup by val** (PRPMultiIndexCache &prp_cache, const ParamResponsePair &search_pr)
  find a ParamResponsePair based on the interface id, variables, and ActiveSet search data within search_pr.

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

- **PRPCacheOIter lookup by nearby val** (PRPMultiIndexCache &prp_cache, const String &search_interface_id, const Variables &search_vars, const ActiveSet &search_set, Real tol)

- **PRPCacheOIter lookup by ids** (PRPMultiIndexCache &prp_cache, const IntStringPair &search_ids)

- **PRPCacheOIter lookup by ids** (PRPMultiIndexCache &prp_cache, const IntStringPair &search_ids, const ParamResponsePair &search_pr)

- **PRPQueueHIter lookup by val** (PRPMultiIndexQueue &prp_queue, const ParamResponsePair &search_pr)

- **PRPQueueHIter lookup by val** (PRPMultiIndexQueue &prp_queue, const String &search_interface_id, const Variables &search_vars, const ActiveSet &search_set)

- **PRPQueueHIter lookup by eval id** (PRPMultiIndexQueue &prp_queue, int search_id)

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

- **void print restart pdb** (int argc, char **argv, String print_dest)
  print a restart file (PDB format)

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

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

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

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

- **static HANDLE * wait_setup** (std::map< pid_t, int > *M, size_t *pn)

- **static int wait_for_one** (size_t n, HANDLE *h, int req1, size_t *pi)

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

- **void find env token** (const char *s0, const char **s1, const char **s2, const char **s3)

- **const char ** arg_list_adjust** (const char **, void **)
  Utility function from legacy, "not executable" module – DO NOT TOUCH!

- **bool contains** (const bfs::path &dir_path, const std::string &file_name, boost::filesystem::path &complete_filepath)
  Helper for "which" - sets complete_filepath from dir_path/file_name combo.
CHAPTER 12. NAMESPACE DOCUMENTATION

Variables

- PRPCache `data_pairs`
  contains all parameter/response pairs.
- static time `start_time`
- static int `dakdrive`
- static char `slmap` [256]
- `std::ostream∗ dakota_cout = &std::cout`  
  DAKOTA stdout initially points to `<std::cout, but may be redirected to a tagged ofstream if there are < concurrent iterators.
- `std::ostream∗ dakota_cerr = &std::cerr`  
  DAKOTA stderr initially points to `<std::cerr, but may be redirected to a tagged ofstream if there are < concurrent iterators.
- ResultsManager `iterator_results_db`
  Global results database for iterator results.
- Graphics `dakota_graphics`
  the global Dakota::Graphics object used by `<strategies, models, and approximations`
- int `write_precision = 10`  
  used in ostream data output functions `<(restart_util.cpp overrides default value)`
- MPlManager `dummy_mpi_mgr`
  dummy MPlManager for ref initialization
- ProgramOptions `dummy_prg_opt`
  dummy ProgramOptions for ref initialization
- OutputManager `dummy_out_mgr`
  dummy OutputManager for ref initialization
- ParallelLibrary `dummy_lib`
  dummy ParallelLibrary for ref initialization
- ProblemDescDB `dummy_db`
  dummy ProblemDescDB for ref initialization
- int `mc_ptr_int = 0`
  global pointer for ModelCenter API
- int `dc_ptr_int = 0`
  global pointer for ModelCenter eval DB
- ProblemDescDB * `Dak_pdb`
  set by ProblemDescDB, for use in parsing
- const `size_t NPOS = ~(size_t)0`
  special value returned by index() when entry not found
- Interface `dummy_interface`
  dummy Interface object used for mandatory
  < reference initialization or default virtual < function return by reference when a real < Interface instance is unavailable
- Model `dummy_model`
  dummy Model object used for mandatory reference
  < initialization or default virtual function < return by reference when a real Model instance < is unavailable
- Iterator `dummy_iterator`
dummy Iterator object used for mandatory <reference initialization or default virtual <function return by reference when a real < Iterator instance is unavailable

- Dakota_funcs * DF
- Dakota_funcs DakFuncs0
- const char * FIELD_NAMES []
- const int NUMBER_OF_FIELDS = 23
- static GuiKeyWord kw_1 [1]
- static GuiKeyWord kw_2 [1]
- static GuiKeyWord kw_3 [6]
- static GuiKeyWord kw_4 [1]
- static GuiKeyWord kw_5 [4]
- static GuiKeyWord kw_6 [1]
- static GuiKeyWord kw_7 [4]
- static GuiKeyWord kw_8 [1]
- static GuiKeyWord kw_9 [2]
- static GuiKeyWord kw_10 [7]
- static GuiKeyWord kw_11 [9]
- static GuiKeyWord kw_12 [12]
- static GuiKeyWord kw_13 [2]
- static GuiKeyWord kw_14 [2]
- static GuiKeyWord kw_15 [3]
- static GuiKeyWord kw_16 [2]
- static GuiKeyWord kw_17 [2]
- static GuiKeyWord kw_18 [9]
- static GuiKeyWord kw_19 [2]
- static GuiKeyWord kw_20 [1]
- static GuiKeyWord kw_21 [1]
- static GuiKeyWord kw_22 [2]
- static GuiKeyWord kw_23 [4]
- static GuiKeyWord kw_24 [2]
- static GuiKeyWord kw_25 [3]
- static GuiKeyWord kw_26 [2]
- static GuiKeyWord kw_27 [2]
- static GuiKeyWord kw_28 [3]
- static GuiKeyWord kw_29 [2]
- static GuiKeyWord kw_30 [15]
- static GuiKeyWord kw_31 [7]
- static GuiKeyWord kw_32 [2]
- static GuiKeyWord kw_33 [19]
- static GuiKeyWord kw_34 [2]
- static GuiKeyWord kw_35 [2]
- static GuiKeyWord kw_36 [5]
- static GuiKeyWord kw_37 [11]
- static GuiKeyWord kw_38 [1]
- static GuiKeyWord kw_39 [4]
• static GuiKeyWord kw_40 [6]
• static GuiKeyWord kw_41 [2]
• static GuiKeyWord kw_42 [2]
• static GuiKeyWord kw_43 [2]
• static GuiKeyWord kw_44 [2]
• static GuiKeyWord kw_45 [2]
• static GuiKeyWord kw_46 [2]
• static GuiKeyWord kw_47 [2]
• static GuiKeyWord kw_48 [6]
• static GuiKeyWord kw_49 [2]
• static GuiKeyWord kw_50 [2]
• static GuiKeyWord kw_51 [5]
• static GuiKeyWord kw_52 [1]
• static GuiKeyWord kw_53 [1]
• static GuiKeyWord kw_54 [4]
• static GuiKeyWord kw_55 [4]
• static GuiKeyWord kw_56 [9]
• static GuiKeyWord kw_57 [4]
• static GuiKeyWord kw_58 [7]
• static GuiKeyWord kw_59 [8]
• static GuiKeyWord kw_60 [2]
• static GuiKeyWord kw_61 [12]
• static GuiKeyWord kw_62 [3]
• static GuiKeyWord kw_63 [2]
• static GuiKeyWord kw_64 [3]
• static GuiKeyWord kw_65 [2]
• static GuiKeyWord kw_66 [5]
• static GuiKeyWord kw_67 [4]
• static GuiKeyWord kw_68 [15]
• static GuiKeyWord kw_69 [3]
• static GuiKeyWord kw_70 [2]
• static GuiKeyWord kw_71 [2]
• static GuiKeyWord kw_72 [18]
• static GuiKeyWord kw_73 [14]
• static GuiKeyWord kw_74 [12]
• static GuiKeyWord kw_75 [10]
• static GuiKeyWord kw_76 [1]
• static GuiKeyWord kw_77 [15]
• static GuiKeyWord kw_78 [15]
• static GuiKeyWord kw_79 [2]
• static GuiKeyWord kw_80 [2]
• static GuiKeyWord kw_81 [2]
• static GuiKeyWord kw_82 [7]
• static GuiKeyWord kw_83 [9]
• static GuiKeyWord kw_84 [3]
• static GuiKeyWord kw_85 [1]
• static GuiKeyWord kw_86 [9]
• static GuiKeyWord kw_87 [1]
• static GuiKeyWord kw_88 [11]
• static GuiKeyWord kw_89 [2]
• static GuiKeyWord kw_90 [2]
• static GuiKeyWord kw_91 [2]
• static GuiKeyWord kw_92 [3]
• static GuiKeyWord kw_93 [2]
• static GuiKeyWord kw_94 [11]
• static GuiKeyWord kw_95 [2]
• static GuiKeyWord kw_96 [2]
• static GuiKeyWord kw_97 [3]
• static GuiKeyWord kw_98 [2]
• static GuiKeyWord kw_99 [2]
• static GuiKeyWord kw_100 [2]
• static GuiKeyWord kw_101 [2]
• static GuiKeyWord kw_102 [5]
• static GuiKeyWord kw_103 [12]
• static GuiKeyWord kw_104 [2]
• static GuiKeyWord kw_105 [2]
• static GuiKeyWord kw_106 [2]
• static GuiKeyWord kw_107 [2]
• static GuiKeyWord kw_108 [5]
• static GuiKeyWord kw_109 [8]
• static GuiKeyWord kw_110 [2]
• static GuiKeyWord kw_111 [1]
• static GuiKeyWord kw_112 [1]
• static GuiKeyWord kw_113 [2]
• static GuiKeyWord kw_114 [2]
• static GuiKeyWord kw_115 [2]
• static GuiKeyWord kw_116 [3]
• static GuiKeyWord kw_117 [2]
• static GuiKeyWord kw_118 [2]
• static GuiKeyWord kw_119 [16]
• static GuiKeyWord kw_120 [2]
• static GuiKeyWord kw_121 [1]
• static GuiKeyWord kw_122 [2]
• static GuiKeyWord kw_123 [1]
• static GuiKeyWord kw_124 [1]
• static GuiKeyWord kw_125 [5]
• static GuiKeyWord kw_126 [1]
• static GuiKeyWord kw_127 [2]
• static GuiKeyWord kw_128 [8]
• static GuiKeyWord kw_129 [2]
• static GuiKeyWord kw_130 [3]
• static GuiKeyWord kw_131 [2]
• static GuiKeyWord kw_132 [12]
• static GuiKeyWord kw_133 [2]
• static GuiKeyWord kw_134 [3]
• static GuiKeyWord kw_135 [2]
• static GuiKeyWord kw_136 [1]
• static GuiKeyWord kw_137 [1]
• static GuiKeyWord kw_138 [2]
• static GuiKeyWord kw_139 [3]
• static GuiKeyWord kw_140 [2]
• static GuiKeyWord kw_141 [7]
• static GuiKeyWord kw_142 [3]
• static GuiKeyWord kw_143 [5]
• static GuiKeyWord kw_144 [4]
• static GuiKeyWord kw_145 [10]
• static GuiKeyWord kw_146 [1]
• static GuiKeyWord kw_147 [2]
• static GuiKeyWord kw_148 [4]
• static GuiKeyWord kw_149 [2]
• static GuiKeyWord kw_150 [7]
• static GuiKeyWord kw_151 [16]
• static GuiKeyWord kw_152 [2]
• static GuiKeyWord kw_153 [5]
• static GuiKeyWord kw_154 [3]
• static GuiKeyWord kw_155 [1]
• static GuiKeyWord kw_156 [6]
• static GuiKeyWord kw_157 [3]
• static GuiKeyWord kw_158 [2]
• static GuiKeyWord kw_159 [1]
• static GuiKeyWord kw_160 [3]
• static GuiKeyWord kw_161 [1]
• static GuiKeyWord kw_162 [2]
• static GuiKeyWord kw_163 [4]
• static GuiKeyWord kw_164 [22]
• static GuiKeyWord kw_165 [1]
• static GuiKeyWord kw_166 [1]
• static GuiKeyWord kw_167 [7]
• static GuiKeyWord kw_168 [2]
• static GuiKeyWord kw_169 [5]
• static GuiKeyWord kw_170 [10]
• static GuiKeyWord kw_171 [2]
• static GuiKeyWord kw_172 [3]
• static GuiKeyWord kw_173 [2]
• static GuiKeyWord kw_174 [8]
• static GuiKeyWord kw_175 [1]
• static GuiKeyWord kw_176 [2]
• static GuiKeyWord kw_177 [4]
12.1. DAKOTA NAMESPACE REFERENCE

- static GuiKeyWord kw_178 [2]
- static GuiKeyWord kw_179 [2]
- static GuiKeyWord kw_180 [3]
- static GuiKeyWord kw_181 [1]
- static GuiKeyWord kw_182 [1]
- static GuiKeyWord kw_183 [2]
- static GuiKeyWord kw_184 [2]
- static GuiKeyWord kw_185 [1]
- static GuiKeyWord kw_186 [17]
- static GuiKeyWord kw_187 [3]
- static GuiKeyWord kw_188 [6]
- static GuiKeyWord kw_189 [2]
- static GuiKeyWord kw_190 [6]
- static GuiKeyWord kw_191 [3]
- static GuiKeyWord kw_192 [2]
- static GuiKeyWord kw_193 [4]
- static GuiKeyWord kw_194 [3]
- static GuiKeyWord kw_195 [2]
- static GuiKeyWord kw_196 [5]
- static GuiKeyWord kw_197 [2]
- static GuiKeyWord kw_198 [30]
- static GuiKeyWord kw_199 [1]
- static GuiKeyWord kw_200 [4]
- static GuiKeyWord kw_201 [1]
- static GuiKeyWord kw_202 [12]
- static GuiKeyWord kw_203 [2]
- static GuiKeyWord kw_204 [2]
- static GuiKeyWord kw_205 [3]
- static GuiKeyWord kw_206 [2]
- static GuiKeyWord kw_207 [2]
- static GuiKeyWord kw_208 [4]
- static GuiKeyWord kw_209 [2]
- static GuiKeyWord kw_210 [4]
- static GuiKeyWord kw_211 [2]
- static GuiKeyWord kw_212 [28]
- static GuiKeyWord kw_213 [2]
- static GuiKeyWord kw_214 [13]
- static GuiKeyWord kw_215 [12]
- static GuiKeyWord kw_216 [11]
- static GuiKeyWord kw_217 [3]
- static GuiKeyWord kw_218 [4]
- static GuiKeyWord kw_219 [16]
- static GuiKeyWord kw_220 [5]
- static GuiKeyWord kw_221 [2]
- static GuiKeyWord kw_222 [1]
- static GuiKeyWord kw_223 [10]
• static GuiKeyWord kw_224 [4]
• static GuiKeyWord kw_225 [5]
• static GuiKeyWord kw_226 [2]
• static GuiKeyWord kw_227 [2]
• static GuiKeyWord kw_228 [2]
• static GuiKeyWord kw_229 [2]
• static GuiKeyWord kw_230 [4]
• static GuiKeyWord kw_231 [20]
• static GuiKeyWord kw_232 [15]
• static GuiKeyWord kw_233 [7]
• static GuiKeyWord kw_234 [2]
• static GuiKeyWord kw_235 [7]
• static GuiKeyWord kw_236 [1]
• static GuiKeyWord kw_237 [4]
• static GuiKeyWord kw_238 [6]
• static GuiKeyWord kw_239 [13]
• static GuiKeyWord kw_240 [4]
• static GuiKeyWord kw_241 [90]
• static GuiKeyWord kw_242 [1]
• static GuiKeyWord kw_243 [2]
• static GuiKeyWord kw_244 [7]
• static GuiKeyWord kw_245 [2]
• static GuiKeyWord kw_246 [1]
• static GuiKeyWord kw_247 [2]
• static GuiKeyWord kw_248 [6]
• static GuiKeyWord kw_249 [2]
• static GuiKeyWord kw_250 [2]
• static GuiKeyWord kw_251 [2]
• static GuiKeyWord kw_252 [3]
• static GuiKeyWord kw_253 [2]
• static GuiKeyWord kw_254 [4]
• static GuiKeyWord kw_255 [7]
• static GuiKeyWord kw_256 [2]
• static GuiKeyWord kw_257 [2]
• static GuiKeyWord kw_258 [2]
• static GuiKeyWord kw_259 [3]
• static GuiKeyWord kw_260 [3]
• static GuiKeyWord kw_261 [4]
• static GuiKeyWord kw_262 [4]
• static GuiKeyWord kw_263 [5]
• static GuiKeyWord kw_264 [3]
• static GuiKeyWord kw_265 [21]
• static GuiKeyWord kw_266 [6]
• static GuiKeyWord kw_267 [3]
• static GuiKeyWord kw_268 [2]
• static GuiKeyWord kw_269 [2]
12.1. DAKOTA NAMESPACE REFERENCE

- static GuiKeyWord kw_270 [5]
- static GuiKeyWord kw_271 [7]
- static GuiKeyWord kw_272 [6]
- static GuiKeyWord kw_273 [6]
- static GuiKeyWord kw_274 [8]
- static GuiKeyWord kw_275 [15]
- static GuiKeyWord kw_276 [4]
- static GuiKeyWord kw_277 [10]
- static GuiKeyWord kw_278 [2]
- static GuiKeyWord kw_279 [1]
- static GuiKeyWord kw_280 [2]
- static GuiKeyWord kw_281 [8]
- static GuiKeyWord kw_282 [6]
- static GuiKeyWord kw_283 [8]
- static GuiKeyWord kw_284 [11]
- static GuiKeyWord kw_285 [8]
- static GuiKeyWord kw_286 [7]
- static GuiKeyWord kw_287 [1]
- static GuiKeyWord kw_288 [2]
- static GuiKeyWord kw_289 [19]
- static GuiKeyWord kw_290 [6]
- static GuiKeyWord kw_291 [11]
- static GuiKeyWord kw_292 [5]
- static GuiKeyWord kw_293 [12]
- static GuiKeyWord kw_294 [10]
- static GuiKeyWord kw_295 [8]
- static GuiKeyWord kw_296 [8]
- static GuiKeyWord kw_297 [4]
- static GuiKeyWord kw_298 [4]
- static GuiKeyWord kw_299 [9]
- static GuiKeyWord kw_300 [8]
- static GuiKeyWord kw_301 [4]
- static GuiKeyWord kw_302 [4]
- static GuiKeyWord kw_303 [6]
- static GuiKeyWord kw_304 [6]
- static GuiKeyWord kw_305 [5]
- static GuiKeyWord kw_306 [7]
- static GuiKeyWord kw_307 [7]
- static GuiKeyWord kw_308 [4]
- static GuiKeyWord kw_309 [7]
- static GuiKeyWord kw_310 [11]
- static GuiKeyWord kw_311 [9]
- static GuiKeyWord kw_312 [5]
- static GuiKeyWord kw_313 [2]
- static GuiKeyWord kw_314 [4]
- static GuiKeyWord kw_315 [11]
• static GuiKeyWord kw_316 [7]
• static GuiKeyWord kw_317 [5]
• static GuiKeyWord kw_318 [11]
• static GuiKeyWord kw_319 [3]
• static GuiKeyWord kw_320 [9]
• static GuiKeyWord kw_321 [7]
• static GuiKeyWord kw_322 [7]
• static GuiKeyWord kw_323 [37]
• static GuiKeyWord kw_324 [6]
• static KeyWord kw_1 [1]
• static KeyWord kw_2 [1]
• static KeyWord kw_3 [6]
• static KeyWord kw_4 [1]
• static KeyWord kw_5 [4]
• static KeyWord kw_6 [1]
• static KeyWord kw_7 [4]
• static KeyWord kw_8 [1]
• static KeyWord kw_9 [2]
• static KeyWord kw_10 [7]
• static KeyWord kw_11 [9]
• static KeyWord kw_12 [12]
• static KeyWord kw_13 [2]
• static KeyWord kw_14 [2]
• static KeyWord kw_15 [3]
• static KeyWord kw_16 [2]
• static KeyWord kw_17 [2]
• static KeyWord kw_18 [9]
• static KeyWord kw_19 [1]
• static KeyWord kw_20 [2]
• static KeyWord kw_21 [2]
• static KeyWord kw_22 [1]
• static KeyWord kw_23 [1]
• static KeyWord kw_24 [2]
• static KeyWord kw_25 [4]
• static KeyWord kw_26 [4]
• static KeyWord kw_27 [2]
• static KeyWord kw_28 [3]
• static KeyWord kw_29 [2]
• static KeyWord kw_30 [2]
• static KeyWord kw_31 [3]
• static KeyWord kw_32 [2]
• static KeyWord kw_33 [11]
• static KeyWord kw_34 [9]
• static KeyWord kw_35 [7]
• static KeyWord kw_36 [2]
• static KeyWord kw_37 [11]
• static KeyWord kw_38 [2]
• static KeyWord kw_39 [2]
• static KeyWord kw_40 [5]
• static KeyWord kw_41 [1]
• static KeyWord kw_42 [1]
• static KeyWord kw_43 [4]
• static KeyWord kw_44 [6]
• static KeyWord kw_45 [2]
• static KeyWord kw_46 [2]
• static KeyWord kw_47 [2]
• static KeyWord kw_48 [2]
• static KeyWord kw_49 [2]
• static KeyWord kw_50 [3]
• static KeyWord kw_51 [2]
• static KeyWord kw_52 [2]
• static KeyWord kw_53 [4]
• static KeyWord kw_54 [2]
• static KeyWord kw_55 [2]
• static KeyWord kw_56 [5]
• static KeyWord kw_57 [1]
• static KeyWord kw_58 [1]
• static KeyWord kw_59 [4]
• static KeyWord kw_60 [2]
• static KeyWord kw_61 [8]
• static KeyWord kw_62 [4]
• static KeyWord kw_63 [5]
• static KeyWord kw_64 [3]
• static KeyWord kw_65 [2]
• static KeyWord kw_66 [4]
• static KeyWord kw_67 [2]
• static KeyWord kw_68 [8]
• static KeyWord kw_69 [3]
• static KeyWord kw_70 [2]
• static KeyWord kw_71 [3]
• static KeyWord kw_72 [2]
• static KeyWord kw_73 [5]
• static KeyWord kw_74 [4]
• static KeyWord kw_75 [11]
• static KeyWord kw_76 [2]
• static KeyWord kw_77 [3]
• static KeyWord kw_78 [2]
• static KeyWord kw_79 [2]
• static KeyWord kw_80 [12]
• static KeyWord kw_81 [8]
• static KeyWord kw_82 [4]
• static KeyWord kw_83 [3]
• static KeyWord kw_84 [1]
• static KeyWord kw_85 [14]
• static KeyWord kw_86 [7]
• static KeyWord kw_87 [2]
• static KeyWord kw_88 [2]
• static KeyWord kw_89 [2]
• static KeyWord kw_90 [7]
• static KeyWord kw_91 [5]
• static KeyWord kw_92 [3]
• static KeyWord kw_93 [1]
• static KeyWord kw_94 [8]
• static KeyWord kw_95 [1]
• static KeyWord kw_96 [11]
• static KeyWord kw_97 [2]
• static KeyWord kw_98 [2]
• static KeyWord kw_99 [2]
• static KeyWord kw_100 [3]
• static KeyWord kw_101 [2]
• static KeyWord kw_102 [7]
• static KeyWord kw_103 [2]
• static KeyWord kw_104 [2]
• static KeyWord kw_105 [3]
• static KeyWord kw_106 [2]
• static KeyWord kw_107 [2]
• static KeyWord kw_108 [2]
• static KeyWord kw_109 [2]
• static KeyWord kw_110 [5]
• static KeyWord kw_111 [8]
• static KeyWord kw_112 [2]
• static KeyWord kw_113 [2]
• static KeyWord kw_114 [2]
• static KeyWord kw_115 [2]
• static KeyWord kw_116 [5]
• static KeyWord kw_117 [7]
• static KeyWord kw_118 [2]
• static KeyWord kw_119 [1]
• static KeyWord kw_120 [1]
• static KeyWord kw_121 [3]
• static KeyWord kw_122 [2]
• static KeyWord kw_123 [2]
• static KeyWord kw_124 [2]
• static KeyWord kw_125 [3]
• static KeyWord kw_126 [2]
• static KeyWord kw_127 [2]
• static KeyWord kw_128 [14]
• static KeyWord kw_129 [2]
12.1. DAKOTA NAMESPACE REFERENCE

- static KeyWord kw_130 [3]
- static KeyWord kw_131 [1]
- static KeyWord kw_132 [2]
- static KeyWord kw_133 [1]
- static KeyWord kw_134 [1]
- static KeyWord kw_135 [5]
- static KeyWord kw_136 [1]
- static KeyWord kw_137 [2]
- static KeyWord kw_138 [6]
- static KeyWord kw_139 [2]
- static KeyWord kw_140 [3]
- static KeyWord kw_141 [2]
- static KeyWord kw_142 [8]
- static KeyWord kw_143 [2]
- static KeyWord kw_144 [3]
- static KeyWord kw_145 [2]
- static KeyWord kw_146 [1]
- static KeyWord kw_147 [1]
- static KeyWord kw_148 [2]
- static KeyWord kw_149 [3]
- static KeyWord kw_150 [2]
- static KeyWord kw_151 [7]
- static KeyWord kw_152 [3]
- static KeyWord kw_153 [5]
- static KeyWord kw_154 [4]
- static KeyWord kw_155 [10]
- static KeyWord kw_156 [1]
- static KeyWord kw_157 [2]
- static KeyWord kw_158 [4]
- static KeyWord kw_159 [2]
- static KeyWord kw_160 [5]
- static KeyWord kw_161 [8]
- static KeyWord kw_162 [2]
- static KeyWord kw_163 [5]
- static KeyWord kw_164 [3]
- static KeyWord kw_165 [1]
- static KeyWord kw_166 [6]
- static KeyWord kw_167 [7]
- static KeyWord kw_168 [3]
- static KeyWord kw_169 [2]
- static KeyWord kw_170 [1]
- static KeyWord kw_171 [3]
- static KeyWord kw_172 [1]
- static KeyWord kw_173 [2]
- static KeyWord kw_174 [4]
- static KeyWord kw_175 [8]
• static KeyWord kw_176 [1]
• static KeyWord kw_177 [1]
• static KeyWord kw_178 [5]
• static KeyWord kw_179 [2]
• static KeyWord kw_180 [5]
• static KeyWord kw_181 [10]
• static KeyWord kw_182 [2]
• static KeyWord kw_183 [3]
• static KeyWord kw_184 [2]
• static KeyWord kw_185 [4]
• static KeyWord kw_186 [1]
• static KeyWord kw_187 [2]
• static KeyWord kw_188 [4]
• static KeyWord kw_189 [2]
• static KeyWord kw_190 [2]
• static KeyWord kw_191 [1]
• static KeyWord kw_192 [2]
• static KeyWord kw_193 [1]
• static KeyWord kw_194 [3]
• static KeyWord kw_195 [1]
• static KeyWord kw_196 [1]
• static KeyWord kw_197 [2]
• static KeyWord kw_198 [2]
• static KeyWord kw_199 [1]
• static KeyWord kw_200 [17]
• static KeyWord kw_201 [3]
• static KeyWord kw_202 [6]
• static KeyWord kw_203 [2]
• static KeyWord kw_204 [6]
• static KeyWord kw_205 [3]
• static KeyWord kw_206 [2]
• static KeyWord kw_207 [4]
• static KeyWord kw_208 [3]
• static KeyWord kw_209 [2]
• static KeyWord kw_210 [3]
• static KeyWord kw_211 [2]
• static KeyWord kw_212 [25]
• static KeyWord kw_213 [1]
• static KeyWord kw_214 [4]
• static KeyWord kw_215 [1]
• static KeyWord kw_216 [7]
• static KeyWord kw_217 [2]
• static KeyWord kw_218 [2]
• static KeyWord kw_219 [3]
• static KeyWord kw_220 [2]
• static KeyWord kw_221 [2]
static KeyWord kw_222 [4]
static KeyWord kw_223 [2]
static KeyWord kw_224 [4]
static KeyWord kw_225 [2]
static KeyWord kw_226 [23]
static KeyWord kw_227 [2]
static KeyWord kw_228 [3]
static KeyWord kw_229 [4]
static KeyWord kw_230 [2]
static KeyWord kw_231 [4]
static KeyWord kw_232 [3]
static KeyWord kw_233 [3]
static KeyWord kw_234 [4]
static KeyWord kw_235 [7]
static KeyWord kw_236 [5]
static KeyWord kw_237 [2]
static KeyWord kw_238 [1]
static KeyWord kw_239 [8]
static KeyWord kw_240 [3]
static KeyWord kw_241 [5]
static KeyWord kw_242 [2]
static KeyWord kw_243 [2]
static KeyWord kw_244 [2]
static KeyWord kw_245 [2]
static KeyWord kw_246 [4]
static KeyWord kw_247 [6]
static KeyWord kw_248 [5]
static KeyWord kw_249 [7]
static KeyWord kw_250 [2]
static KeyWord kw_251 [7]
static KeyWord kw_252 [1]
static KeyWord kw_253 [4]
static KeyWord kw_254 [6]
static KeyWord kw_255 [13]
static KeyWord kw_256 [4]
static KeyWord kw_257 [90]
static KeyWord kw_258 [1]
static KeyWord kw_259 [2]
static KeyWord kw_260 [7]
static KeyWord kw_261 [2]
static KeyWord kw_262 [1]
static KeyWord kw_263 [2]
static KeyWord kw_264 [6]
static KeyWord kw_265 [2]
static KeyWord kw_266 [2]
static KeyWord kw_267 [2]
• static KeyWord kw_268 [3]
• static KeyWord kw_269 [2]
• static KeyWord kw_270 [4]
• static KeyWord kw_271 [7]
• static KeyWord kw_272 [2]
• static KeyWord kw_273 [2]
• static KeyWord kw_274 [2]
• static KeyWord kw_275 [3]
• static KeyWord kw_276 [3]
• static KeyWord kw_277 [4]
• static KeyWord kw_278 [4]
• static KeyWord kw_279 [5]
• static KeyWord kw_280 [3]
• static KeyWord kw_281 [21]
• static KeyWord kw_282 [6]
• static KeyWord kw_283 [3]
• static KeyWord kw_284 [1]
• static KeyWord kw_285 [2]
• static KeyWord kw_286 [2]
• static KeyWord kw_287 [5]
• static KeyWord kw_288 [7]
• static KeyWord kw_289 [6]
• static KeyWord kw_290 [6]
• static KeyWord kw_291 [8]
• static KeyWord kw_292 [15]
• static KeyWord kw_293 [4]
• static KeyWord kw_294 [8]
• static KeyWord kw_295 [3]
• static KeyWord kw_296 [2]
• static KeyWord kw_297 [1]
• static KeyWord kw_298 [2]
• static KeyWord kw_299 [8]
• static KeyWord kw_300 [6]
• static KeyWord kw_301 [8]
• static KeyWord kw_302 [11]
• static KeyWord kw_303 [7]
• static KeyWord kw_304 [1]
• static KeyWord kw_305 [2]
• static KeyWord kw_306 [19]
• static KeyWord kw_307 [6]
• static KeyWord kw_308 [11]
• static KeyWord kw_309 [5]
• static KeyWord kw_310 [12]
• static KeyWord kw_311 [10]
• static KeyWord kw_312 [8]
• static KeyWord kw_313 [8]
• static KeyWord kw_314 [4]
• static KeyWord kw_315 [4]
• static KeyWord kw_316 [9]
• static KeyWord kw_317 [8]
• static KeyWord kw_318 [4]
• static KeyWord kw_319 [4]
• static KeyWord kw_320 [6]
• static KeyWord kw_321 [6]
• static KeyWord kw_322 [5]
• static KeyWord kw_323 [7]
• static KeyWord kw_324 [7]
• static KeyWord kw_325 [4]
• static KeyWord kw_326 [7]
• static KeyWord kw_327 [11]
• static KeyWord kw_328 [9]
• static KeyWord kw_329 [5]
• static KeyWord kw_330 [2]
• static KeyWord kw_331 [4]
• static KeyWord kw_332 [11]
• static KeyWord kw_333 [7]
• static KeyWord kw_334 [5]
• static KeyWord kw_335 [11]
• static KeyWord kw_336 [3]
• static KeyWord kw_337 [9]
• static KeyWord kw_338 [7]
• static KeyWord kw_339 [7]
• static KeyWord kw_340 [37]
• static KeyWord kw_341 [6]

FILE * nируется
• static const char * aln_scaletypes[] = {"auto", "log", "none", 0}
• static Var_uinfo CAUVLbl[CAUVar_Nkinds]
• static Var_uinfo DAUIVLbl[DAUIVar_Nkinds]
• static Var_uinfo DAURVLbl[DAURVar_Nkinds]
• static Var_uinfo CEUVLbl[CEUVar_Nkinds]
• static Var_uinfo DEUVLbl[DEUVar_Nkinds]
• static Var_uinfo DiscSetLbl[DiscSetVar_Nkinds]
• static VarLabelChk Vlch[]
• static VLreal VLR[N_VLR]
• static VLint VLI[N_VLI]
• static int VLR_aleatory[N_VLR] = {1, 0, 1, 0}
• static int VLI_aleatory[N_VLI] = {1, 0}
• static Var_check var_mp_check_cv[]
• static Var_check var_mp_check_dset[]
• static Var_check var_mp_check_cau[]
• static Var_check var_mp_check_dau[]
• static Var_check var_mp_check_daur [ ]
• static Var_check var_mp_check_ceu [ ]
• static Var_check var_mp_check_deui [ ]
• static Var_check var_mp_check_deur [ ]
• static Var_rcheck var_mp_cbound [ ]
• static Var_rcheck var_mp_drange [ ]
• const char * SCIFIELD_NAMES [ ]
• const int SCI_NUMBER_OF_FIELDS = 26
• const int LARGE_SCALE = 100
  a (perhaps arbitrary) definition of large scale; choose a large-scale algorithm if numVars >= LARGE_SCALE
• const double POW_VAL = 1.0
  offset used text book exponent: 1.0 is nominal, 1.4 used for B&B testing

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

12.1.2 Typedef Documentation
typedef bmi::multi_index_container<Dakota::ParamResponsePair, bmi::indexed_by<bmi::ordered_non_unique<bmi::tag<ordered>>, bmi::const_mem_fun<Dakota::ParamResponsePair, const IntStringPair&, &Dakota::ParamResponsePair::eval_interface_ids>, bmi::hashed_non_unique<bmi::tag<hashed>>, bmi::identity<Dakota::ParamResponsePair>, partial_prp_hash, partial_prp_equality>> > > PRPMultiIndexCache

Boost Multi-Index Container for globally caching ParamResponsePairs.
For a global cache, both evaluation and interface id’s are used for tagging ParamResponsePair records.

typedef bmi::multi_index_container<Dakota::ParamResponsePair, bmi::indexed_by<bmi::ordered_unique<bmi::tag<ordered>>, bmi::const_mem_fun<Dakota::ParamResponsePair, int, &Dakota::ParamResponsePair::eval_id>, bmi::hashed_non_unique<bmi::tag<hashed>>, bmi::identity<Dakota::ParamResponsePair>, partial_prp_hash, partial_prp_equality>> > > PRPMultiIndexQueue

Boost Multi-Index Container for locally queueing ParamResponsePairs.
For a local queue, interface id’s are expected to be consistent, such that evaluation id’s are sufficient for tracking particular evaluations.

12.1.3 Enumeration Type Documentation
anonymous enum
Sub-methods, including sampling, inference algorithm, opt algorithm types.

Enumerator

SUBMETHOD_COLLABORATIVE  Type of hybrid meta-iterator:
12.1.4 Function Documentation

**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.
  References CommandShell::flush().
  Referenced by SysCallApplicInterface::spwan_analysis_to_shell(), SysCallApplicInterface::spwan_evaluation_to_shell(), SysCallApplicInterface::spwan_input_filter_to_shell(), and SysCallApplicInterface::spwan_output_filter_to_shell().

**void start_dakota_heartbeat** ( int )

Heartbeat function provided by dakota_filesystem_utils; pass output interval in seconds, or -1 to use $DAKOTA_HEARTBEAT
  Referenced by OutputManager::OutputManager().

**int Dakota::my_cp** ( const char ∗ file, const struct stat ∗ sb, int ftype, int depth, void ∗ v )

my_cp is a wrapper around 'cp -r'. The extra layer allows for symlink to be used instead of file copy.

**void Dakota::get_npath** ( int appdrive, std::string ∗ pnpath )

get_npath ”shuffles” the string representing the current $PATH variable definition so that ’.’ is first in the $PATH. It then returns the new string as the result (last arg in the call).
  References get_cwd().

**void register_signal_handlers** ( )

Tie various signal handlers to Dakota’s abort_handler function.
  Global function to register signal handlers at top-level.
  References abort_handler().
  Referenced by main().

**void mpi_debug_hold** ( )

Global function to hold Dakota processes to help with MPI debugging.
  See details in code for details, depending on MPI implementation in use.
  Referenced by main().

**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); }
  References abort_handler().

**bool Dakota::operator!=( const ActiveSet & set1, const ActiveSet & set2 ) [inline]**

inequality operator for ActiveSet
  inequality operator
bool Dakota::operator==(const Model &m1, const Model &m2) [inline]
equality operator for Envelope is true if same letter instance
equality operator (detect same letter instance)
References Model::modelRep.

bool Dakota::operator!=(const Model &m1, const Model &m2) [inline]
inequality operator for Envelope is true if different letter instance
equality operator (detect different letter instances)
References Model::modelRep.

bool Dakota::operator==(const Response &resp1, const Response &resp2) [inline]
equality operator for Response
equality operator

bool Dakota::operator!=(const Response &resp1, const Response &resp2) [inline]
inequality operator for Response
equality operator

bool Dakota::operator!=(const Variables &vars1, const Variables &vars2) [inline]
inequality operator for Variables
strict inequality operator

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

Real Dakota::mindist(const RealVector &x, const RealMatrix &xset, int except)
Returns the minimum distance between the point x and the points in the set xset (compares against all points in xset except point "except"); if except is not needed, pass 0.
References getdist().
References by getRmax(), and NonDAdaptImpSampling::select_rep_points().

Real Dakota::mindistindx(const RealVector &x, const RealMatrix &xset, const IntArray &indx)
 gets the min distance between x and points in the set xset defined by the nindx values in indx.
References getdist().
References by GaussProcApproximation::pointsel_add Sel().

Real Dakota::getRmax(const RealMatrix &xset)
Gets the maximum of the min distance between each point and the rest of the set.
References mindist().
References by GaussProcApproximation::pointsel_add Sel().


```
int Dakota::start_grid_computing ( char * analysis_driver_script, char * params_file, char * results_file )
sample function prototype for launching grid computing

int Dakota::stop_grid_computing ( )
sample function prototype for terminating grid computing

int Dakota::perform_analysis ( char * iteration_num )
sample function prototype for submitting a grid evaluation

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.

Referenced by JEGAOptimizer::LoadTheConstraints().

bool Dakota::operator== ( const ParamResponsePair & pair1, const ParamResponsePair & pair2 )
[inline]
equality operator for ParamResponsePair
equality operator
References ParamResponsePair::evalInterfaceIds, ParamResponsePair::prPairParameters, and ParamResponsePair::prPairResponse.

bool Dakota::operator!= ( const ParamResponsePair & pair1, const ParamResponsePair & pair2 )
[inline]
inequality operator for ParamResponsePair
equality operator

bool Dakota::set_compare ( const ParamResponsePair & database_pr, const ActiveSet & search_set )
[inline]
search function for a particular ParamResponsePair within a PRPList based on ActiveSet content (request vector and derivative variables vector)
a global function to compare the ActiveSet of a particular database_pr (presumed to be in the global history list) with a passed in ActiveSet (search_set).
References ParamResponsePair::active_set(), ActiveSet::derivative_vector(), and ActiveSet::request_vector().
Referenced by lookup_by_val().
```
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

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

References by partial_prp_equality::operator()().

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

find a ParamResponsePair based on the interface id, variables, and ActiveSet search data within search_pr.

Lookup occurs in two steps: (1) PRPMultiIndexCache lookup based on strict equality in interface id and

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

Referenced by ApplicationInterface::duplication_detect(), Model::estimate_derivatives(), SurrBasedLocalMinimizer::find.center_approx(), Optimizer::local_objective_recast_reduce(), lookup_by_val(), SNLLLeastSq::post_run(), SurrBasedMinimizer::print_results(), LeastSq::print_results(), Optimizer::print_results(), DiscrepancyCorrection::search_db(), and NonDLocalReliability::update_mpp_search_data().

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

find a ParamResponsePair based on the interface id, variables, and ActiveSet search data within search_pr.

Lookup occurs in two steps: (1) PRPMultiIndexQueue lookup based on strict equality in interface id and

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

Referenced by ApplicationInterface::duplication_detect(), Model::estimate_derivatives(), SurrBasedLocalMinimizer::find.center_approx(), Optimizer::local_objective_recast_reduce(), lookup_by_val(), SNLLLeastSq::post_run(), SurrBasedMinimizer::print_results(), LeastSq::print_results(), Optimizer::print_results(), DiscrepancyCorrection::search_db(), and NonDLocalReliability::update_mpp_search_data().

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

print a restart file

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

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

References abort_handler(), ParamResponsePair::eval_id(), ParamResponsePair::write_annotated(), and write_precision.

Referenced by main().

void print_restart_pdb ( int argc, char ** argv, String print_dest )

print a restart file (PDB format)

Usage: "dakota_restart_util to_pdb dakota.rst dakota.pdb"

Unrolls all data associated with a particular tag for all evaluations and then writes this data in a tabular format
(e.g., to a PDB database or MATLAB/TECPLLOT data file).

References abort_handler(), Variables::continuous_variables(), Variables::discrete_int_variables(), Variables::discrete_real_variables(), and Response::function_values().

Referenced by main().
12.1. DAKOTA NAMESPACE REFERENCE

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

print a restart file (tabular format)

Usage: "dakota_restart_util to_tabular dakota.rst dakota.txt"

Unrolls all data associated with a particular tag for all evaluations and then writes this data in a tabular format
(e.g., to a PDB database or MATLAB/TECPLT data file).

References abort_handler(), Variables::all_continuous_variable_labels(), Variables::all_discrete_int_variable_labels(), Variables::all_discrete_real_variable_labels(), Response::function_labels(), ParamResponsePair::interface_id(), ParamResponsePair::prp_parameters(), ParamResponsePair::prp_response(), and ParamResponsePair::write_tabular().

Referenced by main().

void read_neutral ( int argc, char ** argv )

read a restart file (neutral file format)

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

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

References ParamResponsePair::read_annotated().

Referenced by main().

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

repair a restart file by removing corrupted evaluations

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

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

References abort_handler(), Response::active_set_request_vector(), contains(), ParamResponsePair::eval_id(),
Response::function_values(), and ParamResponsePair::prp_response().

Referenced by main().

void concatenate_restart ( int argc, char ** argv )

concatenate multiple restart files

Usage: "dakota_restart_util cat dakota_1.rst ... dakota_n.rst dakota_new.rst"

Combines multiple restart files into a single restart database.

References abort_handler().

Referenced by main().

12.1.5 Variable Documentation

char slmap[256] [static]

Initial value:

= {

0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8, 0x9, 0xa, 0xb, 0xc, 0xd, 0xe, 0xf,
0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f,
0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29, 0x2a, 0x2b, 0x2c, 0x2d, 0x2e, 0x2f,
0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x3a, 0x3b, 0x3c, 0x3d, 0x3e, 0x3f,
0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, 0x4a, 0x4b, 0x4c, 0x4d, 0x4e, 0x4f,
Dakota_funcs DakFuncs0

Initial value:

```c
= {
    fprintf,
    abort_handler,
    dlsolver_option,
    continuous_lower_bounds1,
    continuous_upper_bounds1,
    nonlinear_ineq_constraint_lower_bounds1,
    nonlinear_ineq_constraint_upper_bounds1,
    nonlinear_eq_constraint_targets1,
    linear_ineq_constraint_lower_bounds1,
    linear_ineq_constraint_upper_bounds1,
    linear_eq_constraint_targets1,
    linear_eq_constraint_coeffs1,
    ComputeResponses1,
    GetFuncs1,
    GetGrads1,
    GetContVars1,
    SetBestContVars1,
    SetBestDiscVars1,
    SetBestRespFns1,
    Get_Real1,
    Get_Int1,
    Get_Bool1
}
```

```
const char∗ FIELD NAMES[] =

Initial value:

```c
= { "numFns", "numVars", "numACV", "numADIV", "numADRV", "numDerivVars", "xC", "xDI", "xDR", "xLabels", "xDILabels", "xDRLabels", "directFnASV", "directFnDVV", "fnFlag", "gradFlag", "hessFlag", "fnVals", "fnGrads", "fnHessians", "fnLabels", "failure", "currEvalId" }
```

fields to pass to Matlab in Dakota structure

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

```
const int NUMBER_OF_FIELDS = 23
```

number of fields in above structure

Referenced by MatlabInterface::matlab_engine_run(), and MatlabInterface::MatlabInterface().
GuiKeyWord kw_1[1]  [static]
Initial value:

= {
   {"results_output_file",1,0,1,0,13,0,0,0,0,0,0,0,"{File name for results output}
    EnvCommands.html#EnvOutput"}
}

1231 distinct keywords (plus 180 aliases)

GuiKeyWord kw_2[1]  [static]
Initial value:

= {
   {"tabular_graphics_file",1,0,1,0,7,0,0,0,0,0,0,"{File name for tabular graphics data}
    EnvCommands.html#EnvOutput"}
}

GuiKeyWord kw_3[6]  [static]
Initial value:

= {
   {"graphics",1,0,1,0,3,0,0,0,0,0,0,"{Graphics flag}
    EnvCommands.html#EnvOutput"},
   {"method_pointer",3,0,5,0,14},
   {"output_precision",0,29,0,3,0,0,0,0,"{Numeric output precision}
    EnvCommands.html#EnvOutput"},
   {"results_output",8,1,4,0,11,kw_1,0,0,0,0,0,"{Enable results output}
    EnvCommands.html#EnvOutput"},
   {"tabular_graphics_data",8,1,2,0,5,kw_2,0,0,0,0,0,"{Tabulation of graphics data}
    EnvCommands.html#EnvOutput"},
   {"top_method_pointer",11,0,5,0,15,0,0,0,0,0,0,"{Method pointer}
    EnvCommands.html#EnvMethPtr"}
}

GuiKeyWord kw_4[1]  [static]
Initial value:

= {
   {"cache_tolerance",10,0,1,0,2263}
}

GuiKeyWord kw_5[4]  [static]
Initial value:

= {
   {"active_set_vector",8,0,1,0,2257},
   {"evaluation_cache",8,0,2,0,2259},
   {"restart_file",8,0,4,0,2263},
   {"strict_cache_equality",8,1,3,0,2261,kw_4}
}
GuiKeyWord kw_6[1]  [static]
Initial value:

= {
   "processors_per_analysis", 0x19, 0, 1, 0, 2233, 0, 0., 0., 0., 0., "\{Number of processors per analysis server\} InterfCommands.html#InterfApplicDF"}
}

GuiKeyWord kw_7[4]  [static]
Initial value:

= {
   "abort", 8, 0, 1, 1, 2247, 0, 0., 0., 0., 0., 0., "\{CHOOSE failure mitigation\}"],
   "continuation", 8, 0, 1, 1, 2253],
   "recover", 14, 0, 1, 1, 2251],
   "retry", 9, 0, 1, 1, 2249]}
}

GuiKeyWord kw_8[1]  [static]
Initial value:

= {
   "numpy", 8, 0, 1, 0, 2239, 0, 0., 0., 0., 0., "\{Python NumPy dataflow\} InterfCommands.html#InterfApplicMSP"}
}

GuiKeyWord kw_9[2]  [static]
Initial value:

= {
   "copy", 8, 0, 1, 0, 2227, 0, 0., 0., 0., 0., 0., "\{Copy template files\} InterfCommands.html#InterfApplicF"
},
   "replace", 8, 0, 2, 0, 2229, 0, 0., 0., 0., 0., 0., "\{Replace existing files\} InterfCommands.html#InterfApplicF"
}

GuiKeyWord kw_10[7]  [static]
Initial value:

= {
   "dir_save", 0, 0, 3, 0, 2220],
   "dir_tag", 0, 0, 2, 0, 2218],
   "directory_save", 8, 0, 3, 0, 2221, 0, 0., 0., 0., 0., 0., "\{Save work directory\} InterfCommands.html#InterfApplicF"],
   "directory_tag", 8, 0, 2, 0, 2219, 0, 0., 0., 0., 0., 0., "\{Tag work directory\} InterfCommands.html#InterfApplicF"],
   "named", 11, 0, 1, 0, 2217, 0, 0., 0., 0., 0., 0., "\{Name of work directory\} InterfCommands.html#InterfApplicF"],
   "template_directory", 11, 2, 4, 0, 2223, 0, 0., 0., 0., 0., 0., "\{Template directory\} InterfCommands.html#InterfApplicF"],
   "template_files", 15, 2, 4, 0, 2225, 0, 0., 0., 0., 0., 0., "\{Template files\} InterfCommands.html#InterfApplicF"
}
}
GuiKeyWord kw_11[9]  [static]

Initial value:

```java
{ "allow_existing.results", 8, 0, 3, 0, 2205, 0, 0., 0., 0., 0., "[Allow existing results files]
InterfCommands.html#InterfApplicF" },
{ "aprepro", 8, 0, 5, 0, 2209, 0, 0., 0., 0., 0., "[Aprepro parameters file format]
InterfCommands.html#InterfApplicF" },
{ "dprepro", 0, 0, 5, 0, 2208 },
{ "file_name", 8, 0, 7, 0, 2215, 0, 0., 0., 0., 0., "[Parameters and results file saving]
InterfCommands.html#InterfApplicF" },
{ "file_tag", 8, 0, 6, 0, 2211, 0, 0., 0., 0., 0., "[Parameters and results file tagging]
InterfCommands.html#InterfApplicF" },
{ "parameters_file", 11, 0, 1, 0, 2201, 0, 0., 0., 0., 0., "[Parameters file name]
InterfCommands.html#InterfApplicF" },
{ "results_file", 11, 0, 2, 0, 2203, 0, 0., 0., 0., 0., "[Results file name]
InterfCommands.html#InterfApplicF" },
{ "verbatim", 8, 0, 4, 0, 2207, 0, 0., 0., 0., 0., "[Verbatim driver/filter invocation syntax]
InterfCommands.html#InterfApplicF" },
{ "work_directory", 8, 7, 8, 0, 2215, 0, 0., 0., 0., 0., "[Create work directory]
InterfCommands.html#InterfApplicF" }
```

GuiKeyWord kw_12[12]  [static]

Initial value:

```java
{ "analysis_components", 15, 0, 1, 0, 2191, 0, 0., 0., 0., 0., "[Additional identifiers for use by the analysis_drivers]
InterfCommands.html#InterfApplic" },
{ "deactivate", 8, 4, 6, 0, 2255, kw_5, 0, 0., 0., 0., "[Feature deactivation]
InterfCommands.html#InterfApplicF" },
{ "direct", 8, 1, 4, 1, 2231, kw_6, 0, 0., 0., 0., 0., "[CHOOSE interface type] [Direct function interface]
InterfCommands.html#InterfApplicDF" },
{ "failure_capture", 8, 4, 5, 0, 2245, kw_7, 0, 0., 0., 0., "[Failure capturing]
InterfCommands.html#InterfApplicF" },
{ "fork", 8, 9, 4, 1, 2199, kw_11, 0, 0., 0., 0., "[Fork interface]
InterfCommands.html#InterfApplicF" },
{ "grid", 8, 0, 4, 1, 2243, 0, 0., 0., 0., 0., "[Grid interface]
InterfCommands.html#InterfApplicF" },
{ "input_filter", 11, 0, 2, 0, 2193, 0, 0., 0., 0., "[Input filter]
InterfCommands.html#InterfApplicF" },
{ "matlab", 8, 0, 4, 1, 2235, 0, 0., 0., 0., 0., "[Matlab interface]
InterfCommands.html#InterfApplicMSP" },
{ "output_filter", 11, 0, 3, 0, 2195, 0, 0., 0., 0., 0., "[Output filter]
InterfCommands.html#InterfApplicF" },
{ "python", 8, 1, 4, 1, 2237, kw_8, 0, 0., 0., 0., "[Python interface]
InterfCommands.html#InterfApplicMSP" },
{ "scilab", 8, 0, 4, 1, 2241, 0, 0., 0., 0., 0., "[Scilab interface]
InterfCommands.html#InterfApplicMSP" },
{ "system", 8, 9, 4, 1, 2197, kw_11 } }
```

GuiKeyWord kw_13[2]  [static]

Initial value:

```java
{ "master", 8, 0, 1, 1, 2297 },
{ "peer", 8, 0, 1, 1, 2299 }
```
GuiKeyWord kw_14[2] [static]
Initial value:
= {
    {"dynamic",8,0,1,1,2273},
    {"static",8,0,1,1,2275}
}

GuiKeyWord kw_15[3] [static]
Initial value:
= {
    {"analysis_concurrency",0x19,0,3,0,2277,0,0,0,0,0,"Asynchronous analysis concurrency InterfCommands.html#InterfIndControl"},
    {"evaluation_concurrency",0x19,0,1,0,2269,0,0,0,0,0,"Asynchronous evaluation concurrency InterfCommands.html#InterfIndControl"},
    {"local_evaluation_scheduling",8,2,2,0,2271,kw_14,0,0,0,0,"Local evaluation scheduling InterfCommands.html#InterfIndControl"}
}

GuiKeyWord kw_16[2] [static]
Initial value:
= {
    {"dynamic",8,0,1,1,2287},
    {"static",8,0,1,1,2289}
}

GuiKeyWord kw_17[2] [static]
Initial value:
= {
    {"master",8,0,1,1,2283},
    {"peer",8,2,1,1,2285,kw_16,0,0,0,0,"Peer scheduling of evaluations InterfCommands.html#InterfIndControl"}
}

GuiKeyWord kw_18[9] [static]
Initial value:
= {
    {"algebraic_mappings",11,0,2,0,2187,0,0,0,0,0,"Algebraic mappings file InterfCommands.html#InterfAlgebraic"},
    {"analysis_drivers",15,12,3,0,2189,kw_12,0,0,0,0,"Analysis drivers InterfCommands.html#InterfApplic"},
    {"analysis_scheduling",8,2,9,0,2295,kw_13,0,0,0,0,"Message passing configuration for scheduling of analyses InterfCommands.html#InterfIndControl"},
    {"analysis_servers",0x19,0,8,0,2293,0,0,0,0,0,"Number of analysis servers InterfCommands.html#InterfIndControl"},
    {"asynchronous",8,3,4,0,2267,kw_15,0,0,0,0,0,"Asynchronous interface usage InterfCommands.html#InterfIndControl"},
    {"evaluation_scheduling",8,2,6,0,2281,kw_17,0,0,0,0,0,"Message passing configuration for scheduling of evaluations InterfCommands.html#InterfIndControl"},
    {"evaluation_servers",0x19,0,5,0,2279,0,0,0,0,0,"Number of evaluation servers InterfCommands.html#InterfIndControl"},
    {"id_interface",11,0,1,0,2185,0,0,0,0,0,"Interface set identifier InterfCommands.html#InterfIndControl"},
    {"processors_per_evaluation",0x19,0,7,0,2291,0,0,0,0,0,"Number of processors per evaluation server InterfCommands.html#InterfIndControl"}
}
GuiKeyWord kw_19[2]  [static]
Initial value:

= {
    
    "complementary",8,0,1,1,1131,
    "cumulative",8,0,1,1,1129
}

GuiKeyWord kw_20[1]  [static]
Initial value:

= {
    
    "num_gen_reliability_levels",13,0,1,0,1139,0,0..0,0."{Number of generalized
    reliability levels} MethodCommands.html#MethodNonD"
}

GuiKeyWord kw_21[1]  [static]
Initial value:

= {
    
    "num_probability_levels",13,0,1,0,1135,0,0..0,0."{Number of probability levels}
    MethodCommands.html#MethodNonD"
}

GuiKeyWord kw_22[2]  [static]
Initial value:

= {
    
    "mt19937",8,0,1,1,1143,
    "rnum2",8,0,1,1,1145
}

GuiKeyWord kw_23[4]  [static]
Initial value:

= {
    
    "constant_liar",8,0,1,1,1023,
    "distance_penalty",8,0,1,1,1019,
    "naive",8,0,1,1,1017],
    "topology",8,0,1,1,1021
}

GuiKeyWord kw_24[2]  [static]
Initial value:

= {
    
    "annotated",8,0,1,0,1035,
    "freeform",8,0,1,0,1037
}
GuiKeyWord kw_25[3] [static]
Initial value:
= {
  "distance",8,0,1,1,1011,
  "gradient",8,0,1,1,1013,
  "predicted_variance",8,0,1,1,1009
}

GuiKeyWord kw_26[2] [static]
Initial value:
= {
  "annotated",8,0,1,0,1029,
  "freeform",8,0,1,0,1031
}

GuiKeyWord kw_27[2] [static]
Initial value:
= {
  "parallel",8,0,1,1,1053,
  "series",8,0,1,1,1051
}

GuiKeyWord kw_28[3] [static]
Initial value:
= {
  "gen_reliabilities",8,0,1,1,1047,
  "probabilities",8,0,1,1,1045,
  "system",8,2,2,0,1049,kw_27
}

GuiKeyWord kw_29[2] [static]
Initial value:
= {
  "compute",8,3,2,0,1043,kw_28,
  "num_response_levels",13,0,1,0,1041
}

GuiKeyWord kw_30[15] [static]
Initial value:
= {
  "batch_selection",8,4,3,0,1015,kw_23,0.,0.,0.,0,"{Batch selection strategy}
MethodCommands.html#MethodNonDAdaptive"},
  "batch_size",9,0,4,0,1025,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,"{Batch size (number of points added each
iteration)} MethodCommands.html#MethodNonDAdaptive"),
  "distribution",8,2,12,0,1127,kw_19,0.,0.,0.,0.,"{Distribution type}
MethodCommands.html#MethodNonD"),
  "emulator_samples",9,0,1,0,1005,0.,0.,0.,0.,0.,0.,"{Number of samples on the emulator to
generate a new true sample each iteration); MethodCommands.html#MethodNonDAdaptive*

  *export_points_file*, 11, 2, 6, 0, 1033, kw, 24, 0, 0, 0, 0, 0, "File name for exporting
approximation-based samples from evaluating the GP
MethodCommands.html#MethodNonDAdaptive*

  *fitness_metric*, 8, 3, 2, 0, 1007, kw, 25, 0, 0, 0, 0, "Fitness metric
MethodCommands.html#MethodNonDAdaptive*

*import_points_file*, 11, 2, 5, 0, 1027, kw, 26, 0, 0, 0, 0, 0, "File name for points to be imported
as the basis for the initial GP
MethodCommands.html#MethodNonDAdaptive*

  *misc_options*, 15, 0, 8, 0, 1055

  *model_pointer*, 11, 0, 9, 0, 1033

  *probability_levels*, 14, 1, 13, 0, 1133, kw, 21, 0, 0, 0, 0, 0, "Probability levels
MethodCommands.html#MethodNonD"

  *rng*, 8, 2, 15, 0, 1141, kw, 22, 0, 0, 0, 0, "Random number generator
MethodCommands.html#MethodNonDMC*

  *samples*, 9, 0, 10, 0, 1361, 0, 0, 0, 0, 0, 0, "Number of samples
MethodCommands.html#MethodNonDMC"

  *seed*, 0x19, 0, 11, 0, 1363, 0, 0, 0, 0, 0, 0, "Refinement seed
MethodCommands.html#MethodNonDLocalRel*

GuiKeyWord kw_31[7] [static]

Initial value:

= {
  *merit1*, 8, 0, 1, 1, 305, 0, 0, 0, 0, 0, 0, [CHOOSE merit function]*
  *merit1_smooth*, 8, 0, 1, 1, 307
  *merit2*, 8, 0, 1, 1, 309
  *merit2_smooth*, 8, 0, 1, 1, 311, 0, 0, 0, 0, 0, 0, *@*
  *merit2_squared*, 8, 0, 1, 1, 313
  *merit_max*, 8, 0, 1, 1, 301
  *merit_max_smooth*, 8, 0, 1, 1, 303
}

GuiKeyWord kw_32[2] [static]

Initial value:

= {
  *blocking*, 8, 0, 1, 1, 295, 0, 0, 0, 0, 0, 0, [CHOOSE synchronization]*
  *nonblocking*, 8, 0, 1, 1, 297, 0, 0, 0, 0, 0, 0, *@*
}

GuiKeyWord kw_33[19] [static]

Initial value:

= {
  *constraint_penalty*, 10, 0, 7, 0, 315, 0, 0, 0, 0, 0, 0, [Constraint penalty
MethodCommands.html#MethodAPPSDC*]
  *contraction_factor*, 10, 0, 2, 0, 287, 0, 0, 0, 0, 0, 0, [Pattern contraction factor
MethodCommands.html#MethodAPPSDC*]
  *initial_delta*, 10, 0, 1, 0, 285, 0, 0, 0, 0, 0, 0, [Initial offset value
MethodCommands.html#MethodAPPSDC*]
  *linear_inequality_constraint_matrix*, 14, 0, 15, 0, 465, 0, 0, 0, 0, 0, 0, [Linear equality
coefficient matrix
MethodCommands.html#MethodMin*]
  *linear_inequality_constraint_matrix*, 14, 0, 455, 0, 0, 0, 0, 0, 0, 0, 0, 0, [Linear equality
coefficient matrix
MethodCommands.html#MethodMin*]
  *linear_inequality_target_matrix*, 14, 0, 456, 0, 0, 0, 0, 0, 0, 0, 0, 0, [Linear equality targets
MethodCommands.html#MethodMin*]
  *linear_inequality_constraint_matrix*, 14, 0, 455, 0, 0, 0, 0, 0, 0, 0, 0, [Linear inequality
coefficient matrix} MethodCommands.html#MethodMin", 
{"linear.inequality.lower.bounds",14,0,11,0,457,0,0,0,0,0,0,0,0,"{Linear inequality lower bounds}" MethodCommands.html#MethodMin"}, 
{"linear.inequality.scale.types",15,0,13,0,461,0,0,0,0,0,0,0,0,"{Linear inequality scaling types}" MethodCommands.html#MethodMin"}, 
{"linear.inequality.scales",14,0,14,0,463,0,0,0,0,0,0,0,0,"{Linear inequality scales}" MethodCommands.html#MethodMin"}, 
{"linear.inequality.upper.bounds",14,0,12,0,459,0,0,0,0,0,0,0,0,"{Linear inequality upper bounds}" MethodCommands.html#MethodMin"}, 
{"merit.function",8,7,6,0,289,0,0,0,0,0,0,0,0,"{Merit function}" MethodCommands.html#MethodAPPSDC"}, 
{"model_pointer",11,0,9,0,1593}, 
{"smoothing_factor",10,0,8,0,317,0,0,0,0,0,0,0,0,"{Smoothing factor}" MethodCommands.html#MethodAPPSDC"}, 
{"solution_accuracy",2,0,4,0,290}, 
{"solution_target",10,0,4,0,291,0,0,0,0,0,0,0,0,"{Solution target}" MethodCommands.html#MethodAPPSDC"}, 
{"synchronization",8,2,5,0,293,0,0,0,0,0,0,0,0,"{Evaluation synchronization}" MethodCommands.html#MethodAPPSDC"}, 
{"threshold_delta",10,0,3,0,289,0,0,0,0,0,0,0,0,"{Threshold for offset values}" MethodCommands.html#MethodAPPSDC"} 
} )

GuiKeyWord kw_34[2] [static]
Initial value:

= {
  "annotated",8,0,1,0,1289],
  "freeform",8,0,1,0,1291]
}

GuiKeyWord kw_35[2] [static]
Initial value:

= {
  "annotated",8,0,1,0,1289],
  "freeform",8,0,1,0,1285]
}

GuiKeyWord kw_36[5] [static]
Initial value:

= {
  "dakota",8,0,1,1,1277},
  "emulator_samples",9,0,2,0,1279},
  "export_points_file",11,2,4,0,1287,kw_34},
  "import_points_file",11,2,3,0,1281,kw_35},
  "surfpack",8,0,1,1,1275}
}

GuiKeyWord kw_37[1] [static]
Initial value:

= {
  "sparse_grid_level",13,0,1,0,1295}
GuiKeyWord kw_38[1]  [static]
Initial value:

= {
    {"sparse_grid_level",13,0,1,0,1299}
}

GuiKeyWord kw_39[4]  [static]
Initial value:

= {
    {"gaussian_process",8,1,1,1,1273,kw_36},
    {"kriging",0,5,1,1,1272,kw_36},
    {"pce",8,1,1,1,1293,kw_37},
    {"sc",8,1,1,1,1297,kw_38}
}

GuiKeyWord kw_40[6]  [static]
Initial value:

= {
    {"chains",0x29,0,1,0,1261,0,3,0,0,0,0,"{Number of chains} MethodCommands.html#MethodNonDBayesCalib"},
    {"crossover_chain_pairs",0x29,0,3,0,1265,0,0,0,0,0,0,"{Number of chain pairs used in crossover} MethodCommands.html#MethodNonDBayesCalib"},
    {"emulator",8,4,6,0,1271,kw_39},
    {"gr_threshold",0x1a,0,4,0,1267,0,0,0,0,0,0,"{Gelman-Rubin Threshold for convergence} MethodCommands.html#MethodNonDBayesCalib"},
    {"jump_step",0x29,0,5,0,1269,0,0,0,0,0,0,"{Jump-Step} MethodCommands.html#MethodNonDBayesCalib"},
    {"num_cr",0x29,0,2,0,1263,0,1,0,0,0,0,"{Number of candidate points used in burn-in adaptation} MethodCommands.html#MethodNonDBayesCalib"}
}

GuiKeyWord kw_41[2]  [static]
Initial value:

= {
    {"adaptive",8,0,1,1,1247},
    {"hastings",8,0,1,1,1245}
}

GuiKeyWord kw_42[2]  [static]
Initial value:

= {
    {"delayed",8,0,1,1,1241},
    {"standard",8,0,1,1,1239}
}
GuiKeyWord kw_43[2]  [static]
Initial value:
= {
    "metropolis",8,2,2,0,1243,kw_41,0,0,0,0,"{Metropolis type for the MCMC algorithm } MethodCommands.html#MethodNonDBayesCalib",
    "rejection",8,2,1,0,1237,kw_42
}

GuiKeyWord kw_44[2]  [static]
Initial value:
= {
  "dram",8,2,1,1235,kw_43,
  "multilevel",8,0,1,1,1249
}

GuiKeyWord kw_45[2]  [static]
Initial value:
= {
  "mt19937",8,0,1,1,1253,
  "rnum2",8,0,1,1,1255
}

GuiKeyWord kw_46[2]  [static]
Initial value:
= {
  "annotated",8,0,1,0,1229,
  "freeform",8,0,1,0,1231
}

GuiKeyWord kw_47[2]  [static]
Initial value:
= {
  "annotated",8,0,1,0,1223,
  "freeform",8,0,1,0,1225
}

GuiKeyWord kw_48[6]  [static]
Initial value:
= {
  "emulator_samples",9,0,1,1,1219,
  "export_points_file",11,2,3,0,1227,kw_46,
  "import_points_file",11,2,2,0,1221,kw_47,
  "mcmc_type",8,2,4,0,1233,kw_44,
  "proposal_covariance_scale",14,0,6,0,1257,0,0,0,0,"{Proposal covariance scaling} MethodCommands.html#MethodNonDBayesCalib",
  "rng",8,2,5,0,1251,kw_45,0,0,0,0,"{Random seed generator} MethodCommands.html#MethodNonDBayesCalib"}
GuiKeyWord kw_49[2] [static]
Initial value:
= {
    {"annotated", 8, 0, 1, 0, 1205},
    {"freeform", 8, 0, 1, 0, 1207}
}

GuiKeyWord kw_50[2] [static]
Initial value:
= {
    {"annotated", 8, 0, 1, 0, 1199},
    {"freeform", 8, 0, 1, 0, 1201}
}

GuiKeyWord kw_51[5] [static]
Initial value:
= {
    {"dakota", 8, 0, 1, 1, 1193},
    {"emulator_samples", 9, 0, 2, 0, 1195},
    {"export_points_file", 11, 2, 4, 0, 1203, kw_49},
    {"import_points_file", 11, 2, 3, 0, 1197, kw_50},
    {"surfpack", 8, 0, 1, 1, 1191}
}

GuiKeyWord kw_52[1] [static]
Initial value:
= {
    {"sparse_grid_level", 13, 0, 1, 0, 1211}
}

GuiKeyWord kw_53[1] [static]
Initial value:
= {
    {"sparse_grid_level", 13, 0, 1, 0, 1215}
}

GuiKeyWord kw_54[4] [static]
Initial value:
= {
    {"gaussian_process", 8, 5, 1, 1, 1189, kw_51},
    {"kriging", 0, 5, 1, 1, 1188, kw_51},
    {"pce", 8, 1, 1, 1, 1209, kw_52},
    {"sc", 8, 1, 1, 1, 1213, kw_53}
}
GuiKeyword kw_55[4] [static]

Initial value:

```c
{ "emulator", 8, 4, 1, 0, 1187, kw_54 },
{ "mcmc_type", 8, 2, 0, 1233, kw_44 },
{ "proposal_covariance_scale", 14, 0, 4, 0, 1257, 0, 0, 0, 0, 0, 0, "Proposal covariance scaling" MethodCommands.html#MethodNonDBayesCalib" },
{ "rng", 8, 2, 3, 0, 1251, kw_45, 0, 0, 0, 0, 0, 0, "Random seed generator" MethodCommands.html#MethodNonDBayesCalib" }
```

GuiKeyword kw_56[9] [static]

Initial value:

```c
{ "calibrate_sigma", 8, 0, 4, 0, 1305, 0, 0, 0, 0, 0, 0, "Calibrate sigma flag" MethodCommands.html#MethodNonDBayesCalib" },
{ "dream", 8, 6, 1, 1259, kw_40 },
{ "gmmss", 8, 6, 1, 1217, kw_44 },
{ "likelihood_scale", 10, 0, 3, 0, 1303, 0, 0, 0, 0, 0, 0, "Likelihood scale factor" MethodCommands.html#MethodNonDBayesCalib" },
{ "model_pointer", 11, 0, 5, 0, 1593 },
{ "queso", 8, 4, 1, 1185, kw_55 },
{ "samples", 9, 0, 6, 0, 1361, 0, 0, 0, 0, 0, 0, "Number of samples" MethodCommands.html#MethodNonCMC" }
```

GuiKeyword kw_57[4] [static]

Initial value:

```c
{ "deltas_per_variable", 5, 0, 2, 2, 1576 },
{ "model_pointer", 11, 0, 3, 0, 1593 },
{ "step_vector", 14, 0, 1, 1, 1575, 0, 0, 0, 0, 0, 0, "Step vector" MethodCommands.html#MethodPSCPS" },
{ "steps_per_variable", 13, 0, 2, 2, 1577, 0, 0, 0, 0, 0, 0, "Number of steps per variable" MethodCommands.html#MethodPSCPS" }
```

GuiKeyword kw_58[7] [static]

Initial value:

```c
{ "beta_solver_name", 11, 0, 1, 1, 603 },
{ "misc_options", 15, 0, 6, 0, 611, 0, 0, 0, 0, 0, 0, "Specify miscellaneous options" MethodCommands.html#MethodSCOLIBDC" },
{ "model_pointer", 11, 0, 2, 0, 1593 },
{ "seed", 0x19, 0, 4, 0, 607, 0, 0, 0, 0, 0, 0, "Random seed for stochastic pattern search" MethodCommands.html#MethodSCOLIBPS" },
{ "show_misc_options", 8, 0, 5, 0, 609, 0, 0, 0, 0, 0, 0, "Show miscellaneous options" MethodCommands.html#MethodSCOLIBDC" },
{ "solution_accuracy", 2, 0, 3, 0, 604 },
{ "solution_target", 10, 0, 3, 0, 605, 0, 0, 0, 0, 0, 0, "Desired solution target" MethodCommands.html#MethodSCOLIBDC" }
```
GuiKeyWord kw_59[8] [static]
Initial value:
= {
    {"initial_delta",10,0,6,0,521,0,0,..,0,0,0,0,*{Initial offset value
MethodCommands.html#MethodSCOLIBPS"},
    {"misc_options",15,0,5,0,611,0,0,..,0,0,0,0,*{Specify miscellaneous options
MethodCommands.html#MethodSCOLIBDC"},
    {"model_pointer",11,0,0,1,0,1593},
    {"seed",0x19,0,3,0,607,0,0,..,0,0,0,0,*{Random seed for stochastic pattern search
MethodCommands.html#MethodSCOLIBPS"},
    {"show_misc_options",8,0,4,0,609,0,0,..,0,0,0,0,*{Show miscellaneous options
MethodCommands.html#MethodSCOLIBDC"},
    {"solution_accuracy",2,0,2,0,604},
    {"solution_target",10,0,2,0,605,0,0,..,0,0,0,0,*{Desired solution target
MethodCommands.html#MethodSCOLIBDC"},
    {"threshold_delta",10,0,7,0,523,0,0,..,0,0,0,0,*{Threshold for offset values
MethodCommands.html#MethodSCOLIBPS"}
}
}

GuiKeyWord kw_60[2] [static]
Initial value:
= {
    {"all_dimensions",8,0,1,1,531},
    {"major_dimension",8,0,1,1,539}
}

GuiKeyWord kw_61[12] [static]
Initial value:
= {
    {"constraint_penalty",10,0,6,0,541,0,0,..,0,0,0,0,*{Constraint penalty
MethodCommands.html#MethodSCOLIBDIR"},
    {"division",8,2,1,0,527,kw_60,0,0,..,0,0,0,0,*{Box subdivision approach
MethodCommands.html#MethodSCOLIBDIR"},
    {"global_balanceParameter",10,0,2,0,533,0,0,..,0,0,0,0,*{Global search balancing parameter
MethodCommands.html#MethodSCOLIBDIR"},
    {"local_balance_parameter",10,0,3,0,535,0,0,..,0,0,0,0,*{Local search balancing parameter
MethodCommands.html#MethodSCOLIBDIR"},
    {"max_boxsize_limit",10,0,4,0,537,0,0,..,0,0,0,0,*{Maximum boxsize limit
MethodCommands.html#MethodSCOLIBDIR"},
    {"min_boxsize_limit",10,0,5,0,539,0,0,..,0,0,0,0,*{Minimum boxsize limit
MethodCommands.html#MethodSCOLIBDIR"},
    {"misc_options",15,0,11,0,611,0,0,..,0,0,0,0,*{Specify miscellaneous options
MethodCommands.html#MethodSCOLIBDC"},
    {"model_pointer",11,0,7,0,1593},
    {"seed",0x19,0,5,0,607,0,0,..,0,0,0,0,*{Random seed for stochastic pattern search
MethodCommands.html#MethodSCOLIBPS"},
    {"show_misc_options",8,0,10,0,609,0,0,..,0,0,0,0,*{Show miscellaneous options
MethodCommands.html#MethodSCOLIBDC"},
    {"solution_accuracy",2,0,8,0,604},
    {"solution_target",10,0,7,0,605,0,0,..,0,0,0,0,*{Desired solution target
MethodCommands.html#MethodSCOLIBDC"}
}
}

GuiKeyWord kw_62[3] [static]
Initial value:
= {
    {"blend",8,0,1,1,577},
    {"two_point",8,0,1,1,575},
    {"uniform",8,0,1,1,579}
}
GuiKeyWord kw_63[2] [static]
Initial value:

```csharp
= {
    {"linear_rank",8,0,1,1,557},
    {"merit_function",8,0,1,1,559}
}
```

GuiKeyWord kw_64[3] [static]
Initial value:

```csharp
= {
    {"flat_file",11,0,1,1,553},
    {"simple_random",8,0,1,1,549},
    {"unique_random",8,0,1,1,551}
}
```

GuiKeyWord kw_65[2] [static]
Initial value:

```csharp
= {
    {"mutation_range",9,0,2,0,595,0,0,0,0,0,0,"{Mutation range}
MethodCommands.html#MethodSCOLIBEA"},
    {"mutation_scale",10,0,1,0,593,0,0,0,0,0,"{Mutation scale}
MethodCommands.html#MethodSCOLIBEA"}
}
```

GuiKeyWord kw_66[5] [static]
Initial value:

```csharp
= {
    {"nonadaptive",8,0,2,0,597,0,0,0,0,0,0,"{Non-adaptive mutation flag}
MethodCommands.html#MethodSCOLIBEA"},
    {"offset_cauchy",8,2,1,1,589,kw_65},
    {"offset_normal",8,2,1,1,587,kw_65},
    {"offset_uniform",8,2,1,1,591,kw_65},
    {"replace_uniform",8,0,1,1,585}
}
```

GuiKeyWord kw_67[4] [static]
Initial value:

```csharp
= {
    {"chc",9,0,1,1,565,0,0,0,0,0,"{CHC replacement type} MethodCommands.html#MethodSCOLIBEA"},
    {"elitist",9,0,1,1,567,0,0,0,0,0,"{Elitist replacement type}
MethodCommands.html#MethodSCOLIBEA"},
    {"new_solutions_generated",9,0,2,0,569,0,0,0,0,0,"{New solutions generated}
MethodCommands.html#MethodSCOLIBEA"},
    {"random",9,0,1,1,563,0,0,0,0,0,"{Random replacement type}
MethodCommands.html#MethodSCOLIBEA"}
}
```
GuiKeyWord kw_68[15] [static]

Initial value:

= {
    {"constraint_penalty",10,0,9,0,599},
    {"crossover_rate",10,0,5,0,571,0,0,0,0,0,"{Crossover rate
MethodCommands.html#MethodSCOLIBEA"},
    {"crossover_type",8,3,6,0,573,kw_62,0,0,0,0,"{Crossover type
MethodCommands.html#MethodSCOLIBEA"},
    {"fitness_type",8,2,3,0,555,kw_63,0,0,0,0,"{Fitness type
MethodCommands.html#MethodSCOLIBEA"},
    {"initialization_type",8,3,2,0,547,kw_64,0,0,0,0,"{Initialization type
MethodCommands.html#MethodSCOLIBEA"},
    {"misc_options",15,0,14,0,611,0,0,0,0,0,"{Specify miscellaneous options
MethodCommands.html#MethodSCOLIBDC"},
    {"model_pointer",11,0,10,0,1593},
    {"mutation_rate",10,0,7,0,581,0,0,0,0,0,"{Mutation rate
MethodCommands.html#MethodSCOLIBEA"},
    {"mutation_type",8,5,0,0,583,kw_66,0,0,0,0,0,"{Mutation type
MethodCommands.html#MethodSCOLIBEA"},
    {"population_size",0x19,0,1,0,545,0,0,0,0,0,"{Number of population members
MethodCommands.html#MethodSCOLIBEA"},
    {"replacement_type",8,4,4,0,561,kw_67,0,0,0,0,0,"{Replacement type
MethodCommands.html#MethodSCOLIBEA"},
    {"seed",0x19,0,12,0,607,0,0,0,0,1,"{Random seed for stochastic pattern search
MethodCommands.html#MethodSCOLIBEA"},
    {"show_misc_options",8,0,13,0,609,0,0,0,0,0,"{Show miscellaneous options
MethodCommands.html#MethodSCOLIBDC"},
    {"solution_accuracy",2,0,11,0,604},
    {"solution_target",10,0,11,0,605,0,0,0,0,0,"{Desired solution target
MethodCommands.html#MethodSCOLIBDC"}
}

GuiKeyWord kw_69[3] [static]

Initial value:

= {
    {"adaptive_pattern",8,0,1,1,495},
    {"basic_pattern",8,0,1,1,497},
    {"multi_step",8,0,1,1,493}
}

GuiKeyWord kw_70[2] [static]

Initial value:

= {
    {"coordinate",8,0,1,1,483},
    {"simplex",8,0,1,1,485}
}

GuiKeyWord kw_71[2] [static]

Initial value:

= {
    {"blocking",8,0,1,1,501},
    {"nonblocking",8,0,1,1,503}
}
GuiKeyWord kw_72[18] [static]

Initial value:

```c
= {
    "constant_penalty", 8,0,1,0,475,0,0,0,0,0,0,"[Control of dynamic penalty]
MethodCommands.html#MethodSCOLIBPS",
    "constraint_penalty", 10,0,17,0,517,0,0,0,0,0,0,"[Constraint penalty]
MethodCommands.html#MethodSCOLIBPS",
    "contraction_factor", 10,0,16,0,515,0,0,0,0,0,0,"[Pattern contraction factor]
MethodCommands.html#MethodSCOLIBPS",
    "expand_after_success", 9,0,3,0,479,0,0,0,0,0,0,"[Number of consecutive improvements
before expansion] MethodCommands.html#MethodSCOLIBPS",
    "initial_delta", 10,0,14,0,521,0,0,0,0,0,0,"[Initial offset value]
MethodCommands.html#MethodSCOLIBPS",
    "misc_options", 15,0,13,0,611,0,0,0,0,0,0,"[Specify miscellaneous options]
MethodCommands.html#MethodSCOLIBDC",
    "model_pointer", 11,0,9,0,1593,0,0,0,0,0,0,"[Initial offset value]
MethodCommands.html#MethodSCOLIBPS",
    "no_expansion", 8,0,2,0,477,0,0,0,0,0,0,"[No expansion flag]
MethodCommands.html#MethodSCOLIBPS",
    "pattern_basis", 8,2,4,0,481,0,0,0,0,0,0,"[Pattern basis selection]
MethodCommands.html#MethodSCOLIBPS",
    "seed", 0x19,0,11,0,607,0,0,0,0,0,0,"[Random seed for stochastic pattern search]
MethodCommands.html#MethodSCOLIBPS",
    "show_misc_options", 8,0,12,0,609,0,0,0,0,0,0,"[Show miscellaneous options]
MethodCommands.html#MethodSCOLIBPS",
    "solution_accuracy", 2,0,10,0,604,0,0,0,0,0,0,"[Desired solution target]
MethodCommands.html#MethodSCOLIBPS",
    "solution_target", 10,0,10,0,605,0,0,0,0,0,0,"[Desired solution target]
MethodCommands.html#MethodSCOLIBPS",
    "stochastic", 8,0,5,0,487,0,0,0,0,0,0,"[Stochastic pattern search]
MethodCommands.html#MethodSCOLIBPS",
    "synchronization", 8,2,8,0,499,0,0,0,0,0,0,"[Evaluation synchronization]
MethodCommands.html#MethodSCOLIBPS",
    "threshold_delta", 10,0,15,0,523,0,0,0,0,0,0,"[Threshold for offset values]
MethodCommands.html#MethodSCOLIBPS",
    "total_pattern_size", 5,0,6,0,489,0,0,0,0,0,0,"[Total number of points in pattern]
MethodCommands.html#MethodSCOLIBPS"
}
```

GuiKeyWord kw_73[14] [static]

Initial value:

```c
= {
    "constant_penalty", 8,0,4,0,513,0,0,0,0,0,0,"[Control of dynamic penalty]
MethodCommands.html#MethodSCOLIBPS",
    "constraint_penalty", 10,0,13,0,517,0,0,0,0,0,0,"[Constraint penalty]
MethodCommands.html#MethodSCOLIBPS",
    "contraction_factor", 10,0,12,0,515,0,0,0,0,0,0,"[Pattern contraction factor]
MethodCommands.html#MethodSCOLIBPS",
    "expand_after_success", 9,0,3,0,511,0,0,0,0,0,0,"[Number of consecutive improvements
before expansion] MethodCommands.html#MethodSCOLIBPS",
    "initial_delta", 10,0,10,0,521,0,0,0,0,0,0,"[Initial offset value]
MethodCommands.html#MethodSCOLIBPS",
    "misc_options", 15,0,9,0,611,0,0,0,0,0,0,"[Specify miscellaneous options]
MethodCommands.html#MethodSCOLIBDC",
    "model_pointer", 11,0,5,0,1593,0,0,0,0,0,0,"[Initial offset value]
MethodCommands.html#MethodSCOLIBPS",
    "no_expansion", 8,0,2,0,509,0,0,0,0,0,0,"[No expansion flag]
MethodCommands.html#MethodSCOLIBPS",
    "pattern_basis", 8,2,4,0,507,0,0,0,0,0,0,"[Pattern basis selection]
MethodCommands.html#MethodSCOLIBPS",
    "random_seed", 0x19,0,11,0,607,0,0,0,0,0,0,"[Random seed for stochastic pattern search]
MethodCommands.html#MethodSCOLIBPS",
    "seed", 0x19,0,7,0,499,0,0,0,0,0,0,"[Random seed for stochastic pattern search]
MethodCommands.html#MethodSCOLIBPS",
    "solution_accuracy", 2,0,10,0,604,0,0,0,0,0,0,"[Desired solution target]
MethodCommands.html#MethodSCOLIBPS",
    "solution_target", 10,0,11,0,605,0,0,0,0,0,0,"[Desired solution target]
MethodCommands.html#MethodSCOLIBPS",
    "threshold_delta", 10,0,11,0,523,0,0,0,0,0,0,"[Threshold for offset values]
MethodCommands.html#MethodSCOLIBPS",
    "total_pattern_size", 5,0,6,0,511,0,0,0,0,0,0,"[Total number of points in pattern]
MethodCommands.html#MethodSCOLIBPS"
```
GuiKeyWord kw_.74[12]  [static]
Initial value:

```csharp
= {
    {"frcg",8,0,1,1,219},
    {"linear.equality.constraint_matrix",14,0,8,0,465,0,0,0,0,0,"{linear equality coefficient matrix} MethodCommands.html#MethodMin"},
    {"linear.equality.scale_types",15,0,10,0,469,0,0,0,0,0,"{linear equality scaling types} MethodCommands.html#MethodMin"},
    {"linear.equality.scales",14,0,11,0,471,0,0,0,0,0,"{linear equality scales} MethodCommands.html#MethodMin"},
    {"linear.equality.targets",14,0,9,0,467,0,0,0,0,0,"{linear equality targets} MethodCommands.html#MethodMin"},
    {"mfd",8,0,1,1,221},
    {"model_pointer",11,0,2,0,1593}
}
```

GuiKeyWord kw_.75[10]  [static]
Initial value:

```csharp
= {
    {"drop_tolerance",10,0,1,0,1329},
    {"linear.equality.constraint_matrix",14,0,7,0,465,0,0,0,0,0,"{linear equality coefficient matrix} MethodCommands.html#MethodMin"},
    {"linear.equality.scale_types",15,0,9,0,469,0,0,0,0,0,"{linear equality scaling types} MethodCommands.html#MethodMin"},
    {"linear.equality.scales",14,0,10,0,471,0,0,0,0,0,"{linear equality scales} MethodCommands.html#MethodMin"},
    {"linear.equality.targets",14,0,8,0,467,0,0,0,0,0,"{linear equality targets} MethodCommands.html#MethodMin"},
    {"linear.inequality.constraint_matrix",14,0,2,0,455,0,0,0,0,0,"{linear inequality coefficient matrix} MethodCommands.html#MethodMin"},
    {"linear.inequality.scale_types",15,0,3,0,457,0,0,0,0,0,"{linear inequality scaling types} MethodCommands.html#MethodMin"},
    {"linear.inequality.lower_bounds",14,0,4,0,457,0,0,0,0,0,"{linear inequality lower bounds} MethodCommands.html#MethodMin"},
    {"linear.inequality.scales",14,0,7,0,463,0,0,0,0,0,"{linear inequality scales} MethodCommands.html#MethodMin"},
    {"linear.inequality.upper_bounds",14,0,5,0,459,0,0,0,0,0,"{linear inequality upper bounds} MethodCommands.html#MethodMin"},
    {"mfd",8,0,1,1,221},
    {"model_pointer",11,0,2,0,1593}
}
```

GuiKeyWord kw_.76[1]  [static]
Initial value:

```csharp
= {
    "{drop.tolerance",10,0,1,0,1329}
}
```
GuiKeyWord kw_77[15] [static]
Initial value:
= {
  "boxbehnken",8,0,1,1,1319,0,0,0,0,0,0,"[CHOOSE DACE type]",
  "central_composite",8,0,1,1,1321,
  "fixed_seed",8,0,5,0,1351,0,0,0,0,0,"[Fixed seed flag] MethodCommands.html#MethodDDACE"
},
  "grid",8,0,1,1,1309,
  "main_effects",8,0,2,0,1323,0,0,0,0,0,"[Main effects] MethodCommands.html#MethodDDACE",
  "model_pointer",11,0,7,0,1593,
  "oa_lhs",8,0,1,1,1317,
  "oas",8,0,1,1,1313,
  "quality_metrics",8,0,3,0,1325,0,0,0,0,0,"[Quality metrics] MethodCommands.html#MethodDDACE",
  "random",8,0,1,1,1311,
  "samples",9,0,8,0,1361,0,0,0,0,0,"[Number of samples] MethodCommands.html#MethodNonCMC",
  "seed",Gu19,9,0,1363,0,0,0,0,0,"[Refinement seed] MethodCommands.html#MethodNonDLocalRel",
  "symbols",9,0,6,0,1333,0,0,0,0,0,"[Number of symbols] MethodCommands.html#MethodDDACE",
  "variance_based_decomp",8,1,4,0,1327,kw_76,0,0,0,0,0,"[Variance based decomposition] MethodCommands.html#MethodDDACE"
}

GuiKeyWord kw_78[15] [static]
Initial value:
= {
  "bfgs",8,0,1,1,207,
  "frcg",8,0,1,1,203,
  "linear_inequality_constraint_matrix",14,0,0,0,465,0,0,0,0,0,0,"[Linear equality coefficient matrix] MethodCommands.html#MethodMin",
  "linear_inequality_scale_types",15,0,10,0,469,0,0,0,0,0,0,"[Linear equality scaling types] MethodCommands.html#MethodMin",
  "linear_inequality_scalars",14,0,11,0,471,0,0,0,0,0,0,"[Linear equality scales] MethodCommands.html#MethodMin",
  "linear_inequality_targets",14,0,9,0,467,0,0,0,0,0,0,"[Linear equality targets] MethodCommands.html#MethodMin",
  "linear_inequality_constraint_matrix",14,0,3,0,455,0,0,0,0,0,0,"[Linear inequality coefficient matrix] MethodCommands.html#MethodMin",
  "linear_inequality_lower_bounds",14,0,4,0,457,0,0,0,0,0,0,"[Linear inequality lower bounds] MethodCommands.html#MethodMin",
  "linear_inequality_upper_bounds",14,0,5,0,459,0,0,0,0,0,0,"[Linear inequality upper bounds] MethodCommands.html#MethodMin",
  "mmfd",8,0,1,1,205,
  "model_pointer",11,0,2,0,1593,
  "slp",8,0,1,1,209,
  "sqp",8,0,1,1,211,
}

GuiKeyWord kw_79[2] [static]
Initial value:
= {
  "annotated",8,0,1,0,669,
  "freeform",8,0,1,0,671,
}
GuiKeyWord kw_80[2]  [static]
Initial value:

```c
= {
    {"dakota",8,0,1,1,657},
    {"surfpack",8,0,1,1,655}
}
```

GuiKeyWord kw_81[2]  [static]
Initial value:

```c
= {
    {"annotated",8,0,1,0,663},
    {"freeform",8,0,1,0,665}
}
```

GuiKeyWord kw_82[7]  [static]
Initial value:

```c
= {
    {"export_points_file",11,2,4,0,667,kw_79,0.,0.,0.,0,"{File name for exporting approximation-based samples from evaluating the GP} MethodCommands.html#MethodEG"},
    {"gaussian_process",8,2,1,0,653,kw_80,0.,0.,0.,0,"{GP selection} MethodCommands.html#MethodEG"},
    {"import_points_file",11,2,3,0,661,kw_81,0.,0.,0.,0,"{File name for points to be imported as the basis for the initial GP} MethodCommands.html#MethodEG"},
    {"model_pointer",11,0,6,0,1593},
    {"seed",0x19,0,5,0,673,0.,0.,0.,0.,0,"{Random seed} MethodCommands.html#MethodEG"},
    {"use_derivatives",8,0,2,0,659,0.,0.,0.,0.,0,"{Derivative usage} MethodCommands.html#MethodEG"}  
```

GuiKeyWord kw_83[9]  [static]
Initial value:

```c
= {
    {"batch_size",9,0,2,0,1079},
    {"distribution",8,2,6,0,1127,kw_19,0.,0.,0.,0.,0,"{Distribution type} MethodCommands.html#MethodNonD"},
    {"emulator_samples",9,0,1,0,1077},
    {"gen_reliability_levels",14,1,8,0,1137,kw_20,0.,0.,0.,0.,0,"{Generalized reliability levels} MethodCommands.html#MethodNonD"},
    {"model_pointer",11,0,3,0,1593},
    {"probability_levels",14,1,7,0,1133,kw_21,0.,0.,0.,0.,0,"{Probability levels} MethodCommands.html#MethodNonD"},
    {"rng",8,2,9,0,1141,kw_22,0.,0.,0.,0.,0,"{Random number generator} MethodCommands.html#MethodNonDMC"},
    {"samples",9,0,4,0,1361,0.,0.,0.,0.,0.,0."{Number of samples} MethodCommands.html#MethodNonDMC"},
    {"seed",0x19,0,5,0,1363,0.,0.,0.,0.,0.,0,"{Refinement seed} MethodCommands.html#MethodNonDLocalRel"}
```
GuiKeyWord kw_84[3] [static]
Initial value:
= {
    "grid",8,0,1,1,1349,0,0,0,0,0,"[CHOOSE trial type]",
    "halton",8,0,1,1,1351,
    "random",8,0,1,1,1353,0,0,0,0,0,"@
}

GuiKeyWord kw_85[1] [static]
Initial value:
= {
    "drop_tolerance",10,0,1,0,1343
}

GuiKeyWord kw_86[9] [static]
Initial value:
= {
    "fixed_seed",8,0,4,0,1345,0,0,0,0,0,"[Fixed seed flag]
MethodCommands.html#MethodFSUDACE"},
    {"latinate",8,0,1,0,1337,0,0,0,0,0,"[Latinization of samples]
MethodCommands.html#MethodFSUDACE"},
    {"model_pointer",11,0,7,0,1593},
    {"num_trials",8,0,6,0,1355,0,0,0,0,0,"[Number of trials ]
MethodCommands.html#MethodFSUDACE"},
    {"quality_metrics",8,0,2,0,1339,0,0,0,0,0,"[Quality metrics]
MethodCommands.html#MethodFSUDACE"},
    {"samples",9,0,8,0,1361,0,0,0,0,0,"[Number of samples] MethodCommands.html#MethodNonDMC"},
    {"seed",0x19,0,9,0,1363,0,0,0,0,0,"[Refinement seed]
MethodCommands.html#MethodNonLocalRel"},
    {"trial_type",8,3,0,0,1347,kw_84,0,0,0,0,"[Trial type] MethodCommands.html#MethodFSUDACE"},
    {"variance_based_decomp",8,1,3,0,1341,kw_85,0,0,0,0,0,"[Variance based decomposition]
MethodCommands.html#MethodFSUDACE"},
}

GuiKeyWord kw_87[1] [static]
Initial value:
= {
    "drop_tolerance",10,0,1,0,1543
}

GuiKeyWord kw_88[11] [static]
Initial value:
= {
    "fixed_sequence",8,0,6,0,1547,0,0,0,0,0,"[Fixed sequence flag]
MethodCommands.html#MethodFSUDACE"},
    {"halton",8,0,1,1,1593,0,0,0,0,0,"[CHOOSE sequence type]"},
    {"hammersley",8,0,1,1,1535},
    {"latinate",8,0,2,0,1537,0,0,0,0,0,"[Latinization of samples]
MethodCommands.html#MethodFSUDACE"},
    {"model_pointer",11,0,10,0,1593},
}
{prime_base*, 13, 0, 9, 0, 1553, 0, 0, 0, 0, 0, 0, *Prime bases for sequences*
MethodCommands.html#MethodFSUDACE*},
{quality_metrics*, 8, 0, 3, 0, 1539, 0, 0, 0, 0, 0, *Quality metrics*
MethodCommands.html#MethodFSUDACE*},
{samples*, 9, 0, 5, 0, 1545, 0, 0, 0, 0, 0, *Number of samples taken in the MCMC sampling*
MethodCommands.html#MethodNonDBayesCalib*},
{sequence_leap*, 13, 0, 8, 0, 1551, 0, 0, 0, 0, 0, *Sequence leaping indices*
MethodCommands.html#MethodFSUDACE*},
{sequence_start*, 13, 0, 7, 0, 1549, 0, 0, 0, 0, 0, *Sequence starting indices*
MethodCommands.html#MethodFSUDACE*},
{variance_based_decomp*, 8, 1, 4, 0, 1541, kw87, 0, 0, 0, 0, *Variance based decomposition*
MethodCommands.html#MethodFSUDACE*}

GuiKeyWord kw_89[2] [static]
Initial value:
= {
    {"annotated", 8, 0, 1, 0, 983},
    {"freeform", 8, 0, 1, 0, 985}
}

GuiKeyWord kw_90[2] [static]
Initial value:
= {
    {"annotated", 8, 0, 1, 0, 977},
    {"freeform", 8, 0, 1, 0, 979}
}

GuiKeyWord kw_91[2] [static]
Initial value:
= {
    {"parallel", 8, 0, 1, 1, 1001},
    {"series", 8, 0, 1, 1, 999}
}

GuiKeyWord kw_92[3] [static]
Initial value:
= {
    {"gen_reliabilities", 8, 0, 1, 1, 995},
    {"probabilities", 8, 0, 1, 1, 993},
    {"system", 8, 2, 2, 0, 997, kw91}
}

GuiKeyWord kw_93[2] [static]
Initial value:
= {
    {"compute", 8, 3, 2, 0, 991, kw92},
    {"num_response_levels", 13, 0, 1, 0, 989}
}
GuiKeyWord kw.94[11] [static]

Initial value:

```c
= {
    "distribution",8,2,8,0,1127,kw.19,0.,0.,0.,0.,"{Distribution type} MethodCommands.html#MethodNonD",
    {"emulator_samples",9,0,1,0,973},
    {"export_points_file",11,2,3,0,981,kw.89,0.,0.,0.,0.,"{File name for exporting approximation-based samples from evaluating the emulator} MethodCommands.html#MethodNonDBayesCalib"},
    {"general_reliability_levels",14,1,10,0,1137,kw.20,0.,0.,0.,0.,"{Generalized reliability levels} MethodCommands.html#MethodNonD"},
    {"import_points_file",11,2,2,0,975,kw.90,0.,0.,0.,0.,"{File name for points to be imported as the basis for the initial emulator} MethodCommands.html#MethodNonDBayesCalib"},
    {"model_pointer",11,0,5,0,1593},
    {"probability_levels",14,1,9,0,1133,kw.21,0.,0.,0.,0.,"{Probability levels} MethodCommands.html#MethodNonD"},
    {"response_levels",14,2,4,0,987,kw.93},
    {"rng",8,2,11,0,1141,kw.22,0.,0.,0.,0.,"{Random number generator} MethodCommands.html#MethodNonDMC"},
    {"samples",9,0,6,0,1361,0.,0.,0.,0.,"{Number of samples} MethodCommands.html#MethodNonDMC"},
    {"seed",0x19,0,7,0,1363,0.,0.,0.,0.,0.,"{Refinement seed} MethodCommands.html#MethodNonDLocalRel1"}
}
```

GuiKeyWord kw.95[2] [static]

Initial value:

```c
= {
    {"model_pointer",11,0,2,0,1593},
    {"seed",0x19,0,1,0,649,0.,0.,0.,0.,0.,"{Random seed} MethodCommands.html#MethodNonDMC"}
}
```

GuiKeyWord kw.96[2] [static]

Initial value:

```c
= {
    {"parallel",8,0,1,1,1125},
    {"series",8,0,1,1,1125}
}
```

GuiKeyWord kw.97[3] [static]

Initial value:

```c
= {
    {"gen_reliabilities",8,0,1,1,1119},
    {"probabilities",8,0,1,1,1117},
    {"system",8,2,2,0,1121,kw.96}
}
```

GuiKeyWord kw.98[2] [static]

Initial value:

```c
= {
    {"compute",8,3,2,0,1115,kw.97},
    {"num_response_levels",13,0,1,0,1113}
}
```
GuiKeyWord kw.99[2]  [static]
Initial value:
= {
    {"annotated", 8, 0, 1, 0, 1103},
    {"freeform", 8, 0, 1, 0, 1105}
}

GuiKeyWord kw.100[2]  [static]
Initial value:
= {
    {"dakota", 8, 0, 1, 1, 1091},
    {"surfpack", 8, 0, 1, 1, 1089}
}

GuiKeyWord kw.101[2]  [static]
Initial value:
= {
    {"annotated", 8, 0, 1, 0, 1097},
    {"freeform", 8, 0, 1, 0, 1099}
}

GuiKeyWord kw.102[5]  [static]
Initial value:
= {
    {"export_points_file", 11, 2, 4, 0, 1101, kw.99},
    {"gaussian_process", 8, 2, 1, 0, 1087, kw.100},
    {"import_points_file", 11, 2, 4, 0, 1095, kw.101, 0, 0, 0, 0, "File containing points to evaluate"},
    {"kriging", 0, 2, 1, 0, 1086, kw.100},
    {"use_derivatives", 8, 0, 2, 0, 1093}
}

GuiKeyWord kw.103[12]  [static]
Initial value:
= {
    {"distribution", 8, 2, 6, 0, 1127, kw.19, 0, 0, 0, 0, 0, 0, "Distribution type"},
    {"ea", 8, 0, 1, 0, 1107},
    {"ego", 8, 5, 1, 0, 1085, kw.102},
    {"general_reliability_levels", 14, 1, 8, 0, 1137, kw.20, 0, 0, 0, 0, "Generalized reliability levels"},
    {"lhs", 8, 0, 1, 0, 1109},
    {"model_pointer", 11, 0, 0, 3, 0, 1593},
    {"probability_levels", 14, 1, 7, 0, 1133, kw.21, 0, 0, 0, 0, "Probability levels"},
    {"rng", 8, 2, 9, 0, 1141, kw.22, 0, 0, 0, 0, 0, "Random number generator"},
    {"samples", 9, 0, 4, 0, 1361, 0, 0, 0, 0, 0, "Number of samples"},
    {"sbo", 8, 5, 1, 0, 1083, kw.102},
    {"seed", 0x19, 0, 5, 0, 1363, 0, 0, 0, 0, 0, 0, "Refinement seed"}
}

GuiKeyWord kw_104[2] [static]
Initial value:

= {
    {"mt19937",8,0,1,1,179},
    {"rnum2",8,0,1,1,1181}
}

GuiKeyWord kw_105[2] [static]
Initial value:

= {
    {"annotated",8,0,1,0,1169},
    {"freeform",8,0,1,0,1171}
}

GuiKeyWord kw_106[2] [static]
Initial value:

= {
    {"dakota",8,0,1,1,1157},
    {"surfpack",8,0,1,1,1155}
}

GuiKeyWord kw_107[2] [static]
Initial value:

= {
    {"annotated",8,0,1,0,1163},
    {"freeform",8,0,1,0,1165}
}

GuiKeyWord kw_108[5] [static]
Initial value:

= {
    {"export_points_file",11,2,4,0,1167,kw_105,0,,0,,0,,0,,0,,0,"File name for exporting approximation-based samples from evaluating the GP" MethodCommands.html#MethodNonDGlobalIntervalEst"},
    {"gaussian_process",8,2,1,0,1153,kw_106,0,,0,,0,,0,"EGO GP selection" MethodCommands.html#MethodNonDGlobalIntervalEst"},
    {"import_points_file",11,5,3,0,1161,kw_107,0,,0,,0,,0,"File name for points to be imported as the basis for the initial GP" MethodCommands.html#MethodNonDGlobalIntervalEst"},
    {"kriging",0,2,1,0,1152,kw_106},
    {"use_derivatives",8,0,2,0,1159,0,,0,,0,,0,"Derivative usage" MethodCommands.html#MethodNonDGlobalIntervalEst"}
}
GuiKeyWord kw_109[8] [static]
Initial value:
= {
  {"ea",8,0,1,0,1173},
  {"ego",8,5,1,0,1151,kw},
  {"lhs",8,0,1,0,1175},
  {"model_pointer",11,0,3,0,1593},
  {"rng",8,2,2,0,1177,kw,0,0,0,0,0,"Random seed generator" MethodCommands.html#MethodNonDGlobalIntervalEst"},
  {"samples",9,0,4,0,1361,0,0,0,0,0,"Number of samples" MethodCommands.html#MethodNonDMC"},
  {"sbo",8,5,1,0,1149,kw},
  {"seed",0x19,0,5,0,1363,0,0,0,,0,"Refinement seed" MethodCommands.html#MethodNonDLocalRel"}
}

GuiKeyWord kw_110[2] [static]
Initial value:
= {
  {"complementary",8,0,1,1,1521},
  {"cumulative",8,0,1,1,1519}
}

GuiKeyWord kw_111[1] [static]
Initial value:
= {
  {"num_gen_reliability.levels",13,0,1,0,1529}
}

GuiKeyWord kw_112[1] [static]
Initial value:
= {
  {"num_probability.levels",13,0,1,0,1525}
}

GuiKeyWord kw_113[2] [static]
Initial value:
= {
  {"annotated",8,0,1,0,1487},
  {"freeform",8,0,1,0,1489}
}

GuiKeyWord kw_114[2] [static]
Initial value:
= {
  {"annotated",8,0,1,0,1481},
  {"freeform",8,0,1,0,1483}
}
GuiKeyWord kw_115[2] [static]
Initial value:
= {
   "parallel",8,0,1,1,1515,
   "series",8,0,1,1,1513
}

GuiKeyWord kw_116[3] [static]
Initial value:
= {
   "gen_reliabilities",8,0,1,1,1509,
   "probabilities",8,0,1,1,1507,
   "system",8,2,2,0,1511,kw
}

GuiKeyWord kw_117[2] [static]
Initial value:
= {
   "compute",8,3,2,0,1505,kw_116,
   "num_response_levels",13,0,1,1503
}

GuiKeyWord kw_118[2] [static]
Initial value:
= {
   "mt19937",8,0,1,1,1497,
   "rnum2",8,0,1,1,1499
}

GuiKeyWord kw_119[16] [static]
Initial value:
= {
   "dakota",8,0,2,0,1477,
   "distribution",8,2,10,0,1517,kw_110,
   "export_points_file",11,2,4,0,1485,kw_113,0.,0.,0.,0,"{File name for exporting approximation-based samples from evaluating the GP} MethodCommands.html#MethodNonDGlobalRel"},
   "gen_reliability_levels",14,1,12,0,1527,kw_111,
   "import_points_file",11,6,3,0,1479,kw_114,0.,0.,0.,0,"{File name for points to be imported as the basis for the initial GP} MethodCommands.html#MethodNonDGlobalRel"},
   "model_pointer",11,0,9,0,1593,
   "probability_levels",14,1,11,0,1523,kw_112,
   "response_levels",14,2,8,0,1501,kw_117,
   "rng",8,2,7,0,1495,kw_118,
   "seed",0x19,0,6,0,1493,0,0.,0.,0,"{Random seed for initial GP construction} MethodCommands.html#MethodNonDGlobalRel"},
   "surfpack",8,0,2,0,1475,
   "u_gaussian_process",8,0,1,1,1473,
   "u_kriging",0,0,1,1,1472,
   "use_derivatives",8,0,5,0,1491,0,0.,0.,0.,0,"{Derivative usage} MethodCommands.html#MethodNonDGlobalRel"},
   "x_gaussian_process",8,0,1,1,1471,
   "x_kriging",0,0,1,1,1470
}
GuiKeyWord kw_120[2] [static]
Initial value:
= {
   {"master",8,0,1,1,113},
   {"peer",8,0,1,1,115}
}

GuiKeyWord kw_121[1] [static]
Initial value:
= {
   {"model_pointer_list",11,0,1,0,77,0,0,0,0,0,0,0},"{List of model pointers}
   MethodCommands.html#MethodMetaHybrid"}

GuiKeyWord kw_122[2] [static]
Initial value:
= {
   {"method_name_list",15,1,1,1,75,kw_121,0,0,0,0,0,0,0,"{List of method names}
   MethodCommands.html#MethodMetaHybrid"},
   {"method_pointer_list",15,0,1,1,79,0,0,0,0,0,0,0,"{List of method pointers}
   MethodCommands.html#MethodMetaHybrid"}
}

GuiKeyWord kw_123[1] [static]
Initial value:
= {
   {"global_model_pointer",11,0,1,0,61,0,0,0,0,0,0,0,0,"{Pointer to the global model
   specification} MethodCommands.html#MethodMetaHybrid"}
}

GuiKeyWord kw_124[1] [static]
Initial value:
= {
   {"local_model_pointer",11,0,1,0,67,0,0,0,0,0,0,0,0,"{Pointer to the local model specification}
   MethodCommands.html#MethodMetaHybrid"}
}

GuiKeyWord kw_125[5] [static]
Initial value:
= {
   {"global_method_name",11,1,1,1,59,kw_123,0,0,0,0,0,0,0,"{Name of the global method}
   MethodCommands.html#MethodMetaHybrid"},
   {"global_method_pointer",11,0,1,1,63,0,0,0,0,0,0,0,0,"{Pointer to the global method specification} MethodCommands.html#MethodMetaHybrid"},
   {"local_method_name",11,1,2,2,65,kw_124,0,0,0,0,0,0,0,"{Name of the local method}
   MethodCommands.html#MethodMetaHybrid"},
   {"local_method_pointer",11,0,2,2,69,0,0,0,0,0,0,0,0,"{Pointer to the local method specification} MethodCommands.html#MethodMetaHybrid"},
   {"local_search_probability",10,0,3,0,71,0,0,0,0,0,0,0,0,"{Probability of executing local searches} MethodCommands.html#MethodMetaHybrid"}
}
GUIKeyWord kw_126[1] [static]
Initial value:

```java
= {
    "model_pointer_list",11,0,1,0,53,0,0..0,0,"{List of model pointers}
    MethodCommands.html#MethodMetaHybrid"
}
```

GUIKeyWord kw_127[2] [static]
Initial value:

```java
= {
    "method_name_list",15,1,1,1,51,kw_126,0,0,0,0,"{List of method names}
    MethodCommands.html#MethodMetaHybrid"],
    "method_pointer_list",15,0,1,1,55,0,0,0,0,0,"{List of method pointers}
    MethodCommands.html#MethodMetaHybrid"
}
```

GUIKeyWord kw_128[8] [static]
Initial value:

```java
= {
    "collaborative",8,2,1,1,73,kw_122,0,0,0,0,"{Collaborative hybrid}
    MethodCommands.html#MethodMetaHybrid"],
    "coupled",0,5,1,1,56,kw_125,
    "embedded",8,5,1,1,57,kw_125,0,0,0,0,"{Embedded hybrid}
    MethodCommands.html#MethodMetaHybrid"],
    "iterator_scheduling",8,2,3,0,111,kw_120,0,0,0,0,"{Message passing configuration for
    scheduling of iterator jobs} MethodCommands.html#MethodMeta"],
    "iterator_servers",0x19,0,2,0,109,0,0,0,0,"{Number of iterator servers}
    MethodCommands.html#MethodMeta"],
    "processors_per_iterator",0x19,0,4,0,117,0,0,0,0,0,"{Number of processors per iterator
    server} MethodCommands.html#MethodMeta"],
    "sequential",8,2,1,1,49,kw_127,0,0,0,0,"{Sequential hybrid}
    MethodCommands.html#MethodMetaHybrid"],
    "uncoupled",8,2,1,1,48,kw_127
```

GUIKeyWord kw_129[2] [static]
Initial value:

```java
= {
    "parallel",8,0,1,1,969],
    "series",8,0,1,1,967]
```

GUIKeyWord kw_130[3] [static]
Initial value:

```java
= {
    "gen_reliabilities",8,0,1,1,963],
    "probabilities",8,0,1,1,961],
    "system",8,2,2,0,965,kw_129
```
GuiKeyWord kw_131[2]  [static]
Initial value:
  = {
    "compute", 8, 3, 2, 0, 959, kw_130,
    "num_response_levels", 13, 0, 1, 0, 957
  }

GuiKeyWord kw_132[12]  [static]
Initial value:
  = {
    "adapt_import", 8, 0, 1, 1, 949,
    "distribution", 8, 2, 7, 0, 1127, kw_19, 0, 0, 0, 0, /*Distribution type MethodCommands.html#MethodNonD*/,
    "gen_reliability_levels", 14, 1, 0, 0, 1137, kw_20, 0, 0, 0, 0, /*Generalized reliability levels MethodCommands.html#MethodNonD*/,
    "import", 8, 0, 1, 1, 947,
    "mm_adapt_import", 8, 0, 1, 1, 951,
    "model_pointer", 11, 0, 4, 0, 1593,
    "probability_levels", 14, 1, 8, 0, 1133, kw_21, 0, 0, 0, 0, /*Probability levels MethodCommands.html#MethodNonD*/,
    "refinement_samples", 9, 0, 2, 0, 953,
    "response_levels", 14, 2, 3, 0, 955, kw_131,
    "rng", 8, 4, 10, 0, 1141, kw_22, 0, 0, 0, 0, 0, /*Random number generator MethodCommands.html#MethodNonD*/,
    "samples", 9, 0, 5, 0, 1361, 0, 0, 0, 0, 0, /*Number of samples MethodCommands.html#MethodNonDMC*/,
    "seed", 0x19, 0, 6, 0, 1363, 0, 0, 0, 0, 0, /*Refinement seed MethodCommands.html#MethodNonDLocalRel*/
  }

GuiKeyWord kw_133[2]  [static]
Initial value:
  = {
    "annotated", 8, 0, 1, 0, 1569,
    "freeform", 8, 0, 1, 0, 1571
  }

GuiKeyWord kw_134[3]  [static]
Initial value:
  = {
    "import_points_file", 11, 2, 1, 1, 1567, kw_133,
    "list_of_points", 14, 0, 1, 1, 1565, 0, 0, 0, 0, 0, /*List of points to evaluate MethodCommands.html#MethodSIPS*/,
    "model_pointer", 11, 0, 2, 0, 1593
  }

GuiKeyWord kw_135[2]  [static]
Initial value:
  = {
    "complementary", 8, 0, 1, 1, 1399,
    "cumulative", 8, 0, 1, 1, 1397
  }
GuiKeyWord kw_136[1]  [static]
Initial value:
= {
    "num_gen.reliability_levels",13,0,1,0,1393
}

GuiKeyWord kw_137[1]  [static]
Initial value:
= {
    "num_probability_levels",13,0,1,0,1389
}

GuiKeyWord kw_138[2]  [static]
Initial value:
= {
    "parallel",8,0,1,1,1385,
    "series",8,0,1,1,1383
}

GuiKeyWord kw_139[3]  [static]
Initial value:
= {
    "gen_reliabilities",8,0,1,1,1379,
    "probabilities",8,0,1,1,1377,
    "system",8,2,2,0,1381,kw_138
}

GuiKeyWord kw_140[2]  [static]
Initial value:
= {
    "compute",8,3,2,0,1375,kw_139,
    "num_response_levels",13,0,1,0,1373
}

GuiKeyWord kw_141[7]  [static]
Initial value:
= {
    "distribution",8,2,5,0,1395,kw_135,
    "gen_reliability_levels",14,1,4,0,1391,kw_136,
    "model_pointer",11,0,6,0,1593,
    "nip",8,0,1,0,1369,
    "probability_levels",14,1,3,0,1387,kw_137,
    "response_levels",14,2,2,0,1371,kw_140,
    "sqp",8,0,1,0,1367
}
GuiKeyWord kw_142[3]  [static]
Initial value:
= {
  "model_pointer", 11, 0, 2, 0, 1593,
  "nip", 8, 0, 1, 0, 1405,
  "sqp", 8, 0, 1, 0, 1403
}

GuiKeyWord kw_143[5]  [static]
Initial value:
= {
  "adapt_import", 8, 0, 1, 1, 1439,
  "import", 8, 0, 1, 1, 1433,
  "mm_adapt_import", 8, 0, 1, 1, 1441,
  "refinement_samples", 9, 0, 2, 0, 1443,
  "seed", 0x19, 0, 3, 0, 1445, 0, 0, 0, 0, 0,"{Random seed}
  MethodCommands.html#MethodNonDBayesCalib"
}

GuiKeyWord kw_144[4]  [static]
Initial value:
= {
  "first_order", 8, 0, 1, 1, 1431,
  "probability_refinement", 8, 0, 2, 0, 1435, kw_143,
  "sample_refinement", 0, 5, 2, 0, 1434, kw_143,
  "second_order", 8, 0, 1, 1, 1433
}

GuiKeyWord kw_145[10]  [static]
Initial value:
= {
  "integration", 8, 4, 3, 0, 1429, kw_144, 0, 0, 0, 0,"{Integration method}
  MethodCommands.html#MethodNonDLocalRel"},
  "nip", 8, 0, 2, 0, 1427,
  "no_approx", 8, 0, 1, 1, 1423,
  "sqp", 8, 0, 2, 0, 1425,
  "x_taylor_mean", 8, 0, 1, 1, 1413,
  "u_taylor_mpp", 8, 0, 1, 1, 1417,
  "u_two_point", 8, 0, 1, 1, 1421,
  "x_taylor_mean", 8, 0, 1, 1, 1411,
  "x_taylor_mpp", 8, 0, 1, 1, 1415,
  "x_two_point", 8, 0, 1, 1, 1419
}

GuiKeyWord kw_146[1]  [static]
Initial value:
= {
  "num_reliability_levels", 13, 0, 1, 0, 1467
}
GuiKeyWord kw_147[2] [static]
Initial value:

```cpp
= {
    {"parallel",8,0,1,1,1463},
    {"series",8,0,1,1,1461}
}
```

GuiKeyWord kw_148[4] [static]
Initial value:

```cpp
= {
    {"gen_reliabilities",8,0,1,1,1457},
    {"probabilities",8,0,1,1,1453},
    {"reliabilities",8,0,1,1,1455},
    {"system",8,2,2,0,1459,kw_147}
}
```

GuiKeyWord kw_149[2] [static]
Initial value:

```cpp
= {
    {"compute",8,4,2,0,1451,kw_148},
    {"num_response_levels",13,0,1,0,1449}
}
```

GuiKeyWord kw_150[7] [static]
Initial value:

```cpp
= {
    {"distribution",8,2,5,0,1517,kw_110},
    {"gen_reliability_levels",14,1,7,0,1527,kw_111},
    {"model_pointer",13,0,4,0,1593},
    {"mpp_search",8,10,1,0,1409,kw_145,0.,0.,0.,0.,"{MPP search type MethodCommands.html#MethodNonLocalRel"},
    {"probability_levels",14,1,6,0,1523,kw_112},
    {"reliability_levels",14,1,3,0,1465,kw_146},
    {"response_levels",14,2,2,0,1447,kw_149}
}
```

GuiKeyWord kw_151[16] [static]
Initial value:

```cpp
= {
    {"display_all_evaluations",8,0,6,0,331,0,0,0,0,0,0,0,0,"{Display NOMAD evaluations}
MethodCommands.html#MethodNOMADDCC"},
    {"display_format",11,0,4,0,327},
    {"function_precision",10,0,1,0,321,0,0,0,0,0,0,"{Function Evaluation Precision}
MethodCommands.html#MethodNOMADDCC"},
    {"history_file",11,0,3,0,325,0,0,0,0,0,"{NOMAD History File}
MethodCommands.html#MethodNOMADDCC"},
    {"linear_equality_constraint_matrix",14,0,13,0,465,0,0,0,0,0,0,"{Linear equality_coefficient_matrix} MethodCommands.html#MethodMin"},
    {"linear_equality_scales",15,0,15,0,469,0,0,0,0,0,0,"{Linear equality scaling types}
MethodCommands.html#MethodMin"},
    {"linear_equality_scales",14,0,16,0,471,0,0,0,0,0,0,"{Linear equality scales}
MethodCommands.html#MethodMin"},
12.1. DAKOTA NAMESPACE REFERENCE

```{linenomath}
{
"linear_equality_targets",14,0,14,0,467,0,0.,0.,0.,0,"{Linear equality targets MethodCommands.html#MethodMin"},
"linear_inequality_constraint_matrix",14,0,8,0,455,0,0.,0.,0.,0,"{Linear inequality coefficient matrix MethodCommands.html#MethodMin"},
"linear_inequality_lower_bounds",14,0,9,0,457,0,0.,0.,0.,0,"{Linear inequality lower bounds MethodCommands.html#MethodMin"},
"linear_inequality_scale_types",15,0,11,0,461,0,0.,0.,0.,0,"{Linear inequality scaling types MethodCommands.html#MethodMin"},
"linear_inequality_scales",14,0,12,0,463,0,0.,0.,0.,0,"{Linear inequality scales MethodCommands.html#MethodMin"},
"linear_inequality_upper_bounds",14,0,10,0,459,0,0.,0.,0.,0,"{Linear inequality upper bounds MethodCommands.html#MethodMin"},
"model_pointer",11,0,7,0,1593,
"seed",0x19,0,2,0,323,0,0.,0.,0.,0,"{Random Seed MethodCommands.html#MethodNOMADDC"},
"variable_neighborhood_search",10,0,5,0,329}
```

GuiKeyWord kw_152[2] [static]

Initial value:

= {
  "num_offspring",0x19,0,2,0,433,0,0.,0.,0.,0,"{Number of offspring in random shuffle crossover MethodCommands.html#MethodJEGADC"},
  "num_parents",0x19,0,1,0,431,0,0.,0.,0.,0,"{Number of parents in random shuffle crossover MethodCommands.html#MethodJEGADC"}
}

GuiKeyWord kw_153[5] [static]

Initial value:

= {
  "crossover_rate",10,0,2,0,435,0,0.,0.,0.,0,"{Crossover rate MethodCommands.html#MethodJEGADC"},
  "multi_point_binary",9,0,1,1,423,0,0.,0.,0.,0,"{Multi point binary crossover MethodCommands.html#MethodJEGADC"},
  "multi_point_parameterized_binary",9,0,1,1,425,0,0.,0.,0.,0,"{Multi point parameterized binary crossover MethodCommands.html#MethodJEGADC"},
  "multi_point_real",9,0,1,1,427,0,0.,0.,0.,0,"{Multi point real crossover MethodCommands.html#MethodJEGADC"},
  "shuffle_random",8,2,1,1,429,kw_152,0.,0.,0.,0,"{Random shuffle crossover MethodCommands.html#MethodJEGADC"}
}

GuiKeyWord kw_154[3] [static]

Initial value:

= {
  "flat_file",11,0,1,1,419,
  "simple_random",8,0,1,1,415,
  "unique_random",8,0,1,1,417}

GuiKeyWord kw_155[1] [static]

Initial value:

= {
  "mutation_scale",10,0,1,0,449,0,0.,0.,0.,0,"{Mutation scale MethodCommands.html#MethodJEGADC"}
}
GuiKeyWord kw_156[6] [static]
Initial value:

```cpp
= {
    "bit_random",8,0,1,1,439,
    "mutation_rate",10,0,2,0,451,0,0,0,0,0,"{Mutation rate MethodCommands.html#MethodJEGADC"},
    "offset_cauchy",8,1,1,1,445,kw_155,
    "offset_normal",8,1,1,1,443,kw_155,
    "offset_uniform",8,1,1,1,447,kw_155,
    "replace_uniform",8,0,1,1,441
}
```

GuiKeyWord kw_157[3] [static]
Initial value:

```cpp
= {
    "metric_tracker",8,0,1,1,365,0,0,0,0,0,"{Convergence type MethodCommands.html#MethodJEGAMOGA"},
    "num_generations",0x29,0,3,0,369,0,0,0,0,0,"{Number generations for metric_tracker converger MethodCommands.html#MethodJEGAMOGA"},
    "percent_change",10,0,2,0,367,0,0,0,0,0,"{Percent change limit for metric_tracker converger MethodCommands.html#MethodJEGAMOGA"}
}
```

GuiKeyWord kw_158[2] [static]
Initial value:

```cpp
= {
    "domination_count",8,0,1,1,339,
    "layer_rank",8,0,1,1,337
}
```

GuiKeyWord kw_159[1] [static]
Initial value:

```cpp
= {
    "num_designs",0x29,0,1,0,361,0,2,0,0,0,0,"{Number designs to keep for max_designs_nicher MethodCommands.html#MethodJEGAMOGA"}
}
```

GuiKeyWord kw_160[3] [static]
Initial value:

```cpp
= {
    "distance",14,0,1,1,357,
    "max_designs",14,1,1,1,339,kw_159,
    "radial",14,0,1,1,355
}
```
GuiKeyWord kw_161[1] [static]

Initial value:

= {
    "orthogonal_distance":14,0,1,1,373,0,0,0,0,0,"Post processor distance"
    MethodCommands.html#MethodJEGAMOGA",
}

GuiKeyWord kw_162[2] [static]

Initial value:

= {
    "shrinkage_fraction":10,0,1,0,351,
    "shrinkage_percentage":2,0,1,0,350
}

GuiKeyWord kw_163[4] [static]

Initial value:

= {
    "below_limit":10,2,1,1,349,kw_162[0],0,0,0,0,"Below limit selection"
    MethodCommands.html#MethodJEGADC",
    "elitist":8,0,1,1,343,
    "roulette_wheel":8,0,1,1,345,
    "unique_roulette_wheel":8,0,1,1,347
}

GuiKeyWord kw_164[22] [static]

Initial value:

= {
    "convergence_type":8,3,4,0,363,kw_157",
    "crossover_type":8,5,20,0,421,kw_153,0,0,0,0,"Crossover type"
    MethodCommands.html#MethodJEGADC",
    "fitness_type":8,2,1,0,335,kw_158,0,0,0,0,"Fitness type"
    MethodCommands.html#MethodJEGAMOGA",
    "initialization_type":8,3,19,0,413,kw_154,0,0,0,0,"Initialization type"
    MethodCommands.html#MethodJEGADC",
    "linear_inequality_constraint_matrix":14,0,12,0,465,0,0,0,0,0,"Linear inequality constraint matrix"
    MethodCommands.html#MethodMin",
    "linear_inequality_type":15,0,14,0,469,0,0,0,0,0,"Linear inequality type"
    MethodCommands.html#MethodJEGADC",
    "linear_inequality_scale_types":14,0,15,0,471,0,0,0,0,0,"Linear inequality scale types"
    MethodCommands.html#MethodMin",
    "linear_inequality_scales":14,0,15,0,471,0,0,0,0,0,"Linear inequality scales"
    MethodCommands.html#MethodMin",
    "linear_inequality_targets":14,0,13,0,467,0,0,0,0,0,"Linear inequality targets"
    MethodCommands.html#MethodMin",
    "linear_upper_bounds":14,0,12,0,455,0,0,0,0,0,"Linear upper bounds"
    MethodCommands.html#MethodMin",
    "linear_upper_bound":14,0,12,0,455,0,0,0,0,0,"Linear upper bound"
    MethodCommands.html#MethodMin",
    "log_file":11,0,17,0,409,0,0,0,0,0,"Log file"
    MethodCommands.html#MethodJEGADC",
    "model_pointer":11,0,6,0,1593,
    "mutation_type":8,6,21,0,437,kw_156,0,0,0,0,0,"Mutation type"
    MethodCommands.html#MethodJEGADC",
    "
{"niching_type",8,3,3,0,353,kw160,0,0,0,0,0,0,"Niche pressure type"
MethodCommands.html#MethodJEGAMOGA"},
{"population_size",0x29,0,156,0,407,0,0,0,0,0,"Number of population members"
MethodCommands.html#MethodJEGADDC"},
{"postprocessor_type",8,1,5,0,371,kw161,0,0,0,0,0,"Post-processor type"
MethodCommands.html#MethodJEGAMOGA"},
{"print_each_pop",8,0,18,0,411,0,0,0,0,0,"Population output"
MethodCommands.html#MethodJEGADDC"},
{"replacement_type",8,4,2,0,341,kw163,0,0,0,0,0,"Replacement type"
MethodCommands.html#MethodJEGAMOGA"},
{"seed",0x19,0,22,0,453,0,0,0,0,0,"Random seed" MethodCommands.html#MethodJEGADDC"}
}

GuiKeyWord kw_165[1] [static]
Initial value:
= {
  {"model_pointer",11,0,1,0,85,0,0,0,0,0,"Model pointer" MethodCommands.html#MethodMeta"}
}

GuiKeyWord kw_166[1] [static]
Initial value:
= {
  {"seed",9,0,1,0,91,0,0,0,0,0,"Seed for random starting points"
MethodCommands.html#MethodMetaMultiStart"}
}

GuiKeyWord kw_167[7] [static]
Initial value:
= {
  {"iterator_scheduling",8,2,5,0,111,kw120,0,0,0,0,0,"Message passing configuration for scheduling of iterator jobs" MethodCommands.html#MethodMeta"},
  {"iterator_servers",0x19,0,4,0,109,0,0,0,0,0,"Number of iterator servers"
MethodCommands.html#MethodMeta"},
  {"method_name",11,1,1,1,83,kw165,0,0,0,0,0,"Identification of a sub-method by name (no separate specification block)" MethodCommands.html#MethodMeta"},
  {"method_pointer",11,0,1,1,87,0,0,0,0,0,"Identification of a sub-method by pointer to a separate specification block" MethodCommands.html#MethodMeta"},
  {"processors_per_iterator",0x19,0,6,0,117,0,0,0,0,0,"Number of processors per iterator server" MethodCommands.html#MethodMeta"},
  {"random_starts",9,1,2,0,89,kw166,0,0,0,0,0,"Number of random starting points"
MethodCommands.html#MethodMetaMultiStart"},
  {"starting_points",14,0,3,0,93,0,0,0,0,0,"List of user-specified starting points"
MethodCommands.html#MethodMetaMultiStart"}
}

GuiKeyWord kw_168[2] [static]
Initial value:
= {
  {"model_pointer",11,0,2,0,159},
  {"partitions",13,0,1,1,1581,0,0,0,0,0,"Partitions per variable"
MethodCommands.html#MethodPSMPS"}
}
GuiKeyWord kw_169[5]  [static]
Initial value:

= {
  {"min_boxsize_limit",10,0,2,0,641,0,0.,0.,0.,0.,"Min boxsize limit"}
  MethodCommands.html#MethodNCSUDC"},
  {"model_pointer",11,0,4,0,1593},
  {"solution_accuracy",2,0,1,0,638},
  {"solution_target",10,0,1,0,639,0,0.,0.,0.,0.,"Solution Target"}
  MethodCommands.html#MethodNCSUDC"},
  {"volume_boxsize_limit",10,0,3,0,643,0,0.,0.,0.,0.,"Volume boxsize limit"}
  MethodCommands.html#MethodNCSUDC"}

GuiKeyWord kw_170[10]  [static]
Initial value:

= {
  {"absolute_conv_tol",10,0,2,0,617,0,0.,0.,0.,0.,"Absolute function convergence tolerance"}
  MethodCommands.html#MethodLSNL2SOL"},
  {"covariance",9,0,8,0,629,0,0.,0.,0.,0.,"Covariance post-processing"}
  MethodCommands.html#MethodLSNL2SOL"},
  {"false_conv_tol",10,0,6,0,625,0,0.,0.,0.,0.,"False convergence tolerance"}
  MethodCommands.html#MethodLSNL2SOL"},
  {"function_precision",10,0,1,0,615,0,0.,0.,0.,0.,"Relative precision in least squares terms"}
  MethodCommands.html#MethodLSNL2SOL"},
  {"initial_trust_radius",10,0,7,0,627,0,0.,0.,0.,0.,"Initial trust region radius"}
  MethodCommands.html#MethodLSNL2SOL"},
  {"model_pointer",11,0,10,0,1593},
  {"regression_diagnostics",8,0,9,0,631,0,0.,0.,0.,0.,"Regression diagnostics post-processing"}
  MethodCommands.html#MethodLSNL2SOL"},
  {"singular_conv_tol",10,0,4,0,621,0,0.,0.,0.,0.,"Singular convergence tolerance"}
  MethodCommands.html#MethodLSNL2SOL"},
  {"singular_radius",10,0,5,0,623,0,0.,0.,0.,0.,"Step limit for sctol"}
  MethodCommands.html#MethodLSNL2SOL"},
  {"x_conv_tol",10,0,3,0,619,0,0.,0.,0.,0.,"Convergence tolerance for change in parameter vector"}
  MethodCommands.html#MethodLSNL2SOL"}

GuiKeyWord kw_171[2]  [static]
Initial value:

= {
  {"parallel",8,0,1,1,1073},
  {"series",8,0,1,1,1071}  
}

GuiKeyWord kw_172[3]  [static]
Initial value:

= {
  {"gen_reliabilities",8,0,1,1,1067},
  {"probabilities",8,0,1,1,1065},
  {"system",8,0,2,0,1069,kw_171}  
}
GuiKeyWord kw_173[2] [static]
Initial value:
= {
    \{"compute",8,3,2,0,1063,kw_172},
    \{"num_response_levels",13,0,1,0,1061}\}

GuiKeyWord kw_174[8] [static]
Initial value:
= {
    \{"distribution",8,2,5,0,1127,kw_19,0,0,0,0,"{Distribution type} MethodCommands.html#MethodNonD"},
    \{"gen_reliability_levels",14,1,7,0,1137,kw_20,0,0,0,0,"{Generalized reliability levels} MethodCommands.html#MethodNonD"},
    \{"model_pointer",11,0,2,0,1593}\},
    \{"probability_levels",14,1,6,0,1133,kw_21,0,0,0,0,"{Probability levels} MethodCommands.html#MethodNonD"},
    \{"response_levels",14,2,1,0,1059,kw_173\},
    \{"rng",8,2,8,0,1141,kw_22,0,0,0,0,"{Random number generator} MethodCommands.html#MethodNonDMC"},
    \{"samples",9,0,3,0,1361,0,0,0,0,"{Number of samples} MethodCommands.html#MethodNonDMC"\},
    \{"seed",8x19,0,4,0,1363,0,0,0,0,0,"{Refinement seed} MethodCommands.html#MethodNonDLocalRel\}\}

GuiKeyWord kw_175[1] [static]
Initial value:
= {
    \{"num_reliability_levels",13,0,1,0,925,0,0,0,0,0,"{Number of reliability levels} MethodCommands.html#MethodNonD"\}
}

GuiKeyWord kw_176[2] [static]
Initial value:
= {
    \{"parallel",8,0,1,1,943},
    \{"series",8,0,1,1,941\}\}

GuiKeyWord kw_177[4] [static]
Initial value:
= {
    \{"gen_reliabilities",8,0,1,1,937},
    \{"probabilities",8,0,1,1,933},
    \{"reliabilities",8,0,1,1,935},
    \{"system",8,2,2,0,939,kw_176\}\}
GuiKeyWord kw_178[2]  [static]
Initial value:
= {
    {*compute*,8,4,2,0,931,kw_177,0,0,0,0,"{Target statistics for response levels}
    MethodCommands.html#MethodNonD"},
    {*num_response_levels*,13,0,1,0,929,0,0,0,0,0,"{Number of response levels}
    MethodCommands.html#MethodNonD"} }

GuiKeyWord kw_179[2]  [static]
Initial value:
= {
    {*annotated*,8,0,1,0,779},
    {*freeform*,8,0,1,0,781} }

GuiKeyWord kw_180[3]  [static]
Initial value:
= {
    {*adapted*,8,0,1,1,719},
    {*tensor_product*,8,0,1,1,715},
    {*total_order*,8,0,1,1,717} }

GuiKeyWord kw_181[1]  [static]
Initial value:
= {
    {*noise_tolerance*,14,0,1,0,741} }

GuiKeyWord kw_182[1]  [static]
Initial value:
= {
    {*noise_tolerance*,14,0,1,0,745} }

GuiKeyWord kw_183[2]  [static]
Initial value:
= {
    {*l2_penalty*,10,0,2,0,751,0,0,0,0,"{l2_penalty used for elastic net modification of
    LASSO} MethodCommands.html#MethodNonDPCE"},
    {*noise_tolerance*,14,0,1,0,749} }

12.1. DAKOTA NAMESPACE REFERENCE
GuiKeyWord kw_184[2]  [static]
Initial value:
= {
    {"equality_constrained",8,0,1,0,731},
    {"svd",8,0,1,0,729}
}

GuiKeyWord kw_185[1]  [static]
Initial value:
= {
    {"noise_tolerance",14,0,1,0,735}
}

GuiKeyWord kw_186[17]  [static]
Initial value:
= {
    {"basis_pursuit",8,0,2,0,737,0,0,0,0,0,"{L1 minimization via Basis Pursuit (BP)} MethodCommands.html#MethodNonDPCE"},
    {"basis_pursuit_denoising",8,1,2,0,739,kw_181,0,0,0,0,"{L1 minimization via Basis Pursuit DeNoising (BPDN) MethodCommands.html#MethodNonDPCE"},
    {"bp",0,0,2,0,736},
    {"bpdn",0,1,2,0,738,kw_181},
    {"cross_validation",8,0,3,0,753,0,0,0,0,"{Specify whether to use cross validation} MethodCommands.html#MethodNonDPCE"},
    {"lars",0,1,2,0,742,kw_182},
    {"lasso",0,2,2,0,746,kw_183},
    {"least_absolute_shrinkage",8,2,2,0,747,kw_183,0,0,0,0,"{L1 minimization via Least Absolute Shrinkage Operator (LASSO) MethodCommands.html#MethodNonDPCE"},
    {"least_angle_regression",8,1,2,0,743,kw_182,0,0,0,0,0,"{L1 minimization via Least Angle Regression (LARS) MethodCommands.html#MethodNonDPCE"},
    {"least_squares",8,2,2,0,727,kw_184,0,0,0,0,"{Least squares regression} MethodCommands.html#MethodNonDPCE"},
    {"omp",0,1,2,0,732,kw_185},
    {"orthogonal_matching_pursuit",8,1,2,0,733,kw_185,0,0,0,0,"{L1 minimization via Orthogonal Matching Pursuit (OMP) MethodCommands.html#MethodNonDPCE"},
    {"ratio_order",10,0,1,0,725,0,0,0,0,0,"{Order of collocation oversampling relationship} MethodCommands.html#MethodNonDPCE"},
    {"reuse_points",8,0,6,0,759},
    {"reuse_samples",0,0,6,0,758},
    {"tensor_grid",8,0,5,0,757},
    {"use_derivatives",8,0,4,0,755}
}

GuiKeyWord kw_187[3]  [static]
Initial value:
= {
    {"incremental_lhs",8,0,2,0,765,0,0,0,0,0,"{Use incremental LHS for expansion_samples} MethodCommands.html#MethodNonDPCE"},
    {"reuse_points",8,0,1,0,763},
    {"reuse_samples",0,0,1,0,762}
GuiKeyWord kw_188[6] [static]
Initial value:

= {
  {"basis_type",8,3,2,0,713,kw_180},
  {"collocation_points",13,17,3,1,721,kw_186,0.,0.,0.,0,"[Number collocation points to
  estimate coeffs] MethodCommands.html#MethodNonDPCE"},
  {"collocation_ratio",10,17,3,1,723,kw_186,0.,0.,0.,0,"{Collocation point oversampling ratio
  to estimate coeffs} MethodCommands.html#MethodNonDPCE"},
  {"dimension_preference",14,0,1,0,711},
  {"expansion_samples",13,3,3,1,761,kw_187,0.,0.,0.,0,"{Number simulation samples to estimate
  coeffs} MethodCommands.html#MethodNonDPCE"},
  {"import_points_file",11,2,4,0,777,kw_179,0.,0.,0.,0.,0,"{File name for points to be imported
  for forming a PCE (unstructured grid assumed)} MethodCommands.html#MethodNonDPCE"}
}

GuiKeyWord kw_189[2] [static]
Initial value:

= {
  {"annotated",8,0,1,0,815},
  {"freeform",8,0,1,0,817}
}

GuiKeyWord kw_190[6] [static]
Initial value:

= {
  {"collocation_points",13,0,1,1,769},
  {"cross_validation",8,0,2,0,771},
  {"import_points_file",11,2,5,0,777,kw_179,0.,0.,0.,0.,0,"{File name for points to be imported
  for forming a PCE (unstructured grid assumed)} MethodCommands.html#MethodNonDPCE"},
  {"reuse_points",8,0,4,0,775},
  {"reuse_samples",0,0,4,0,774},
  {"tensor_grid",13,0,3,0,773}
}

GuiKeyWord kw_191[3] [static]
Initial value:

= {
  {"decay",8,0,1,1,685},
  {"generalized",8,0,1,1,687},
  {"sobol",8,0,1,1,683}
}

GuiKeyWord kw_192[2] [static]
Initial value:

= {
  {"dimension_adaptive",8,3,1,1,681,kw_191},
  {"uniform",8,0,1,1,678}
}
GuiKeyWord kw_193[4]  [static]
Initial value:

```csharp
= {
    {"adapt_import",8,0,1,1,807},
    {"import",8,0,1,1,805},
    {"mm_adapt_import",8,0,1,1,809},
    {"refinement_samples",9,0,2,0,811,0,0.,0.,0.,0.*Refinement samples
MethodCommands.html#MethodNonDLocalRel"}
}
```

GuiKeyWord kw_194[3]  [static]
Initial value:

```csharp
= {
    {"dimension_preference",14,0,1,0,701,0,0.,0.,0.,0.*Dimension preference for anisotropic
tensor and sparse grids MethodCommands.html#MethodNonDPCE"},
    {"nested",8,0,2,0,703},
    {"non_nested",8,0,2,0,705}
}
```

GuiKeyWord kw_195[2]  [static]
Initial value:

```csharp
= {
    {"lhs",8,0,1,1,799},
    {"random",8,0,1,1,801}
}
```

GuiKeyWord kw_196[5]  [static]
Initial value:

```csharp
= {
    {"dimension_preference",14,0,2,0,701,0,0.,0.,0.,0.*Dimension preference for anisotropic
tensor and sparse grids MethodCommands.html#MethodNonDPCE"},
    {"nested",8,0,3,0,703},
    {"non_nested",8,0,3,0,705},
    {"restricted",8,0,1,0,697},
    {"unrestricted",8,0,1,0,699}
}
```

GuiKeyWord kw_197[2]  [static]
Initial value:

```csharp
= {
    {"drop_tolerance",10,0,2,0,789,0,0.,0.,0.,0.*VBD tolerance for omitting small indices
MethodCommands.html#MethodNonDMC"},
    {"interaction_order",0x19,0,1,0,787,0,0.,0.,0.,0.*Restriction of order of VBD iterations
MethodCommands.html#MethodNonDPCE"}
}
```
GuiKeyWord kw_199[1]  [static]
Initial value:
= {
    {*previous_samples",9,0,1,1,915,0,0.,0.,0.,0,"{Previous samples for incremental approaches}
    MethodCommands.html#MethodNonDMC*}
}

GuiKeyWord kw_200[4]  [static]
Initial value:
= {
    {*incremental.lhs",8,1,1,911,kw_199},
    {*incremental.random",8,1,1,913,kw_199},
    {*lhs",8,0,1,1,909},
    {*random",8,0,1,1,907}
}

GuiKeyWord kw_201[1]  [static]
Initial value:
= {
    {*drop.tolerance",10,0,1,9,919}
}

GuiKeyWord kw_202[12]  [static]
Initial value:
= {
    {*distribution",8,2,6,0,1127,kw_19,0.,0.,0.,0,"{Distribution type}
    MethodCommands.html#MethodNonD"},
    {*fixed_seed",8,0,12,0,921,0.,0.,0.,0,"{Fixed seed flag} MethodCommands.html#MethodNonDMC"},
    {*gen_reliability_levels",14,1,8,0,1137,kw_20,0.,0.,0.,0,"{Generalized reliability levels}
    MethodCommands.html#MethodNonD"},
    {*model_pointer",11,0,3,0,1593},
    {*probability_levels",14,1,7,0,1133,kw_21,0.,0.,0.,0,"{Probability levels}
    MethodCommands.html#MethodNonD"},
    {*reliability_levels",14,1,10,0,923,kw_175,0.,0.,0.,0,"{Reliability levels}
    MethodCommands.html#MethodNonD"},
    {*response_levels",14,2,11,0,927,kw_178,0.,0.,0.,0,"{Response levels}
    MethodCommands.html#MethodNonD"},
    {*rng",8,2,9,0,1141,kw_22,0.,0.,0.,0,"{Random number generator}
    MethodCommands.html#MethodNonDMC"},
    {*sample_type",8,4,1,0,905,kw_260},
    {*samples",9,0,4,0,1361,0.,0.,0.,0,"{Number of samples} MethodCommands.html#MethodNonDMC"},
    {*seed",0x19,0.5,0,1363,0.0.,0.,0.,0,"{Refinement seed}
    MethodCommands.html#MethodNonDLocalRel"},
    {*variance_based_decomp",8,1,2,0,917,kw_201}
}

GuiKeyWord kw_203[2]  [static]
Initial value:
= {
    {*annotated",8,0,1,0,899},
    {*freeform",8,0,1,0,901}
}
GuiKeyWord kw_204[2] [static]
Initial value:

```cpp
= {
    {"generalized",8,0,1,1,841},
    {"sobol",8,0,1,1,839}
}
```

GuiKeyWord kw_205[3] [static]
Initial value:

```cpp
= {
    {"dimension_adaptive",8,2,1,1,837,kw_204},
    {"local_adaptive",8,0,1,1,843},
    {"uniform",8,0,1,1,835}
}
```

GuiKeyWord kw_206[2] [static]
Initial value:

```cpp
= {
    {"generalized",8,0,1,1,831},
    {"sobol",8,0,1,1,829}
}
```

GuiKeyWord kw_207[2] [static]
Initial value:

```cpp
= {
    {"dimension_adaptive",8,2,1,1,827,kw_206},
    {"uniform",8,0,1,1,825}
}
```

GuiKeyWord kw_208[4] [static]
Initial value:

```cpp
= {
    {"adapt_import",8,0,1,1,891},
    {"import",8,0,1,1,889},
    {"mm_adapt_import",8,0,1,1,893},
    {"refinement_samples",9,0,2,0,895}
}
```

GuiKeyWord kw_209[2] [static]
Initial value:

```cpp
= {
    {"lhs",8,0,1,1,883},
    {"random",8,0,1,1,885}
}
```
GuiKeyWord kw_210[4]  [static]
Initial value:
= {
    {"hierarchical",8,0,2,0,861},
    {"nodal",8,0,2,0,859},
    {"restricted",8,0,1,0,855},
    {"unrestricted",8,0,1,0,857}
}

GuiKeyWord kw_211[2]  [static]
Initial value:
= {
    {"drop_tolerance",10,0,2,0,875,0,0.,0.,0.,0.,"VBD tolerance for omitting small indices" MethodCommands.html#MethodNonDSC"},
    {"interaction_order",0x19,0,0.,0.,0.,0.,0.,0.,0.,"Restriction of order of VBD iterations" MethodCommands.html#MethodNonDSC"},
}

GuiKeyWord kw_212[28]  [static]
Initial value:
= {
    {"askey",8,0,2,0,847},
    {"diagonal_covariance",8,0,8,0,877},
    {"dimension_preference",14,0,4,0,863,0,0.,0.,0.,0.,"Dimension preference for anisotropic tensor and sparse grids" MethodCommands.html#MethodNonDSC"},
    {"distribution",8,2,15,0,1127,kw_19,0,0.,0.,0.,"Distribution type" MethodCommands.html#MethodNonDSC"},
    {"drop_tolerance",11,2,11,0,897,kw_203,0,0.,0.,0.,0.,"File name for exporting approximation-based samples from evaluating the interpolant" MethodCommands.html#MethodNonDSC"},
    {"fixed_seed",8,0,21,0,921,0,0.,0.,0.,0.,"Fixed seed flag" MethodCommands.html#MethodNonDMC"},
    {"full_covariance",8,0,8,0,879},
    {"gen_reliability_levels",14,1,17,0,1137,kw_20,0,0.,0.,0.,"Generalized reliability levels" MethodCommands.html#MethodNonDSC"},
    {"h_refinement",8,3,1,0,833,kw_205},
    {"model_pointer",11,0,12,0,1593},
    {"nested",8,0,6,0,867},
    {"non_nested",8,0,6,0,869},
    {"p_refinement",8,2,1,0,823,kw_207},
    {"piecewise",8,0,2,0,845},
    {"probability_levels",14,1,16,0,1133,kw_21,0,0.,0.,0.,"Probability levels" MethodCommands.html#MethodNonDSC"},
    {"probability_refinement",8,4,10,0,887,kw_208},
    {"quadrature_order",13,0,3,1,851,0,0.,0.,0.,"Quadrature order for collocation points" MethodCommands.html#MethodNonDSC"},
    {"reliability_levels",14,1,19,0,923,kw_175,0,0.,0.,0.,"Reliability levels" MethodCommands.html#MethodNonDSC"},
    {"response_levels",14,2,20,0,927,kw_178,0,0.,0.,0.,"Response levels" MethodCommands.html#MethodNonDSC"},
    {"rng",8,2,18,0,1141,kw_22,0,0.,0.,0.,"Random number generator" MethodCommands.html#MethodNonDMC"},
    {"sample_refinement",0,4,10,0,886,kw_208},
    {"sample_type",8,2,9,0,881,kw_209},
    {"samples",9,0,13,0,1361,0,0.,0.,0.,0.,"Number of samples" MethodCommands.html#MethodNonDMC"},
    {"seed",0x19,0,14,0,1363,0,0.,0.,0.,0.,"Refinement seed" MethodCommands.html#MethodNonDLocalRel"},
    {"sparse_grid_level",13,4,3,1,853,kw_210,0,0.,0.,0.,"Sparse grid level for collocation points" MethodCommands.html#MethodNonDSC"},
    {"use_derivatives",8,0,5,0,865,0,0.,0.,0.,0.,"Derivative enhancement flag" MethodCommands.html#MethodNonDSC"},
CHAPTER 12. NAMESPACE DOCUMENTATION

{{"variance_based_decomp",8,2,7,0,871,\\kw211,0.,0.,0.,0.,}{\{Variance-based decomposition (VBD)\}\ MethodCommands.html\#MethodNonDSC"},
{{"wiener",8,0,2,0,849}}}

GuiKeyWord kw.213[2] [static]

Initial value:

= {
    {"misc_options",15,0,1,0,635},
    {"model_pointer",11,0,2,0,1593}
}

GuiKeyWord kw.214[13] [static]

Initial value:

= {
    {"function_precision",10,0,12,0,237,0,0.,0.,0.,0.,}{\{Function precision\}\ MethodCommands.html\#MethodNPSOLDC"},
    {"linear_equality_constraint_matrix",14,0,7,0,465,0,0.,0.,0.,0.,}{\{Linear equality coefficient matrix\}\ MethodCommands.html\#MethodMin"},
    {"linear_equality_scale_types",15,0,9,0,469,0,0.,0.,0.,0.,}{\{Linear equality scaling types\}\ MethodCommands.html\#MethodMin"},
    {"linear_equality_scales",14,0,8,0,467,0,0.,0.,0.,0.,}{\{Linear equality scales\}\ MethodCommands.html\#MethodMin"},
    {"linear_equality_targets",14,0,9,0,469,0,0.,0.,0.,0.,}{\{Linear equality targets\}\ MethodCommands.html\#MethodMin"},
    {"linear_inequality_constraint_matrix",14,0,2,0,455,0,0.,0.,0.,0.,}{\{Linear inequality coefficient matrix\}\ MethodCommands.html\#MethodMin"},
    {"linear_inequality_lower_bounds",14,0,5,0,457,0,0.,0.,0.,0.,}{\{Linear inequality lower bounds\}\ MethodCommands.html\#MethodMin"},
    {"linear_inequality_scale_types",15,0,5,0,461,0,0.,0.,0.,0.,}{\{Linear inequality scaling types\}\ MethodCommands.html\#MethodMin"},
    {"linear_inequality_scales",14,0,6,0,463,0,0.,0.,0.,0.,}{\{Linear inequality scales\}\ MethodCommands.html\#MethodMin"},
    {"linear_inequality_upper_bounds",14,0,4,0,459,0,0.,0.,0.,0.,}{\{Linear inequality upper bounds\}\ MethodCommands.html\#MethodMin"},
    {"linearsearch_tolerance",10,0,13,0,239,0,0.,0.,0.,0.,}{\{Line search tolerance\}\ MethodCommands.html\#MethodNPSOLDC"},
    {"model_pointer",11,0,1,0,1593},
    {"verify_level",9,0,11,0,235,0,0.,0.,0.,0.,}{\{Gradient verification level\}\ MethodCommands.html\#MethodNPSOLDC"}
}

GuiKeyWord kw.215[12] [static]

Initial value:

= {
    {"gradient_tolerance",10,0,12,0,277},
    {"linear_equality_constraint_matrix",14,0,7,0,465,0,0.,0.,0.,0.,}{\{Linear equality coefficient matrix\}\ MethodCommands.html\#MethodMin"},
    {"linear_equality_scale_types",15,0,9,0,469,0,0.,0.,0.,0.,}{\{Linear equality scaling types\}\ MethodCommands.html\#MethodMin"},
    {"linear_equality_scales",14,0,10,0,471,0,0.,0.,0.,0.,}{\{Linear equality scales\}\ MethodCommands.html\#MethodMin"},
    {"linear_equality_targets",14,0,8,0,467,0,0.,0.,0.,0.,}{\{Linear equality targets\}\ MethodCommands.html\#MethodMin"},
    {"linear_inequality_constraint_matrix",14,0,2,0,455,0,0.,0.,0.,0.,}{\{Linear inequality constraint matrix\}\ MethodCommands.html\#MethodMin"},
    {"linear_inequality_lower_bounds",14,0,3,0,457,0,0.,0.,0.,0.,}{\{Linear inequality lower bounds\}\ MethodCommands.html\#MethodMin"},
    {"linear_inequality_scale_types",15,0,5,0,461,0,0.,0.,0.,0.,}{\{Linear inequality scaling types\}\ MethodCommands.html\#MethodMin"},
    {"linear_inequality_scales",14,0,6,0,463,0,0.,0.,0.,0.,}{\{Linear inequality scales\}\ MethodCommands.html\#MethodMin"},
    {"linear_inequality_upper_bounds",14,0,4,0,459,0,0.,0.,0.,0.,}{\{Linear inequality upper bounds\}\ MethodCommands.html\#MethodMin"},
    {"verify_level",9,0,11,0,235,0,0.,0.,0.,0.,}{\{Gradient verification level\}\ MethodCommands.html\#MethodNPSOLDC"}
}
12.1. DAKOTA NAMESPACE REFERENCE

{types} MethodCommands.html#MethodMin",
{"linear.inequality.scales",14,0,6,0,463,0,0..0,0,"{Linear inequality scales}
MethodCommands.html#MethodMin",
{"linear.inequality.upper.bounds",14,0,4,0,459,0,0..0,0,"{Linear inequality upper
bounds} MethodCommands.html#MethodMin",
{"max_step",10,0,11,0,275},
{"model_pointer",11,0,1,0,1593}

GuiKeyWord kw_216[11] [static]

Initial value:

= {
  {"linear.equality.constraint_matrix",14,0,8,0,465,0,0..0,0,"{Linear equality
  coefficient matrix} MethodCommands.html#MethodMin"},
  {"linear.equality.scale_types",15,0,10,0,469,0,0..0,0,"{Linear equality scaling types}
  MethodCommands.html#MethodMin"},
  {"linear.equality.scales",14,0,11,0,471,0,0..0,0,"{Linear equality scales}
  MethodCommands.html#MethodMin"},
  {"linear.equality.targets",14,0,9,0,467,0,0..0,0,"{Linear equality targets}
  MethodCommands.html#MethodMin"},
  {"linear.inequality.constraint_matrix",14,0,3,0,455,0,0..0,0,"{Linear inequality
  coefficient matrix} MethodCommands.html#MethodMin"},
  {"linear.inequality.lower.bounds",14,0,4,0,457,0,0..0,0,"{Linear inequality lower
  bounds} MethodCommands.html#MethodMin"},
  {"linear.inequality.scale_types",15,0,6,0,461,0,0..0,0,"{Linear inequality scaling
  types} MethodCommands.html#MethodMin"},
  {"linear.inequality.scales",14,0,7,0,463,0,0..0,0,"{Linear inequality scales}
  MethodCommands.html#MethodMin"},
  {"linear.inequality.upper.bounds",14,0,5,0,459,0,0..0,0,"{Linear inequality upper
  bounds} MethodCommands.html#MethodMin"},
  {"model_pointer",11,0,2,0,1593},
  {"search.scheme.size",9,0,1,0,281}
}

GuiKeyWord kw_217[3] [static]

Initial value:

= {
  {"argaez_tapia",8,0,1,1,267},
  {"el_bakry",8,0,1,1,265},
  {"van_shanno",8,0,1,1,269}
}

GuiKeyWord kw_218[4] [static]

Initial value:

= {
  {"gradient_based.line_search",8,0,1,1,257,0,0..0,0,"{CHOOSE line search type}"},
  {"trgds",8,0,1,1,261},
  {"trust_region",8,0,1,1,259},
  {"value_based.line_search",8,0,1,1,255}
}
**Chapter 12. Namespace Documentation**

GuiKeyWord kw_219[16] [static]

Initial value:

```cpp
= {
    "centering-parameter", 10, 0, 4, 0.273,
    "gradient_tolerance", 10, 0, 16, 0.277,
    "linear_equality_constraint_matrix", 14, 0, 11, 0, 465, 0, 0, 0, 0, 0, 0 /* Linear equality
        coefficient matrix */
    MethodCommands.html#MethodMin",
    "linear_equality_scale_types", 15, 0, 13, 0, 469, 0, 0, 0, 0, 0 /* Linear equality scaling types */
    MethodCommands.html#MethodMin",
    "linear_equality_scales", 14, 0, 14, 0, 471, 0, 0, 0, 0, 0 /* Linear equality scales */
    MethodCommands.html#MethodMin",
    "linear_inequality_constraint_matrix", 14, 0, 6, 0, 455, 0, 0, 0, 0, 0 /* Linear inequality
        coefficient matrix */
    MethodCommands.html#MethodMin",
    "linear_inequality_scale_types", 15, 0, 9, 0, 461, 0, 0, 0, 0, 0 /* Linear inequality scaling
        types */
    MethodCommands.html#MethodMin",
    "linear_inequality_scales", 14, 0, 10, 0, 463, 0, 0, 0, 0, 0 /* Linear inequality scales */
    MethodCommands.html#MethodMin",
    "linear_inequality_upper_bounds", 14, 0, 8, 0, 459, 0, 0, 0, 0, 0 /* Linear inequality upper
        bounds */
    MethodCommands.html#MethodMin",
    "max_step", 10, 0, 15, 0, 275,
    "merit_function", 8, 3, 2, 0, 263, kw_217",
    "model_pointer", 11, 0, 5, 0, 1093,
    "search_method", 8, 4, 1, 0, 253, kw_218,
    "steplength_to_boundary", 10, 0, 3, 0, 271
}
```

GuiKeyWord kw_220[5] [static]

Initial value:

```cpp
= {
    "debug", 8, 0, 1, 1, 23, 0, 0, 0, 0, 0, 0 /* Choose output level */
    "normal", 8, 0, 1, 1, 27,
    "quiet", 8, 0, 1, 1, 29,
    "silent", 8, 0, 1, 1, 31,
    "verbose", 8, 0, 1, 1, 25
}
```

GuiKeyWord kw_221[2] [static]

Initial value:

```cpp
= {
    "model_pointer", 11, 0, 1, 5, 99, 0, 0, 0, 0, 0, 0 /* Identification of model by pointer */
    MethodCommands.html#MethodMetaMultiStart",
    "opt_model_pointer", 3, 0, 1, 0, 99
}
```

GuiKeyWord kw_222[1] [static]

Initial value:

```cpp
= {
    "seed", 9, 0, 1, 0, 105, 0, 0, 0, 0, 0, 0 /* Seed for random weighting sets */
    MethodCommands.html#MethodMetaParetoSet",
}
```
12.1. DAKOTA NAMESPACE REFERENCE

GuiKeyWord kw_223[10] [static]
Initial value:

= 
{
  "iterator_scheduling", 8,2,5,0,111,kw_120,0,0,0,0,"{Message passing configuration for scheduling of iterator jobs} MethodCommands.html#MethodMeta",
  "iterator_servers", 0x19,0,4,0,109,0,0,0,0,"{Number of iterator servers} MethodCommands.html#MethodMeta",
  "method_name", 11,2,1,1,97,kw_221,0,0,0,0,"{Identification of sub-iterator by name} MethodCommands.html#MethodMetaMultiStart",
  "method_pointer", 11,0,1,1,101,0,0,0,0,"{Identification of sub-iterator by pointer} MethodCommands.html#MethodMetaMultiStart",
  "multi_objective_weight_sets", 6,0,3,0,106,
  "opt_method_name", 3,2,1,1,96,kw_221,
  "opt_method_pointer", 3,0,1,1,100,
  "processors_per_iterator", 0x19,0,6,0,117,0,0,0,0,0,"{Number of processors per iterator server} MethodCommands.html#MethodMeta",
  "random_weight_sets", 9,1,2,0,103,kw_222,0,0,0,0,"{Number of random weighting sets} MethodCommands.html#MethodMetaParetoSet",
  "weight_sets", 14,0,3,0,107,0,0,0,0,0,"{List of user-specified weighting sets} MethodCommands.html#MethodMetaParetoSet"}
}

GuiKeyWord kw_224[4] [static]
Initial value:

= 
{
  "model_pointer", 11,0,2,0,1593,
  "partitions", 13,0,1,0,1359,0,0,0,0,"{Number of partitions} MethodCommands.html#MethodPSUADE",
  "samples", 9,0,3,0,1361,0,0,0,0,"{Number of samples} MethodCommands.html#MethodNonDMC",
  "seed", 0x19,0,4,0,1363,0,0,0,0,0,"{Refinement seed} MethodCommands.html#MethodNonDLocalRel"}
}

GuiKeyWord kw_225[5] [static]
Initial value:

= 
{
  "converge_order", 8,0,1,1,1587,
  "converge_qoi", 8,0,1,1,1589,
  "estimate_order", 8,0,1,1,1585,
  "model_pointer", 11,0,3,0,1593,
  "refinement_rate", 10,0,2,0,1591,0,0,0,0,"{Refinement rate} MethodCommands.html#MethodSolnRichardson"}
}

GuiKeyWord kw_226[2] [static]
Initial value:

= 
{
  "num_generations", 0x29,0,2,0,405,
  "percent_change", 10,0,1,5,403}
}
GuiKeyWord kw_227[2] [static]
Initial value:
= {
    "num
    generations",0x29,0,2,0,399,0,0,0,0,0,"{Number of generations for convergence
    test} MethodCommands.html#MethodJEGASOGA"},
    "percent
    change",10,0,1,0,397,0,0,0,0,0,"{Percent change in fitness
    MethodCommands.html#MethodJEGASOGA}"
}

GuiKeyWord kw_228[2] [static]
Initial value:
= {
    "average
    fitness
    tracker",8,2,1,1,401,kw_226},
    "best
    fitness
    tracker",8,2,1,1,395,kw_227
}

GuiKeyWord kw_229[2] [static]
Initial value:
= {
    "constraint
    penalty",10,0,2,0,381,0,0,0,0,0,"{Constraint penalty in merit function
    MethodCommands.html#MethodJEGASOGA}"},
    "merit
    function",8,0,1,1,379
}

GuiKeyWord kw_230[4] [static]
Initial value:
= {
    "elitist",8,0,1,1,385},
    "favor
    feasible",8,0,1,1,387},
    "roulette
    wheel",8,0,1,1,389},
    "unique
    roulette
    wheel",8,0,1,1,391
}

GuiKeyWord kw_231[20] [static]
Initial value:
= {
    "convergence
    type",8,2,3,0,393,kw_228,0,0,0,0,0,"{Convergence type
    MethodCommands.html#MethodJEGASOGA}"},
    "crossover
    type",8,5,18,0,421,kw_153,0,0,0,0,0,"{Crossover type
    MethodCommands.html#MethodJEGADC}"},
    "fitness
    type",8,2,1,0,377,kw_228,0,0,0,0,0,"{Fitness type
    MethodCommands.html#MethodJEGASOGA}"},
    "initialization
    type",8,3,17,0,413,kw_153,0,0,0,0,0,"{Initialization type
    MethodCommands.html#MethodJEGADC}"},
    "linear
    equality
    constraint
    matrix",14,0,10,0,465,0,0,0,0,0,"{Linear equality
    coefficient matrix} MethodCommands.html#MethodMin"},
    "linear
    equality
    scaling
    types",15,0,12,0,469,0,0,0,0,0,"{Linear equality scaling types
    MethodCommands.html#MethodMin}"},
    "linear
    equality
    scales",14,0,13,0,471,0,0,0,0,0,"{Linear equality scales
    MethodCommands.html#MethodMin}"},
    "linear
    equality
    targets",14,0,11,0,467,0,0,0,0,0,"{Linear equality targets
    MethodCommands.html#MethodMin}"
}
12.1. DAKOTA NAMESPACE REFERENCE

{"linear.inequality.constraint_matrix",14,0,5,0,455,0,0,0,0,0,"[Linear inequality coefficient matrix] MethodCommands.html#MethodMin"},
{"linear.inequality.constraint_matrix",14,0,6,0,457,0,0,0,0,0,"[Linear inequality lower bounds] MethodCommands.html#MethodMin"},
{"linear.inequality.constraint_matrix",14,0,7,0,459,0,0,0,0,0,"[Linear inequality upper bounds] MethodCommands.html#MethodMin"},
{"linear.inequality.constraint_matrix",14,0,8,0,461,0,0,0,0,0,"[Linear inequality scaling types] MethodCommands.html#MethodMin"},
{"linear.inequality.constraint_matrix",14,0,9,0,463,0,0,0,0,0,"[Linear inequality scales] MethodCommands.html#MethodMin"},
{"linear.inequality.upper.bounds",14,0,7,0,459,0,0,0,0,0,"[Linear inequality upper bounds] MethodCommands.html#MethodMin"},
{"log.file",11,0,15,0,409,0,0,0,0,0,"[Log file] MethodCommands.html#MethodJEGADC"},
{"model.pointer",11,0,4,0,1593},
{"mutation.type",8,6,19,0,437,kw156,0.,0.,0.,0,"[Mutation type] MethodCommands.html#MethodJEGADC"},
{"population.size",0x29,0,14,0,407,0,0,0,0,0,"[Number of population members] MethodCommands.html#MethodJEGADC"},
{"print.each.pop",8,0,16,0,411,0,0,0,0,0,"[Population output] MethodCommands.html#MethodJEGADC"},
{"replacement.type",8,4,2,0,383,kw230,0.,0.,0.,0,"[Replacement type] MethodCommands.html#MethodJEGASOGA"},
{"seed",0x19,0,20,0,453,0,0,0,0,0,"[Random seed] MethodCommands.html#MethodJEGADC"} }

GuiKeyWord kw_232[15] [static]

Initial value:

= {
  {"function.precision",10,0,13,0,237,0,0,0,0,0,"[Function precision] MethodCommands.html#MethodNPSOLDC"},
  {"linear.inequality.constraint_matrix",14,0,8,0,465,0,0,0,0,0,"[Linear equality coefficient matrix] MethodCommands.html#MethodMin"},
  {"linear.inequality.constraint_matrix",14,0,10,0,469,0,0,0,0,0,"[Linear inequality lower bounds] MethodCommands.html#MethodMin"},
  {"linear.inequality.constraint_matrix",14,0,11,0,471,0,0,0,0,0,"[Linear inequality scales] MethodCommands.html#MethodMin"},
  {"linear.inequality.constraint_matrix",14,0,12,0,473,0,0,0,0,0,"[Linear inequality targets] MethodCommands.html#MethodMin"},
  {"linear.inequality.constraint_matrix",14,0,13,0,237,0,0,0,0,0,"[Linear equality coefficient matrix] Method Commands.html#MethodMin"},
  {"linear.inequality.constraint_matrix",14,0,14,0,455,0,0,0,0,0,"[Linear inequality lower bounds] MethodCommands.html#MethodMin"},
  {"linear.inequality.constraint_matrix",14,0,15,0,461,0,0,0,0,0,"[Linear inequality scaling types] MethodCommands.html#MethodMin"},
  {"linear.inequality.constraint_matrix",14,0,16,0,463,0,0,0,0,0,"[Linear inequality scales] MethodCommands.html#MethodMin"},
  {"linear.inequality.constraint_matrix",14,0,17,0,459,0,0,0,0,0,"[Linear inequality upper bounds] MethodCommands.html#MethodMin"},
  {"linesearch.tolerance",10,0,14,0,239,0,0,0,0,0,"[Line search tolerance] MethodCommands.html#MethodNPSOLDC"},
  {"model.pointer",11,0,2,0,1593},
  {"nlopt",3,0,1,1,233},
  {"npsol",8,0,1,1,231},
  {"verify.level",9,0,12,0,235,0,0,0,0,0,"[Gradient verification level] MethodCommands.html#MethodNPSOLDC"} }

GuiKeyWord kw_233[7] [static]

Initial value:

= {
  {"approx.method.name",3,0,1,1,184},
  {"approx.method.pointer",3,0,1,1,182},
  {"approx.model.pointer",3,0,2,2,186},
  {"method.name",11,0,1,1,185},
  {"method.pointer",11,0,1,1,183},
  {"model.pointer",11,0,2,2,187}.
CHAPTER 12. NAMESPACE DOCUMENTATION

GuiKeyWord kw_234[2] [static]
Initial value:
= {
  "filter", 8, 0, 1, 1, 175, 0, 0, 0, 0, 0, 0, "@\text{CHOOSE acceptance logic}"},
  "tr.ratio", 8, 0, 1, 1, 173
}

GuiKeyWord kw_235[7] [static]
Initial value:
= {
  "augmented_lagrangian_objective", 8, 0, 1, 1, 151, 0, 0, 0, 0, 0, 0, "@\text{CHOOSE objective formulation}",
  "lagrangian_objective", 8, 0, 1, 1, 153},
  "linearized_constraints", 8, 0, 2, 2, 157, 0, 0, 0, 0, 0, 0, "@\text{CHOOSE constraint formulation}"},
  "no_constraints", 8, 0, 2, 2, 159},
  "original_constraints", 8, 0, 2, 2, 155, 0, 0, 0, 0, 0, 0, "@"},
  "original_primary", 8, 0, 1, 1, 147, 0, 0, 0, 0, 0, 0, "@"},
  "single_objective", 8, 0, 1, 1, 149
}

GuiKeyWord kw_236[1] [static]
Initial value:
= {
  "homotopy", 8, 0, 1, 1, 179
}

GuiKeyWord kw_237[4] [static]
Initial value:
= {
  "adaptive_penalty_merit", 8, 0, 1, 1, 165, 0, 0, 0, 0, 0, 0, "@\text{CHOOSE merit function}"},
  "augmented_lagrangian_merit", 8, 0, 1, 1, 169, 0, 0, 0, 0, 0, 0, "@"},
  "lagrangian_merit", 8, 0, 1, 1, 167},
  "penalty_merit", 8, 0, 1, 1, 163
}

GuiKeyWord kw_238[6] [static]
Initial value:
= {
  "contract_threshold", 10, 0, 3, 0, 0, 137, 0, 0, 0, 0, 0, 0, "@\text{Shrink trust region if trust region ratio is below this value} MethodCommands.html#MethodSBL"},
  "contraction_factor", 10, 0, 3, 0, 0, 141, 0, 0, 0, 0, 0, 0, "@\text{Trust region contraction factor} MethodCommands.html#MethodSBL"},
  "expand_threshold", 10, 0, 4, 0, 0, 139, 0, 0, 0, 0, 0, 0, "@\text{Expand trust region if trust region ratio is above this value} MethodCommands.html#MethodSBL"},
  "expansion_factor", 10, 0, 6, 0, 0, 143, 0, 0, 0, 0, 0, 0, "@\text{Trust region expansion factor}
12.1. DAKOTA NAMESPACE REFERENCE

MethodCommands.html#MethodSBL*

{initial_size*,10,0,1,0,133,0,0,0,0,0,0,"{Trust region initial size (relative to bounds}
MethodCommands.html#MethodSBL*

{minimum_size*,10,0,2,0,135,0,0,0,0,0,0,"{Trust region minimum size
MethodCommands.html#MethodSBL*}

GuiKeyWord kw_239[13] [static]

Initial value:

= {
    {"acceptance_logic",0,2,8,0,171,kw_234,0,0,0,0,"{SBL iterate acceptance logic
MethodCommands.html#MethodSBL*},
    {"approx_method_name",3,0,1,1,122},
    {"approx_method_pointer",3,0,1,1,120},
    {"approx_subproblem",8,7,6,0,145,kw_235,0,0,0,0,0,"{Approximate subproblem formulation
MethodCommands.html#MethodSBL*},
    {"constraint_relax",8,1,9,0,177,kw_236,0,0,0,0,0,"{SBL constraint relaxation method for
infeasible iterates} MethodCommands.html#MethodSBL*},
    {"merit_function",8,4,7,0,161,kw_237,0,0,0,0,0,"{SBL merit function
MethodCommands.html#MethodSBL*},
    {"method_name",11,0,1,1,123,0,0,0,0,0,"{Identification of minimizer by name
MethodCommands.html#MethodMetaParetoSet"},
    {"method_pointer",11,0,1,1,121,0,0,0,0,0,"{Identification of minimizer by pointer
MethodCommands.html#MethodMetaParetoSet"},
    {"model_pointer",11,0,2,2,125,0,0,0,0,0,"{Identification of model by pointer
MethodCommands.html#MethodMetaParetoSet"},
    {"soft_convergence_limit",9,0,3,0,127,0,0,0,0,0,0,"{Soft convergence limit for SBL
iterations} MethodCommands.html#MethodSBL*},
    {"trust_region",8,6,5,0,131,kw_238,0,0,0,0,0,"{Trust region group specification
MethodCommands.html#MethodSBL*},
    {"truth_surrogate_bypass",8,0,4,0,129,0,0,0,0,0,0,"{Flag for bypassing lower level
surrogates in truth verifications} MethodCommands.html#MethodSBL*}
}

GuiKeyWord kw_240[4] [static]

Initial value:

= {
    {"final_point",14,0,1,1,1557,0,0,0,0,0,"{CHOOSE final pt or increment}{Termination point
of vector} MethodCommands.html#MethodPSVPS"},
    {"model_pointer",11,0,3,0,1593},
    {"num_steps",9,0,2,2,1561,0,0,0,0,0,"{Number of steps along vector
MethodCommands.html#MethodPSVPS"},
    {"step_vector",14,0,1,1,1559,0,0,0,0,0,"{Step vector} MethodCommands.html#MethodPSVPS"}
}

GuiKeyWord kw_242[1] [static]

Initial value:

= {
    {"optional_interface_responses_pointer",11,0,1,0,1799,0,0,0,0,0,"{Responses pointer for
nested model optional interfaces} ModelCommands.html#ModelNested"}
}
GuiKeyWord kw_243[2] [static]
Initial value:
= {
    {"master",8,0,1,1,1807},
    {"peer",8,0,1,1,1809}
}

GuiKeyWord kw_244[7] [static]
Initial value:
= {
    {"iterator_scheduling",8,2,2,0,1805,kw_243},
    {"iterator_servers",0x19,0,1,0,1803},
    {"primary_response_mapping",14,0,6,0,1817,0,0.,0.,0.,0,0,"{Primary response mappings for nested models} ModelCommands.html#ModelNested"},
    {"primary_variable_mapping",15,0,4,0,1813,0,0.,0.,0.,0,"{Primary variable mappings for nested models} ModelCommands.html#ModelNested"},
    {"processors_per_iterator",0x19,0,3,0,1811},
    {"secondary_response_mapping",14,0,7,0,1819,0,0.,0.,0.,0,"{Secondary response mappings for nested models} ModelCommands.html#ModelNested"},
    {"secondary_variable_mapping",15,0,5,0,1815,0,0.,0.,0.,0,"{Secondary variable mappings for nested models} ModelCommands.html#ModelNested"}
}

GuiKeyWord kw_245[2] [static]
Initial value:
= {
    {"optional_interface_pointer",11,1,1,0,1797,kw_242,0,0.,0.,0.,0,"{Optional interface set pointer} ModelCommands.html#ModelNested"},
    {"sub_method_pointer",11,7,2,1,1801,kw_244,0,0.,0.,0,"{Sub-method pointer for nested models} ModelCommands.html#ModelNested"}
}

GuiKeyWord kw_246[1] [static]
Initial value:
= {
    {"interface_pointer",11,0,1,0,1607,0,0.,0.,0.,0,"{Interface set pointer} ModelCommands.html#ModelSingle"}
}

GuiKeyWord kw_247[2] [static]
Initial value:
= {
    {"annotated",8,0,1,0,1761},
    {"freeform",8,0,1,0,1763}
}
GuiKeyWord kw_248[6] [static]
Initial value:
= {
    "additive",8,0,2,2,1743,0,0,0,0,"[CHOOSE correction type]",
    "combined",8,0,2,2,1747,
    "first_order",8,0,1,1,1739,0,0,0,0,"[CHOOSE correction order]",
    "multiplicative",8,0,2,2,1745,
    "second_order",8,0,1,1,1741,
    "zeroth_order",8,0,1,1,1737
}

GuiKeyWord kw_249[2] [static]
Initial value:
= {
    "folds",9,0,1,0,1753,0,0,0,0,"[Number cross validation folds]
    ModelCommands.html#ModelSurrG",
    "percent",10,0,1,0,1755,0,0,0,0,"[Percent points per CV fold]
    ModelCommands.html#ModelSurrG"
}

GuiKeyWord kw_250[2] [static]
Initial value:
= {
    "cross_validate",8,2,1,0,1751,kw_249,
    "press",8,0,2,0,1757,0,0,0,0,"[Perform PRESS cross validation]
    ModelCommands.html#ModelSurrG"
}

GuiKeyWord kw_251[2] [static]
Initial value:
= {
    "annotated",8,0,1,0,1729,
    "freeform",8,0,1,0,1731
}

GuiKeyWord kw_252[3] [static]
Initial value:
= {
    "constant",8,0,1,1,1623,
    "linear",8,0,1,1,1625,
    "reduced_quadratic",8,0,1,1,1627
}

GuiKeyWord kw_253[2] [static]
Initial value:
= {
    "point_selection",8,0,1,0,1619,0,0,0,0,"[GP point selection]
    ModelCommands.html#ModelSurrG",
    "trend",8,3,2,0,1621,kw_252,0,0,0,0,"[GP trend function] ModelCommands.html#ModelSurrG"
}
GuiKeyWord kw.254[4] [static]
Initial value:
= {
    {"constant", 8, 0, 1, 1, 1633},
    {"linear", 8, 0, 1, 1635},
    {"quadratic", 8, 0, 1, 1639},
    {"reduced_quadratic", 8, 0, 1, 1637}
}

GuiKeyWord kw.255[7] [static]
Initial value:
= {
    {"correlation_lengths", 14, 0, 5, 0, 1649, 0, 0, 0, 0, 0},
    {"export_model_file", 11, 0, 6, 0, 1651},
    {"find_nugget", 9, 0, 4, 0, 1647, 0, 0, 0, 0, 0},
    {"max_trials", 11, 0, 3, 0, 1643, 0, 0, 0, 0, 0},
    {"optimization_method", 11, 0, 2, 0, 1641, 0, 0, 0, 0, 0},
    {"trend", 8, 4, 1, 0, 1631, kw.254[4], 0, 0, 0, 0, 0}
}

GuiKeyWord kw.256[2] [static]
Initial value:
= {
    {"dakota", 8, 2, 1, 1, 1617, kw.253},
    {"surfpack", 8, 7, 1, 1, 1629, kw.253}
}

GuiKeyWord kw.257[2] [static]
Initial value:
= {
    {"annotated", 8, 0, 1, 0, 1723, 0, 0, 0, 0, 0},
    {"freeform", 8, 0, 1, 0, 1725, 0, 0, 0, 0, 0}
}

GuiKeyWord kw.258[2] [static]
Initial value:
= {
    {"cubic", 8, 0, 1, 1, 1661},
    {"linear", 8, 0, 1, 1, 1659}
}
GuiKeyWord kw_259[3] [static]

Initial value:

= {
    "export_model_file", 11, 0, 3, 0, 1663,
    "interpolation", 8, 2, 2, 0, 1657, kw_258, 0, 0, 0, 0,"{MARS interpolation} ModelCommands.html#ModelSurrG",
    "max_bases", 9, 0, 1, 0, 1655, 0, 0, 0, 0,"{MARS maximum bases} ModelCommands.html#ModelSurrG"
}

GuiKeyWord kw_260[3] [static]

Initial value:

= {
    "export_model_file", 11, 0, 3, 0, 1671,
    "poly_order", 9, 0, 1, 0, 1667, 0, 0, 0, 0,"{MLS polynomial order} ModelCommands.html#ModelSurrG",
    "weight_function", 9, 0, 2, 0, 1669, 0, 0, 0, 0,"{MLS weight function} ModelCommands.html#ModelSurrG"
}

GuiKeyWord kw_261[4] [static]

Initial value:

= {
    "export_model_file", 11, 0, 4, 0, 1681,
    "nodes", 9, 0, 1, 0, 1675, 0, 0, 0, 0,"{ANN number nodes} ModelCommands.html#ModelSurrG",
    "random_weight", 9, 0, 3, 0, 1679, 0, 0, 0, 0,"{ANN random weight} ModelCommands.html#ModelSurrG",
    "range", 10, 0, 2, 0, 1677, 0, 0, 0, 0,"{ANN range} ModelCommands.html#ModelSurrG"
}

GuiKeyWord kw_262[4] [static]

Initial value:

= {
    "cubic", 8, 0, 1, 1, 1701, 0, 0, 0, 0,"{CHOSE polynomial order}"",
    "export_model_file", 11, 0, 2, 0, 1703,
    "linear", 8, 0, 1, 1, 1697,
    "quadratic", 8, 0, 1, 1, 1699
}

GuiKeyWord kw_263[5] [static]

Initial value:

= {
    "bases", 9, 0, 1, 0, 1685, 0, 0, 0, 0,"{RBF number of bases} ModelCommands.html#ModelSurrG",
    "export_model_file", 11, 0, 5, 0, 1693,
    "max_pts", 9, 0, 2, 0, 1687, 0, 0, 0, 0,"{RBF maximum points} ModelCommands.html#ModelSurrG",
    "max_subsets", 9, 0, 4, 0, 1691,
    "min_partition", 9, 0, 3, 0, 1689, 0, 0, 0, 0,"{RBF minimum partitions} ModelCommands.html#ModelSurrG"
}
GuiKeyWord kw_264[3] [static]
Initial value:

```c
= {
    {"all",8,0,1,1,1715},
    {"none",8,0,1,1,1719},
    {"region",8,0,1,1,1717}
}
```

GuiKeyWord kw_265[21] [static]
Initial value:

```c
= {
    {"challenge_points_file",11,2,10,0,1759,kw_247,0.,0.,0.,0.,"[Challenge file for surrogate
metrics] ModelCommands.html#ModelSurrG"},
    {"correction",8,6,8,0,1735,kw_247,0.,0.,0.,0.,"[Surrogate correction approach]
ModelCommands.html#ModelSurrG"},
    {"dace.method_pointer",11,0,3,0,1711,0,0.,0.,0.,"[Design of experiments method pointer]
ModelCommands.html#ModelSurrG"},
    {"gaussian_process",8,2,1,1,1615,kw_256,0.,0.,0.,0.,"[CHOOSE surrogate type] Dakota Gaussian
process] ModelCommands.html#ModelSurrG"},
    {"import_points_file",11,2,5,0,1721,kw_257,0.,0.,0.,0.,"[File import of samples for global
approximation builds] ModelCommands.html#ModelSurrG"},
    {"kriging",0,2,1,1,1664,kw_256},
    {"metrics",15,2,9,0,1749,kw_257,0.,0.,0.,0.,"[Compute surrogate diagnostics]
ModelCommands.html#ModelSurrG"},
    {"moving_least_squares",8,3,1,1,1665,kw_260,0.,0.,0.,0.,"[Moving least squares]
ModelCommands.html#ModelSurrG"},
    {"neural_network",8,4,1,1,1673,kw_261,0.,0.,0.,0.,"[Artificial neural network]
ModelCommands.html#ModelSurrG"},
    {"radial_basis",8,3,1,1,1683,kw_263},
    {"reduced_points",8,0,2,0,1709},
    {"reuse_points",8,3,4,0,1713,kw_264},
    {"reuse_samples",0,3,4,0,1712,kw_264},
    {"samples_file",3,2,5,0,1720,kw_257},
    {"total_points",9,0,2,0,1705},
    {"use_derivatives",8,0,7,0,1733,0,0.,0.,0.,0.,"[Surfpack GP gradient enhancement]
ModelCommands.html#ModelSurrG"}
}
```

GuiKeyWord kw_266[6] [static]
Initial value:

```c
= {
    {"additive",8,0,2,2,1789,0,0.,0.,0.,0.,"[CHOOSE correction type]"},
    {"combined",8,0,2,2,1793},
    {"first_order",8,0,1,1,1785,0,0.,0.,0.,0.,"[CHOOSE correction order]"},
    {"multiplicative",8,0,2,2,1791},
    {"second_order",8,0,1,1,1787},
    {"zeroth_order",8,0,1,1,1783}
}
```
12.1. DAKOTA NAMESPACE REFERENCE

GuiKeyWord kw\_267[3] [static]

Initial value:

```c
= {
    "correction",8,6,3,1781,kw\_266,0.,0.,0.,0.,"{Surrogate correction approach}
    ModelCommands.html#ModelSurrH"),
    "high\_fidelity\_model\_pointer",11,0,2,2,1779,0.,0.,0.,0.,"{Pointer to the high fidelity
    model specification} ModelCommands.html#ModelSurrH"),
    "low\_fidelity\_model\_pointer",11,0,1,1,1777,0.,0.,0.,0.,"{Pointer to the low fidelity
    model specification} ModelCommands.html#ModelSurrH"
}
```

GuiKeyWord kw\_268[2] [static]

Initial value:

```c
= {
    "actual\_model\_pointer",11,0,2,2,1773,0.,0.,0.,0.,0.,"{Pointer to the truth model
    specification} ModelCommands.html#ModelSurrMP"),
    "taylor\_series",8,0,1,1,1771,0.,0.,0.,0.,0.,"{Taylor series local approximation }
    ModelCommands.html#ModelSurrL"
}
```

GuiKeyWord kw\_269[2] [static]

Initial value:

```c
= {
    "actual\_model\_pointer",11,0,2,2,1773,0.,0.,0.,0.,0.,"{Pointer to the truth model
    specification} ModelCommands.html#ModelSurrMP"),
    "tana",8,0,1,1,1767,0.,0.,0.,0.,0.,"{Two-point adaptive nonlinear approximation }
    ModelCommands.html#ModelSurrMP"
}
```

GuiKeyWord kw\_270[5] [static]

Initial value:

```c
= {
    "global",8,2,1,2,1,1613,kw\_265,0.,0.,0.,0.,"{CHOOSE surrogate category}{Global
    approximations } ModelCommands.html#ModelSurrG"),
    "hierarchical",8,3,2,1,1775,kw\_267,0.,0.,0.,0.,"{Hierarchical approximation } 
    ModelCommands.html#ModelSurrH"),
    "id\_surrogates",13,0,1,0,1611,0.,0.,0.,0.,0.,"{Surrogate response ids}
    ModelCommands.html#ModelSurrogate"),
    "local",8,2,2,1,1765,kw\_268,0.,0.,0.,0.,"{Local approximation}
    ModelCommands.html#ModelSurrL"),
    "multipoint",8,2,2,1,1765,kw\_269,0.,0.,0.,0.,0.,"{Multipoint approximation}
    ModelCommands.html#ModelSurrMP"
}
```

GuiKeyWord kw\_271[7] [static]

Initial value:
= {
  "hierarchical_tagging", 8, 0, 4, 0, 1603, 0, 0, 0, 0, 0, 0, "Hierarchical evaluation tags"
}
ModelCommands.html#ModelIndControl",
  "id_model", 11, 0, 1, 0, 1597, 0, 0, 0, 0, 0, 0, "Model set identifier"
ModelCommands.html#ModelIndControl",
  "nested", 8, 2, 5, 1, 1795, 0, 0, 0, 0, 0, 0, "Choose model type"
ModelCommands.html#ModelIndControl",
  "responses_pointer", 11, 0, 3, 0, 1601, 0, 0, 0, 0, 0, 0, "Responses set pointer"
ModelCommands.html#ModelIndControl",
  "single", 8, 1, 5, 1, 1605, 0, 0, 0, 0, 0, 0, "@

GuiKeyWord kw_272[6] [static]
Initial value:

= {
  "annotated", 8, 0, 3, 0, 2349, 0, 0, 0, 0, 0, 0, "Data file in annotated format"
RespCommands.html#RespFnLS",
  "freeform", 8, 0, 3, 0, 2351, 0, 0, 0, 0, 0, 0, "Data file in freeform format"
RespCommands.html#RespFnLS",
  "num_config_variables", 0x29, 0, 4, 0, 2353, 0, 0, 0, 0, 0, 0, "Configuration variable columns in file"
RespCommands.html#RespFnLS",
  "num_experiments", 0x29, 0, 1, 0, 2345, 0, 0, 0, 0, 0, 0, "Experiments in file"
RespCommands.html#RespFnLS",
  "num_replicates", 13, 0, 2, 0, 2347, 0, 0, 0, 0, 0, 0, "Replicates per each experiment in file"
RespCommands.html#RespFnLS",
  "num_std_deviations", 0x29, 0, 5, 0, 2355, 0, 0, 0, 0, 0, 0, "Standard deviation columns in file"
RespCommands.html#RespFnLS"
}

GuiKeyWord kw_273[6] [static]
Initial value:

= {
  "nonlinear_equality_scale_types", 0x807, 0, 2, 0, 2370, 0, 0, 0, 0, 0, 0, "Nonlinear equality constraints"
nonlinear_equality_constraints",
  "nonlinear_equality_scales", 0x806, 0, 3, 0, 2372, 0, 0, 0, 0, 0, 0, "Nonlinear equality constraints"
nonlinear_equality_constraints",
  "nonlinear_equality_targets", 6, 0, 1, 0, 2368, 0, 0, 0, 0, 0, 0, "Nonlinear equality targets"
nonlinear_equality_constraints",
  "scale_types", 0x805, 0, 2, 0, 2371, 0, 0, 0, 0, 0, 0, "Nonlinear inequality constraints"
nonlinear_inequality_constraints",
  "scales", 0x806, 0, 3, 0, 2373, 0, 0, 0, 0, 0, 0, "Nonlinear inequality constraints"
nonlinear_inequality_constraints",
  "targets", 14, 0, 1, 0, 2369, 0, 0, 0, 0, 0, 0, "Nonlinear targets"
nonlinear_inequality_constraints",
}

GuiKeyWord kw_274[8] [static]
Initial value:

= {
  "lower_bounds", 14, 0, 1, 0, 2359, 0, 0, 0, 0, 0, 0, "Nonlinear inequality lower bounds"
nonlinear_inequality_constraints",
  "nonlinear_inequality_lower_bounds", 6, 0, 1, 0, 2358, 0, 0, 0, 0, 0, 0, "Nonlinear inequality constraints"
nonlinear_inequality_constraints",
  "nonlinear_inequality_scale_types", 0x807, 0, 3, 0, 2362, 0, 0, 0, 0, 0, 0, "Nonlinear inequality constraints"
nonlinear_inequality_constraints",
  "nonlinear_inequality_scales", 0x806, 0, 4, 0, 2364, 0, 0, 0, 0, 0, 0, "Nonlinear inequality constraints"
nonlinear_inequality_constraints",
  "nonlinear_inequality_upper_bounds", 6, 0, 2, 0, 2360, 0, 0, 0, 0, 0, 0, "Nonlinear inequality constraints"}
GuiKeyWord kw_275[15] [static]

Initial value:
= {
  {"calibration_data_file",11,6,4,0,2343,kw_272,0.,0.,0.,0,0,0,"{Calibration data file name} RespCommands.html#RespFnLS"},
  {"calibration_term_scale_types",0x807,0,1,0,2336,0,0.,0.,0.,0,"{calibration terms}"},
  {"calibration_term_scales",0x806,0,2,0,2338,0,0.,0.,0.,0,"{calibration terms}"},
  {"least_squares_data_file",3,6,4,0,2342,kw_272,0.,0.,0.,0,"{Calibration data file name} RespCommands.html#RespFnLS"},
  {"least_squares_term_scale_types",0x807,0,1,0,2336,0,0.,0.,0.,0,"{calibration terms}"},
  {"least_squares_term_scales",0x806,0,2,0,2338,0,0.,0.,0.,0,"{calibration terms}"},
  {"least_squares_weights",6,0,3,0,2340,0,0.,0.,0.,0,"{calibration term weights} RespCommands.html#RespFnLS"},
  {"nonlinear_inequality_constraints",0x29,6,6,0,2363,kw_273,0.,0.,0.,0,"{Number of nonlinear equality constraints} RespCommands.html#RespFnLS"},
  {"nonlinear_inequality_constraints",0x29,8,5,0,2357,kw_274,0.,0.,0.,0,"{Number of nonlinear inequality constraints} RespCommands.html#RespFnLS"},
  {"num_nonlinear_inequality_constraints",0x21,6,6,0,2366,kw_273},
  {"nonlinear_inequality_constraints",0x21,8,5,0,2356,kw_274},
  {"primary_scale_types",0x80f,0,1,0,2337,0,0.,0.,0.,0,"{Calibration scaling types} RespCommands.html#RespFnLS"},
  {"primary_scales",0x80e,0,2,0,2339,0,0.,0.,0.,0,"{Calibration scales} RespCommands.html#RespFnLS"},
  {"weights",14,0,3,0,2341,0,0.,0.,0.,0,"{Calibration term weights} RespCommands.html#RespFnLS"},
  {"num_nonlinear_inequality_constraints",0x29,6,6,0,2363,kw_273},
  {"least_squares_weights",6,0,3,0,2340,0,0.,0.,0.,0,"{calibration term weights} RespCommands.html#RespFnLS"}
}

GuiKeyWord kw_276[4] [static]

Initial value:
= {
  {"absolute",8,0,2,0,2397},
  {"bounds",8,0,2,0,2399},
  {"ignore_bounds",8,0,1,0,2393,0,0.,0.,0.,0,"{Ignore variable bounds} RespCommands.html#RespGradMixed"},
  {"relative",8,0,2,0,2395}
}

GuiKeyWord kw_277[10] [static]

Initial value:
= {
  {"central",8,0,6,0,2404,0,0.,0.,0.,0,"{CHOOSE difference interval}"},
  {"dakota",8,0,6,0,2401,kw_276,0,0.,0.,0.,0,"{CHOOSE gradient source} Interval scaling type RespCommands.html#RespGradMixed"},
  {"fdgradient_step_size",6,0,7,0,2408},
  {"fd_step_size",14,0,7,0,2409,0,0.,0.,0.,0,"{Finite difference step size} RespCommands.html#RespGradMixed"},
  {"forward",8,0,6,0,2405,0,0.,0.,0.,0,"{Method source} RespCommands.html#RespGradMixed"},
  {"idanalyticgradients",13,0,2,2,2385,0,0.,0.,0.,0,"{Analytic derivatives function list} RespCommands.html#RespGradMixed"},
  {"idnumericalgradients",13,0,1,1,2383,0,0.,0.,0.,0,"{Numerical derivatives function list} RespCommands.html#RespGradMixed"},
  {"idintervaltype",8,0,5,0,2403,0,0.,0.,0.,0,"{Interval type} RespCommands.html#RespGradMixed"},
  {"method_source",8,0,3,0,2389,0,0.,0.,0.,0,"{Method source} RespCommands.html#RespGradMixed"},
  {"vendor",8,0,4,0,2401}
}
CHAPTER 12. NAMESPACE DOCUMENTATION

GuiKeyWord kw_278[2] [static]
Initial value:
= {
    "fd_hessian_step_size", 6, 0, 1, 0, 2440,
    "fd_step_size", 14, 0, 1, 0, 2441, 0, 0, 0, 0, 0,
    "Finite difference step size"
}

GuiKeyWord kw_279[1] [static]
Initial value:
= {
    "damped", 8, 0, 1, 0, 2457, 0, 0, 0, 0, 0,
    "Numerical safeguarding of BFGS update"
}

GuiKeyWord kw_280[2] [static]
Initial value:
= {
    "bfgs", 8, 1, 1, 1, 2455, 0, 0, 0, 0, 0,
    "sr1", 8, 0, 1, 1, 2449, 0, 0, 0, 0, 0,
    "Central difference interval"
}

GuiKeyWord kw_281[8] [static]
Initial value:
= {
    "absolute", 8, 0, 2, 0, 2445,
    "bounds", 8, 0, 2, 0, 2447,
    "central", 8, 0, 3, 0, 2451, 0, 0, 0, 0, 0,
    "forward", 8, 0, 3, 0, 2449, 0, 0, 0, 0, 0,
    "relative", 8, 0, 2, 0, 2443,
}

GuiKeyWord kw_282[6] [static]
Initial value:
= {
    "nonlinear_equality_scale_types", 0x807, 0, 2, 0, 2330, 0, 0, 0, 0, 0,
    "nonlinear_equality_scales", 0x806, 0, 3, 0, 2332, 0, 0, 0, 0, 0,
    "nonlinear_equality_constraints",
    "nonlinear_equality_targets", 6, 0, 1, 1, 2328, 0, 0, 0, 0, 0,
    "Nonlinear scaling types (for inequalities or equalities)"
}

RespCommands.html#RespHessMixed
GuiKeyWord kw_283[8] [static]
Initial value:

```csharp
= {
    {"lower_bounds", 14, 0, 1, 0, 2319, 0, 0, 0, 0, 0, 0}, /* Nonlinear inequality constraint lower bounds */
    RespCommands.html#RespFnOpt*, 0, "nonlinear_inequality_constraints"},
    {"upper_bounds", 14, 0, 1, 0, 2319, 0, 0, 0, 0, 0, 0}, /* Nonlinear inequality constraint upper bounds */
    nonl_inequality_constraints*,
    } nonl_inequality_constraints*,

    {"nonlinear_inequality_scale_types", 0x807, 0, 3, 0, 2322, 0, 0, 0, 0, 0, 0}, /* Nonlinear inequality constraints */
    {"nonlinear_inequality_scales", 0x806, 0, 4, 0, 2324, 0, 0, 0, 0, 0, 0}, /* Nonlinear inequality constraints */
    {"scale_types", 0x80f, 0, 3, 0, 2323, 0, 0, 0, 0, 0, 0}, /* Nonlinear inequality constraints */
    {"scales", 0x80e, 0, 4, 0, 2325, 0, 0, 0, 0, 0, 0}, /* Nonlinear inequality constraints */
    {"primary_scale_types", 0x80f, 0, 2, 0, 2405, 0, 0, 0, 0, 0, 0}, /* Objective function scaling types */
    {"primary_scales", 0x80e, 0, 3, 0, 2403, 0, 0, 0, 0, 0, 0}, /* Objective function scaling types */
    {"method_source", 8, 0, 3, 0, 2409, 0, 0, 0, 0, 0, 0}, /* Method source */
    {"vendor", 8, 0, 4, 0, 2401} }
```

GuiKeyWord kw_284[11] [static]
Initial value:

```csharp
= {
    {"multi_objective_weights", 6, 0, 4, 0, 2314, 0, 0, 0, 0, 0, 0}, /* Objective functions */
    {"nonlinear_inequality_constraints", 0x29, 6, 6, 0, 2327, kw_283, 0, 0, 0, 0}, /* Number of nonlinear inequality constraints */
    {"num_nonlinear_inequality_constraints", 0x21, 8, 5, 0, 2317, kw_283, 0, 0, 0, 0, 0, 0}, /* Number of nonlinear inequality constraints */
    {"objective_function_scale_types", 0x807, 0, 2, 0, 2326, 0, 0, 0, 0, 0, 0}, /* Objective function scaling types */
    {"objective_function_scales", 0x806, 0, 3, 0, 2318, 0, 0, 0, 0, 0, 0}, /* Objective function scales */
    {"sense", 0x80f, 0, 1, 0, 2309, 0, 0, 0, 0, 0, 0}, /* Optimization sense */
    {"weights", 14, 0, 4, 0, 2315, 0, 0, 0, 0, 0, 0} /* Multi-objective weightings */
    } nonl_inequality_constraints*,
```

GuiKeyWord kw_285[8] [static]
Initial value:

```csharp
= {
    {"central", 8, 0, 6, 0, 2407, 0, 0, 0, 0, 0, 0}, /* [CHOOSE difference interval] */
    {"dakota", 8, 4, 0, 2391, kw_283, 0, 0, 0, 0}, /* [CHOOSE gradient source] */
    {"method_source", 8, 0, 3, 0, 2389, 0, 0, 0, 0, 0, 0}, /* Method source */
    {"vendor", 8, 0, 4, 0, 2401} }
```
GuiKeyWord kw_286[7] [static]

Initial value:
= {
  {"absolute",8,0,2,0,2419},
  {"bounds",8,0,2,0,2421},
  {"central",8,0,3,0,2425,0,0,0,0,0,"[CHOOSE difference interval]",
   {"fd_hessian_step_size",6,0,1,0,2414},
   {"fd_step_size",14,0,1,0,2415,0,0,0,0,0,"[Finite difference step size]
   RespCommands.html#RespHessNum"},
   {"forward",8,0,3,0,2423,0,0,0,0,0,*"},
   {"relative",8,0,2,0,2417}
}

GuiKeyWord kw_287[1] [static]

Initial value:
= {
  {"damped",8,0,1,0,2431,0,0,0,0,0,"[Numerical safeguarding of BFGS update]
   RespCommands.html#RespHessQuasi"}
}

GuiKeyWord kw_288[2] [static]

Initial value:
= {
  {"bfgs",8,1,1,2429,kw_287,0,0,0,0,"[CHOOSE Hessian approx."]},
  {"sr1",8,0,1,2433}
}

GuiKeyWord kw_289[19] [static]

Initial value:
= {
  {"analytic_gradients",8,0,4,2,2379,0,0,0,0,0,"[CHOOSE gradient type]"},
  {"analytic_hessians",8,0,5,3,2435,0,0,0,0,0,"[CHOOSE Hessian type]"},
  {"calibration_terms",0x29,15,3,1,2335,kw_275,0,0,0,0,0,"[Calibration (Least squares)]
   Number of calibration terms] RespCommands.html#RespFnLS"},
  {"descriptors",15,0,2,0,2405,0,0,0,0,0,"[Response labels] RespCommands.html#RespLabels"},
  {"id_responses",11,0,1,2303,0,0,0,0,0,"[Responses set identifier]
   RespCommands.html#RespSetId"},
  {"least_squares_terms",0x21,15,3,1,2334,kw_275},
  {"mixed_gradients",8,10,4,2,2381,kw_277,0,0,0,0,0,"[Mixed gradients]
   RespCommands.html#RespGradMixed"},
  {"mixed_hessians",8,0,5,3,2437,kw_281,0,0,0,0,0,"[Mixed Hessians]
   RespCommands.html#RespGradMixed"},
  {"no_gradients",8,0,4,2,2377,0,0,0,0,0,"[Generic responses]
   RespCommands.html#RespGradNum"},
  {"num_least_squares_terms",0x21,15,3,1,2334,kw_275},
  {"no_hessians",8,0,5,3,2441,0,0,0,0,0,"[Generic Hessians]
   RespCommands.html#RespGradNum"},
  {"num_objective_functions",0x21,15,3,1,2306,kw_284},
  {"num_response_functions",0x21,0,3,1,2374},
  {"numerical_gradients",8,0,4,2,2387,kw_285,0,0,0,0,0,"[Generic gradients]
   RespCommands.html#RespGradNum"},
  {"quasi_hessians",8,2,5,3,2427,kw_288,0,0,0,0,0,"[Quasi Hessians]
   RespCommands.html#RespGradQuasi"},
  {"objective_functions",0x29,11,3,1,2307,kw_284,0,0,0,0,0,"[Optimization Number of
   objective functions] RespCommands.html#RespFnOpt"},
  {"response_descriptors",7,0,2,0,2304},
  {"response_functions",0x29,0,3,1,2375,0,0,0,0,0,"[Generic responses] Number of response
   functions] RespCommands.html#RespFnGen"}
GuiKeyWord kw_290[6] [static]
Initial value:

= {
    "aleatory",8,0,1,1,1833],
    "all",8,0,1,1,1827],
    "design",8,0,1,1,1829],
    "epistemic",8,0,1,1,1835],
    "state",8,0,1,1,1837],
    "uncertain",8,0,1,1,1831]
}

GuiKeyWord kw_291[11] [static]
Initial value:

= {
    "alphas",14,0,1,1,1963,0,0,0,0,0,"{beta uncertain alphas} VarCommands.html#VarCAUV_Beta
",0,"beta uncertain"},
    "betas",14,0,2,2,1965,0,0,0,0,0,"{beta uncertain betas} VarCommands.html#VarCAUV_Beta",0,"beta uncertain"},
    "buv_alphas",6,0,1,1,1962,0,0,0,0,0,0,"beta uncertain"},
    "buv_betas",6,0,2,2,1964,0,0,0,0,0,0,"beta uncertain"},
    "buv_descriptors",7,0,6,0,1972,0,0,0,0,0,0,"beta uncertain"},
    "buv_lower_bounds",6,0,3,3,1966,0,0,0,0,0,0,"beta uncertain"},
    "buv_upper_bounds",6,0,4,4,1968,0,0,0,0,0,0,"beta uncertain"},
    "descriptors",15,0,6,0,1973,0,0,0,0,0,0,"{Descriptors} VarCommands.html#VarCAUV_Beta",0,"beta uncertain"},
    "initial_point",14,0,5,0,1971,0,0,0,0,0,0,"beta uncertain"},
    "lower_bounds",14,0,3,3,1967,0,0,0,0,0,0,"{Distribution lower bounds} VarCommands.html#VarCAUV_Beta",0,"beta uncertain"},
    "upper_bounds",14,0,4,4,1965,0,0,0,0,0,0,"{Distribution upper bounds} VarCommands.html#VarCAUV_Beta",0,"beta uncertain"}
}

GuiKeyWord kw_292[5] [static]
Initial value:

= {
    "descriptors",15,0,4,0,2045,0,0,0,0,0,"{Descriptors} VarCommands.html#VarDAUV_Binomial
",0,"binomial uncertain"},
    "initial_point",13,0,3,0,2043,0,0,0,0,0,0,"binomial uncertain"},
    "num_trials",13,0,2,2,2041,0,0,0,0,0,0,"{binomial uncertain num trials} VarCommands.html#VarDAUV_Binomial",0,"binomial uncertain"},
    "prob_per_trial",6,0,1,1,2038,0,0,0,0,0,0,0,"{binomial uncertain"},
    "probability_per_trial",14,0,1,1,2039,0,0,0,0,0,0,0,"binomial uncertain"}
}

GuiKeyWord kw_293[12] [static]
Initial value:

= {
    "cdv_descriptors",7,0,6,0,1854,0,0,0,0,0,0,"continuous_design"},
    "cdv_initial_point",6,0,1,0,1844,0,0,0,0,0,0,"continuous_design"},
    "cdv_lower_bounds",6,0,2,0,1846,0,0,0,0,0,0,"continuous_design"},
    "cdv_scale_types",0x807,0,4,0,1850,0,0,0,0,0,0,"continuous_design"},
    "cdv_scales",0x806,0,5,0,1852,0,0,0,0,0,0,"continuous_design"},
    "cdv_upper_bounds",6,0,3,0,1848,0,0,0,0,0,0,"continuous_design"},
    "descriptors",15,0,6,0,1855,0,0,0,0,0,0,"{Descriptors} VarCommands.html#VarCDV",0,"continuous_design"},
    "initial_point",14,0,1,0,1845,0,0,0,0,0,0,"{Initial point} VarCommands.html#VarCDV",0,"continuous_design"}
continuous_design",
{"lower_bounds":14,0,2,0,1847,0,0,0,0,0,0,"{Lower bounds} VarCommands.html#VarCDV",0,"
continuous_design"},
{"scale_types":0x80f,0,4,0,1851,0,0,0,0,0,0,"{Scales} VarCommands.html#VarCDV",0,"
continuous_design"},
{"scales":0x80e,0,5,0,1853,0,0,0,0,0,0,"{Scales} VarCommands.html#VarCDV",0,"
continuous_design"},
{"upper_bounds":14,0,3,0,1849,0,0,0,0,0,0,"{Upper bounds} VarCommands.html#VarCDV",0,"
continuous_design"}]

GuiKeyWord kw_294(10) [static]

Initial value:

= {
  {"descriptors":15,0,6,0,2103,0,0,0,0,0,0,"{Descriptors} VarCommands.html#VarCEUV_Interval",0,"continuous_interval_uncertain"},
  {"interval_probabilities":14,0,2,0,2095,0,0,0,0,0,0,"{Interval probabilities} VarCommands.html#VarCEUV_Interval"},
  {"interval_probs":6,0,2,0,2094},
  {"iuv_descriptors":7,0,6,0,2102,0,0,0,0,0,0,"{IUV descriptors} VarCommands.html#VarCEUV_Interval"},
  {"iuv_num_intervals":5,0,1,0,2092,0,0,0,0,0,0,"{IUV intervals} VarCommands.html#VarCEUV_Interval"},
  {"lower_bounds":14,0,3,1,2097,0,0,0,0,0,0,"{Lower bounds of continuous intervals} VarCommands.html#VarCEUV_Interval"},
  {"num_intervals":13,0,1,0,2093,0,0,0,0,0,0,"{Number of intervals defined for each continuous interval variable} VarCommands.html#VarCEUV_Interval"},
  {"upper_bounds":14,0,4,2,2099,0,0,0,0,0,0,"{Upper bounds of continuous intervals} VarCommands.html#VarCEUV_Interval"}
}

GuiKeyWord kw_295(8) [static]

Initial value:

= {
  {"csv_descriptors":7,0,4,0,2150,0,0,0,0,0,0,"{CSV descriptors} VarCommands.html#VarCSV",0,"continuous_state"},
  {"csv_initial_state":6,0,1,0,2144,0,0,0,0,0,0,"{CSV initial state} VarCommands.html#VarCSV"},
  {"csv_lower_bounds":6,0,2,0,2146,0,0,0,0,0,0,"{CSV lower bounds} VarCommands.html#VarCSV"},
  {"csv_upper_bounds":6,0,3,0,2148,0,0,0,0,0,0,"{CSV upper bounds} VarCommands.html#VarCSV"},
  {"descriptors":15,0,4,0,2151,0,0,0,0,0,0,"{Descriptors} VarCommands.html#VarCSV",0,"continuous_state"},
  {"initial_state":14,0,1,0,2145,0,0,0,0,0,0,"{Initial state} VarCommands.html#VarCSV"},
  {"lower_bounds":14,0,2,0,2147,0,0,0,0,0,0,"{Lower bounds} VarCommands.html#VarCSV"},
  {"num_intervals":13,0,1,0,2093,0,0,0,0,0,0,"{Number of intervals defined for each continuous interval variable} VarCommands.html#VarCSV"},
  {"upper_bounds":14,0,4,2,2099,0,0,0,0,0,0,"{Upper bounds of continuous intervals} VarCommands.html#VarCSV"}
}

GuiKeyWord kw_296(8) [static]

Initial value:

= {
  {"ddv_descriptors":7,0,4,0,1864,0,0,0,0,0,0,"{DDV descriptors} VarCommands.html#VarDDRIV"},
  {"ddv_initial_state":6,0,1,0,1859,0,0,0,0,0,0,"{DDV initial state} VarCommands.html#VarDDRIV"},
  {"ddv_lower_bounds":5,0,2,0,1866,0,0,0,0,0,0,"{DDV lower bounds} VarCommands.html#VarDDRIV"},
  {"ddv_upper_bounds":5,0,3,0,1862,0,0,0,0,0,0,"{DDV upper bounds} VarCommands.html#VarDDRIV"},
  {"descriptors":15,0,4,0,1865,0,0,0,0,0,0,"{Descriptors} VarCommands.html#VarDDRIV",0," discrete_design_range"},
  {"initial_point":13,0,1,0,1859,0,0,0,0,0,0,"{Initial point} VarCommands.html#VarDDRIV",0," discrete_design_range"},
  {"lower_bounds":14,0,2,0,1847,0,0,0,0,0,0,"{Lower bounds} VarCommands.html#VarDDRIV"},
  {"scale_types":0x80f,0,4,0,1851,0,0,0,0,0,0,"{Scales} VarCommands.html#VarDDRIV"},
  {"scales":0x80e,0,5,0,1853,0,0,0,0,0,0,"{Scales} Var Commands.html#VarDDRIV"},
  {"upper_bounds":14,0,3,0,1849,0,0,0,0,0,0,"{Upper bounds} VarCommands.html#VarDDRIV"}
12.1. DAKOTA NAMESPACE REFERENCE

```json
{
  "lower_bounds",13,0,2,0,1861,0,0.,0.,0.,0.,{"Lower bounds VarCommands.html#VarDDRIV",0,*
  discrete.design_range"},
  "upper_bounds",13,0,3,0,1863,0,0.,0.,0.,0.,{"Upper bounds VarCommands.html#VarDDRIV",0,*
  discrete.design_range"}
}
```

GuiKeyWord kw_297[4] [static]
Initial value:

```json
= {
  "descriptors",15,0,4,0,1875,0,0.,0.,0.,0.,{"Descriptors VarCommands.html#VarDDSIV",0,*
  discrete.design_set_integer"},
  "initial_point",13,0,1,0,1869,0,0.,0.,0.,0.,{"Initial point VarCommands.html#VarDDSIV",0,*
  discrete.design_set_integer"},
  "num_set_values",13,0,2,0,1871,0,0.,0.,0.,0.,{"Number of values for each variable
  VarCommands.html#VarDDSIV",0,"discrete.design_set_integer"},
  "set_values",13,0,3,1,1873,0,0.,0.,0.,0.,{"Set values VarCommands.html#VarDDSIV"}
}
```

GuiKeyWord kw_298[4] [static]
Initial value:

```json
= {
  "descriptors",15,0,4,0,1887,0,0.,0.,0.,0.,{"Descriptors VarCommands.html#VarDDSRV",0,*
  discrete.design_set_real"},
  "initial_point",14,0,1,0,1879,0,0.,0.,0.,0.,{"Initial point VarCommands.html#VarDDSRV",0,*
  discrete.design_set_real"},
  "num_set_values",13,0,2,0,1881,0,0.,0.,0.,0.,{"Number of values for each variable
  VarCommands.html#VarDDSRV",0,"discrete.design_set_real"},
  "set_values",14,0,3,1,1883,0,0.,0.,0.,0.,{"Set values VarCommands.html#VarDDSRV"}
}
```

GuiKeyWord kw_299[9] [static]
Initial value:

```json
= {
  "descriptors",15,0,6,0,2117,0,0.,0.,0.,0.,{"Descriptors VarCommands.html#VarDIUV",0,*
  discrete.interval_uncertain"},
  "initial_point",13,0,5,0,2115,0,0.,0.,0.,0.,0.,0.,0.,{"Discrete.interval_uncertain
  VarCommands.html#VarDIUV"},
  "interval_probabilities",14,0,2,0,2109,0,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,{Basic probability assignments per
  interval VarCommands.html#VarDIUV"},
  "interval_probs",6,0,2,0,2108,
  "lower_bounds",13,0,3,1,2111,0,0.,0.,0.,0.,{"Lower bounds VarCommands.html#VarDIUV"},
  "num_intervals",13,0,1,0,2107,0,0.,0.,0.,0.,0.,{Number of intervals defined for each interval
  variable VarCommands.html#VarDIUV"},
  "range_probabilities",6,0,2,0,2108,
  "range_probs",6,0,2,0,2108,
  "upper_bounds",13,0,4,2,2113,0,0.,0.,0.,0.,{"Upper bounds VarCommands.html#VarDIUV"}
}
```

GuiKeyWord kw_300[8] [static]
Initial value:

```json
= {
  "descriptors",15,0,4,0,2161,0,0.,0.,0.,0.,{"Descriptors VarCommands.html#VarDSRIV",0,*
  discrete.state_range"},
  "dsv_descriptors",7,0,4,0,2160,0,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,"discrete.state_range"},
  "dsv_initial_state",5,0,1,0,2154,0,0.,0.,0.,0.,0.,0.,0.,0.,"discrete.state_range"},
```
CHAPTER 12. NAMESPACE DOCUMENTATION

**GuiKeyWord kw_301** [static]

Initial value:

```plaintext
= {
  "descriptors", 15, 0, 4, 0, 2171, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,"descriptors" VarCommands.html#VarDSSIV", 0,"discrete_state_set_integer"],
  "initial_state", 13, 0, 1, 0, 2165, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,"initial_state" VarCommands.html#VarDSSIV", 0,"discrete_state_set_integer"],
  "num_set_values", 13, 0, 2, 0, 2167, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,"num_set_values" VarCommands.html#VarDSSIV", 0,"discrete_state_set_integer"],
  "set_values", 13, 0, 3, 1, 2169, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,"set_values" VarCommands.html#VarDSSIV"}
```

**GuiKeyWord kw_302** [static]

Initial value:

```plaintext
= {
  "descriptors", 15, 0, 4, 0, 2181, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,"descriptors" VarCommands.html#VarDSSRV", 0,"discrete_state_set_real"],
  "initial_state", 14, 0, 1, 0, 2175, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,"initial_state" VarCommands.html#VarDSSRV", 0,"discrete_state_set_real"],
  "num_set_values", 13, 0, 2, 0, 2177, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,"num_set_values" VarCommands.html#VarDSSRV", 0,"discrete_state_set_real"],
  "set_values", 14, 0, 3, 1, 2179, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,"set_values" VarCommands.html#VarDSSRV"}
```

**GuiKeyWord kw_303** [static]

Initial value:

```plaintext
= {
  "descriptors", 15, 0, 5, 0, 2129, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,"descriptors" VarCommands.html#VarDUSIV", 0,"discrete_uncertain_set_integer"],
  "initial_point", 13, 0, 4, 0, 2127, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,"initial_point" VarCommands.html#VarDUSIV", 0,"discrete_uncertain_set_integer"],
  "num_set_values", 13, 0, 1, 0, 2121, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,"num_set_values" VarCommands.html#VarDUSIV", 0,"discrete_uncertain_set_integer"],
  "set_probabilities", 14, 0, 3, 0, 2125, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,"set_probabilities" VarCommands.html#VarDUSIV", 0,"discrete_uncertain_set_integer"],
  "set_probs", 6, 0, 3, 0, 2124,"set_probs" VarCommands.html#VarDUSIV"],
  "set_values", 13, 0, 2, 1, 2123, 0, 0, 0, 0, 0, 0,"set_values" VarCommands.html#VarDUSIV"}
```

**GuiKeyWord kw_304** [static]

Initial value:
12.1. DAKOTA NAMESPACE REFERENCE

```plaintext
= {
  {"descriptors",15,0,5,0,2141,0,0,0,0,0,0,"{Descriptors} VarCommands.html#VarDUSRV",0,"discrete.uncertain.set_real"},
  {"initial_point",14,0,4,0,2139,0,0,0,0,0,0,"discrete.uncertain.set_real"},
  {"num_set_values",13,0,1,0,2133,0,0,0,0,0,0,"{Number of values for each variable} VarCommands.html#VarDUSRV"},
  {"set_probabilities",14,0,3,0,2137,0,0,0,0,0,0,"{Probabilities for each set member} VarCommands.html#VarDUSRV"},
  {"set_probs",6,0,3,0,2136},
  {"set_values",14,0,2,1,2135,0,0,0,0,0,0,"{Set values} VarCommands.html#VarDUSRV"}
}

GuiKeyWord kw_305[5] [static]
Initial value:

```plaintext
= {
  {"betas",14,0,1,1,1955,0,0,0,0,0,0,"{exponential uncertain betas} VarCommands.html#VarCAUV,Exponential",0,"exponential.uncertain"},
  {"descriptors",15,0,3,0,1959,0,0,0,0,0,0,"{Descriptors} VarCommands.html#VarCAUV,Exponential",0,"exponential.uncertain"},
  {"euv_betas",6,0,1,1,1954,0,0,0,0,0,0,"{exponential uncertain betas} VarCommands.html#VarCAUV,Exponential",0,"exponential.uncertain"},
  {"euv_descriptors",7,0,3,0,1958,0,0,0,0,0,0,"{Descriptors} VarCommands.html#VarCAUV,Exponential"},
  {"initial_point",14,0,2,0,1957,0,0,0,0,0,0,"{initial_point} VarCommands.html#VarCAUV,Exponential"}
}

GuiKeyWord kw_306[7] [static]
Initial value:

```plaintext
= {
  {"alphas",14,0,1,1,1997,0,0,0,0,0,0,"{frechet uncertain alphas} VarCommands.html#VarCAUV,Frechet",0,"frechet.uncertain"},
  {"betas",14,0,2,2,1999,0,0,0,0,0,0,"{frechet uncertain betas} VarCommands.html#VarCAUV,Frechet",0,"frechet.uncertain"},
  {"descriptors",15,0,4,0,2003,0,0,0,0,0,0,"{Descriptors} VarCommands.html#VarCAUV,Frechet",0,"frechet.uncertain"},
  {"fuv_alphas",6,0,1,1,1996,0,0,0,0,0,0,"{frechet uncertain alphas} VarCommands.html#VarCAUV,Frechet",0,"frechet.uncertain"},
  {"fuv_betas",6,0,2,2,1998,0,0,0,0,0,0,"{frechet uncertain betas} VarCommands.html#VarCAUV,Frechet",0,"frechet.uncertain"},
  {"fuv_descriptors",7,0,4,0,2002,0,0,0,0,0,0,"{Descriptors} VarCommands.html#VarCAUV,Frechet"},
  {"initial_point",14,0,3,0,2001,0,0,0,0,0,0,"{initial_point} VarCommands.html#VarCAUV,Frechet"}
}

GuiKeyWord kw_307[7] [static]
Initial value:

```plaintext
= {
  {"alphas",14,0,1,1,1977,0,0,0,0,0,0,"{gamma uncertain alphas} VarCommands.html#VarCAUV,Gamma",0,"gamma.uncertain"},
  {"betas",14,0,2,2,1979,0,0,0,0,0,0,"{gamma uncertain betas} VarCommands.html#VarCAUV,Gamma",0,"gamma.uncertain"},
  {"descriptors",15,0,4,0,1983,0,0,0,0,0,0,"{Descriptors} VarCommands.html#VarCAUV,Gamma",0,"gamma.uncertain"},
  {"gauv_alphas",6,0,1,1,1976,0,0,0,0,0,0,"{gamma uncertain alphas} VarCommands.html#VarCAUV,Gamma",0,"gamma.uncertain"},
  {"gauv_betas",6,0,2,2,1978,0,0,0,0,0,0,"{gamma uncertain betas} VarCommands.html#VarCAUV,Gamma",0,"gamma.uncertain"},
  {"gauv_descriptors",7,0,4,0,1982,0,0,0,0,0,0,"{Descriptors} VarCommands.html#VarCAUV,Gamma"},
  {"initial_point",14,0,3,0,1981,0,0,0,0,0,0,"{initial_point} VarCommands.html#VarCAUV,Gamma"}
}
GuiKeyword kw_308[4] [static]
Initial value:
= {
  /*descriptors*/,15,0,3,0,2063,0,0.,0.,0.,0,"geometric_uncertain"},
  /*probability_per_trial*/14,0,1,1,2059,0,0.,0.,0.,0,0,"geometric_uncertain"}
}

GuiKeyword kw_309[7] [static]
Initial value:
= {
  {"alphas",14,0,1,1,1987,0,0.,0.,0.,0,"gumbel_uncertain alphas"},
  {"betas",14,0,2,2,1989,0,0.,0.,0.,0,"gumbel_uncertain betas"},
  {"descriptors",15,0,4,0,1993,0,0.,0.,0.,0,"gumbel_uncertain"},
  {"guuv_alphas",6,0,1,1,1986,0,0.,0.,0.,0,"gumbel_uncertain"},
  {"guuv_betas",6,0,2,2,1988,0,0.,0.,0.,0,"gumbel_uncertain"},
  {"guuv_descriptors",7,0,4,0,1992,0,0.,0.,0.,0,"gumbel_uncertain"},
  {"initial_point",14,0,3,0,1991,0,0.,0.,0,0,0,"gumbel_uncertain"}
}

GuiKeyword kw_310[11] [static]
Initial value:
= {
  {"abscissas",14,0,2,1,2019,0,0.,0.,0.,0,"sets of abscissas for bin-based histogram variables"},
  {"counts",14,0,3,2,2023,0,0.,0.,0.,0,"sets of counts for bin-based histogram variables"},
  {"descriptors",15,0,5,0,2027,0,0.,0.,0.,0,"gumbel_uncertain"},
  {"huv_bin_abscissas",6,0,2,1,2018},
  {"huv_bin_counts",6,0,3,2,2022},
  {"huv_bin_descriptors",7,0,5,0,2026,0,0.,0.,0.,0,"gumbel_uncertain"},
  {"huv_bin_initial_point",14,0,3,0,1991,0,0.,0.,0,0,0,"gumbel_uncertain"},
  {"huv_num_point_pairs",5,0,1,0,2016,0,0.,0.,0.,0,"gumbel_uncertain"},
  {"huv_num_pairs",13,0,1,0,2017,0,0.,0.,0,0,0,"key to apportionment among bin-based histogram variables"},
  {"ordinates",14,0,3,2,2021,0,0.,0.,0.,0,"sets of ordinates for bin-based histogram variables"},
}

GuiKeyword kw_311[9] [static]
Initial value:
= {
  {"abscissas",14,0,2,1,2081,0,0.,0.,0,0,"sets of abscissas for point-based histogram variables"},
  {"counts",14,0,3,2,2083,0,0.,0.,0.,0,"sets of counts for point-based histogram variables"},
  {"descriptors",15,0,5,0,2087,0,0.,0.,0.,0,"gumbel_uncertain"},
  {"guuv_num_point_pairs",5,0,1,0,2018,0,0.,0.,0,0,0,"gumbel_uncertain"},
}
GuiKeyWord kw_312[5] [static]
Initial value:

= {
    {descriptors*,15,0,5,0,2075,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,"Descriptors"},
    {initial_point*,14,0,4,0,2073,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,"Initialize uncertain"},
    {num_drawn*,13,0,3,0,2071,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,"Hypergeometric uncertain num drawn"},
    {selected_population*,13,0,2,0,2069,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,"Hypergeometric uncertain selected population"},
    {total_population*,13,0,1,0,2067,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,"Hypergeometric uncertain total population"}
}

GuiKeyWord kw_313[2] [static]
Initial value:

= {
    {lnuv_zetas*,6,0,1,1,1904,0,0,0,0,0,0,0,0,0,0,0,0,0,0,"Lognormal uncertain zetas"},
    {zetas*,14,0,1,1,1905,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,"Lognormal uncertain standard deviations"}
}

GuiKeyWord kw_314[4] [static]
Initial value:

= {
    {error_factors*,14,0,1,1,1911,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,"Choose variance spec."},
    {lnuv_error_factors*,6,0,1,1,1910,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,"Lognormal uncertain error factors"},
    {lnuv_std_deviations*,6,0,1,1,1908,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,"Lognormal uncertain standard deviations"},
    {std_deviations*,14,0,1,1,1909,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,"Lognormal uncertain standard deviations"}
}

GuiKeyWord kw_315[11] [static]
Initial value:

= {
    {descriptors*,15,0,5,0,1919,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,"Descriptors"},
    {initial_point*,14,0,4,0,1917,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,"Initialize uncertain"},
    {lambdas*,14,2,1,1,1903,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,"Choose characterization"},
    {lnuv_lambdas*,6,2,1,1,1902,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,"Lognormal uncertain lambdas"},
    {lnuv_lower_bounds*,6,0,2,0,1912,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,"Lognormal uncertain lower bounds"},
    {lnuv_means*,6,4,1,1,1906,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,"Lognormal uncertain means"},
    {lnuv_upper_bounds*,6,0,1,1,1914,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,"Lognormal uncertain upper bounds"}
}
CHAPTER 12. NAMESPACE DOCUMENTATION

```json
{"lower_bounds",14,0,2,0,1913,0,0,0,0,0,0,"{distribution lower bounds
VarCommands.html#VarCAUV_Lognormal",0,"lognormal uncertain"},
{"means",14,4,1,1907,0,0,0,0,0,0,"{lognormal uncertain means
VarCommands.html#VarCAUV_Lognormal",0,"lognormal uncertain"},
{"upper_bounds",14,0,3,0,1915,0,0,0,0,0,0,"{distribution upper bounds
VarCommands.html#VarCAUV_Lognormal",0,"lognormal uncertain"} }

GuiKeyWord kw_316[7] [static]
Initial value:  
= {
    
  }

GuiKeyWord kw_317[5] [static]
Initial value:  
= {
    
  }

GuiKeyWord kw_318[11] [static]
Initial value:  
= {
    
  }
```
GuiKeyWord kw_319[3] [static]
Initial value:

```java
= {
    "descriptors",15,0,3,0,2035,0,0,0,0,0,"{Descriptors} VarCommands.html#VarDAUV_Poisson",
   0,"poisson_uncertain"],
    "initial_point",13,0,2,0,2033,0,0,0,0,"{poisson uncertain lambda} VarCommands.html#VarDAUV_Poisson",
   0,"poisson uncertain"
```

GuiKeyWord kw_320[9] [static]
Initial value:

```java
= {
    "descriptors",15,0,5,0,1951,0,0,0,0,0,"{Descriptors} VarCommands.html#VarCAUV_Triangular",
   0,"triangular uncertain"],
    "initial_point",14,0,4,0,1949,0,0,0,0,0,"{triangular uncertain initial point} VarCommands.html#VarCAUV_Triangular",
   0,"triangular uncertain"],
    "lower_bounds",14,0,1,1,1944,0,0,0,0,0,"{Distribution lower bounds} VarCommands.html#VarCAUV_Triangular",
   0,"triangular uncertain"],
    "modes",14,0,1,1,1942,0,0,0,0,0,"{triangular uncertain modes} VarCommands.html#VarCAUV_Triangular",
   0,"triangular uncertain"],
    "tuv_descriptors",7,0,5,0,1950,0,0,0,0,0,"{triangular uncertain tuv descriptors} VarCommands.html#VarCAUV_Triangular",
   0,"triangular uncertain"],
    "tuv_lower_bounds",6,0,2,2,1944,0,0,0,0,0,"{Distribution lower bounds} VarCommands.html#VarCAUV_Triangular",
   0,"triangular uncertain"],
    "tuv_modes",6,0,1,1,1942,0,0,0,0,0,"{triangular uncertain modes} VarCommands.html#VarCAUV_Triangular",
   0,"triangular uncertain"],
    "tuv_upper_bounds",6,0,3,3,1946,0,0,0,0,0,"{Distribution upper bounds} VarCommands.html#VarCAUV_Triangular",
   0,"triangular uncertain"],
    "upper_bounds",14,0,3,3,1947,0,0,0,0,0,"{Distribution upper bounds} VarCommands.html#VarCAUV_Triangular",
   0,"triangular uncertain"
```

GuiKeyWord kw_321[7] [static]
Initial value:

```java
= {
    "descriptors",15,0,4,0,1929,0,0,0,0,0,"{Descriptors} VarCommands.html#VarCAUV_Uniform",
   0,"uniform uncertain"],
    "initial_point",14,0,3,0,1927,0,0,0,0,0,"{uniform uncertain initial point} VarCommands.html#VarCAUV_Uniform",
   0,"uniform uncertain"],
    "lower_bounds",14,0,1,1,1923,0,0,0,0,0,"{Distribution lower bounds} VarCommands.html#VarCAUV_Uniform",
   0,"uniform uncertain"],
    "upper_bounds",14,0,2,2,1925,0,0,0,0,0,"{Distribution upper bounds} VarCommands.html#VarCAUV_Uniform",
   0,"uniform uncertain"],
    "wuv_descriptors",7,0,4,0,1928,0,0,0,0,0,"{uniform uncertain wuv descriptors} VarCommands.html#VarCAUV_Uniform",
   0,"uniform uncertain"],
    "wuv_lower_bounds",6,0,1,1,1922,0,0,0,0,0,"{Distribution lower bounds} VarCommands.html#VarCAUV_Uniform",
   0,"uniform uncertain"],
    "wuv_modes",6,0,1,1,1922,0,0,0,0,0,"{uniform uncertain modes} VarCommands.html#VarCAUV_Uniform",
   0,"uniform uncertain"],
    "wuv_upper_bounds",6,0,2,2,1924,0,0,0,0,0,"{Distribution upper bounds} VarCommands.html#VarCAUV_Uniform",
   0,"uniform uncertain"
```

GuiKeyWord kw_322[7] [static]
Initial value:

```java
= {
    "alphas",14,0,1,1,2007,0,0,0,0,0,"{weibull uncertain alphas} VarCommands.html#VarCAUV_Weibull",
   0,"weibull uncertain alphas"],
    "betas",14,0,2,2,2009,0,0,0,0,0,"{weibull uncertain betas} VarCommands.html#VarCAUV_Weibull",
   0,"weibull uncertain"],
    "descriptors",15,0,4,0,2013,0,0,0,0,0,"{Descriptors} VarCommands.html#VarCAUV_Weibull",
   0,"weibull uncertain"],
    "initial_point",14,0,3,0,2011,0,0,0,0,0,"{weibull uncertain initial point} VarCommands.html#VarCAUV_Weibull",
   0,"weibull uncertain"],
    "wuv_alphas",6,0,1,1,2006,0,0,0,0,0,"{weibull uncertain wuv alphas} VarCommands.html#VarCAUV_Weibull",
   0,"weibull uncertain"],
    "wuv_betas",6,0,2,2,2008,0,0,0,0,0,"{weibull uncertain wuv betas} VarCommands.html#VarCAUV_Weibull",
   0,"weibull uncertain"],
    "wuv_descriptors",7,0,4,0,2012,0,0,0,0,0,"{weibull uncertain wuv descriptors} VarCommands.html#VarCAUV_Weibull",
   0,"weibull uncertain"
```
GuiKeyWord kw_324[6]  [static]
Initial value:
= {
  "environment",0x108,6,1,1,kw,3,0.,0.,0.,0.,0.,"{Environment} The environment specifies the
top level technique which will govern the management of iterators and models in the solution of the problem
of interest. EnvCommands.html"},
  "interface",0x308,9,5,5,2183,kw,18,0.,0.,0.,0.,"{Interface} An interface specifies how
function evaluations will be performed in order to map a set of parameters into a set of responses.
InterfCommands.html"},
  "method",0x308,90,2,2,17,kw,241,0.,0.,0.,0.,"{Method} A method specifies the name and
controls of an iterative procedure, e.g., a sensitivity analysis, uncertainty quantification, or optimization
method. MethodCommands.html"},
  "model",8,7,3,3,1595,kw,271,0.,0.,0.,0.,"{Model} A model consists of a model type and maps
specified variables through an interface to generate responses. ModelCommands.html"},
  "responses",0x308,19,6,6,2301,kw,289,0.,0.,0.,0.,"{Responses} A responses object specifies
the data that can be returned to DAKOTA through the interface after the completion of a function evaluation.
RespCommands.html"},
  "variables",0x308,37,4,4,1821,kw,323,0.,0.,0.,0.,"{Variables} A variables object specifies
the parameter set to be iterated by a particular method. VarCommands.html"
}

KeyWord kw_1[1]  [static]
Initial value:
= {
  "results_output_file",11,0,1,0,0.,0.,0.,0.,N,stm(str,resultsOutputFile)}

1231 distinct keywords (plus 180 aliases)

KeyWord kw_2[1]  [static]
Initial value:
= {
  "tabular_graphics_file",11,0,1,0,0.,0.,0.,0.,N,stm(str,tabularDataFile)}

KeyWord kw_3[6]  [static]
Initial value:
= {
  "graphics",8,0,1,0,0.,0.,0.,0.,N,stm(true,graphicsFlag)},
  "method_pointer",3,0,5,0,0.,0.,0.,0.,4,N,stm(str,topMethodPointer)},
  "output_precision",0x29,0,3,0,0.,0.,0.,0.,N,stm(int,outputPrecision)},
  "results_output",8,1,4,0,kw,1,0.,0.,0.,0.,N,stm(true,resultsOutputFlag)},
  "tabular_graphics_data",8,1,2,0,kw,2,0.,0.,0.,0.,N,stm(true,tabularDataFlag)},
  "top_method_pointer",11,0,5,0,0.,0.,0.,0.,N,stm(str,topMethodPointer)}

KeyWord kw_4[1]  [static]
Initial value:
= {
  "cache_tolerance",10,0,1,0,0.,0.,0.,0.,N,ifm(Real,nearbyEvalCacheTol)}
12.1. DAKOTA NAMESPACE REFERENCE

KeyWord kw_5[4] [static]
Initial value:

```plaintext
= {
  {"active_set_vector", 8, 0, 1, 0, 0, 0, 0, 0, N_ifm(false, activeSetVectorFlag)},
  {"evaluation_cache", 8, 0, 2, 0, 0, 0, 0, 0, N_ifm(false, evalCacheFlag)},
  {"restart_file", 8, 0, 4, 0, 0, 0, 0, 0, N_ifm(false, restartFileFlag)},
  {"strict_cache_equality", 8, 1, 3, 0, kw_4, 0, 0, 0, N_ifm(true, nearbyEvalCacheFlag)}
}
```

KeyWord kw_6[1] [static]
Initial value:

```plaintext
= {
  {"processors_per_analysis", 0x19, 0, 1, 0, 0, 0, 0, 0, N_ifm(pint, procsPerAnalysis)}
}
```

KeyWord kw_7[4] [static]
Initial value:

```plaintext
= {
  {"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, TYPE_DATA, failAction_recover)},
  {"retry", 9, 0, 1, 1, 0, 0, 0, 0, N_ifm(ilit, TYPE_DATA, failAction_retry)}
}
```

KeyWord kw_8[1] [static]
Initial value:

```plaintext
= {
  {"numpy", 8, 0, 1, 0, 0, 0, 0, 0, N_ifm(true, numpyFlag)}
}
```

KeyWord kw_9[2] [static]
Initial value:

```plaintext
= {
  {"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)}
}
```

KeyWord kw_10[7] [static]
Initial value:

```plaintext
= {
  {"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_9, 0, 0, 0, N_ifm(str, templateDir)},
  {"template_files", 15, 2, 4, 0, kw_9, 0, 0, 0, N_ifm(strL, templateFiles)}
}
```
CHAPTER 12. NAMESPACE DOCUMENTATION

KeyWord kw_11[9] [static]
Initial value:
= {
  {"allow_existing_results",8,0,3,0,0,0,0,0,N_ifm(true,allowExistingResultsFlag)},
  {"aprepro",0,0,5,0,0,0,0,0,N_ifm(true,apreproFlag)},
  {"dprepro",0,0,1,0,0,0,0,0,N_ifm(true,dpreproFlag)},
  {"file_save",8,0,7,0,0,0,0,0,N_ifm(true,fileSaveFlag)},
  {"file_tag",8,0,6,0,0,0,0,0,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",8,0,4,0,0,0,0,0,N_ifm(true,verbatimFlag)},
  {"work_directory",8,7,8,0,kw,10,0.,0.,0,N_ifm(true,useWorkdir)}
}

KeyWord kw_12[12] [static]
Initial value:
= {
  {"analysis_components",15,0,1,0,0,0,0,0,N_ifm(str2D,analysisComponents)},
  {"deactivate",8,4,6,0,kw,5},
  {"direct",8,1,4,1,kw,6,0,0,0,N_ifm(lit,interfaceType_direct)},
  {"failure_capture",8,4,5,0,kw,7},
  {"fork",8,9,4,1,kw,11,0,0,0,0,N_ifm(lit,interfaceType_fork)},
  {"grid",8,0,4,1,0,0,0,0,N_ifm(lit,interfaceType_grid)},
  {"input_filter",11,0,2,0,0,0,0,0,N_ifm(str,inputFilter)},
  {"matlab",8,0,4,1,0,0,0,0,0,N_ifm(lit,interfaceType_matlab)},
  {"output_filter",11,0,3,0,0,0,0,0,0,N_ifm(str,outputFilter)},
  {"python",8,1,4,1,kw,8,0,0,0,0,N_ifm(lit,interfaceType_python)},
  {"scilab",8,0,4,1,0,0,0,0,0,N_ifm(lit,interfaceType_scilab)},
  {"system",8,9,4,1,kw,11,0,0,0,0,N_ifm(lit,interfaceType_system)}
}

KeyWord kw_13[2] [static]
Initial value:
= {
  {"master",8,0,1,1,0,0,0,0,N_ifm(type,analysisScheduling_MASTER_SCHEDULING)},
  {"peer",8,0,1,1,0,0,0,0,N_ifm(type,analysisScheduling_PEER_SCHEDULING)}
}

KeyWord kw_14[2] [static]
Initial value:
= {
  {"dynamic",8,0,1,1,0,0,0,0,N_ifm(type,asynchLocalEvalScheduling_DYNAMIC_SCHEDULING)},
  {"static",8,0,1,1,0,0,0,0,N_ifm(type,asynchLocalEvalScheduling_STATIC_SCHEDULING)}
}

KeyWord kw_15[3] [static]
Initial value:
= {
  {"analysis_concurrency",0x19,0,3,0,0,0,0,0,N_ifm(pint,asynchLocalAnalysisConcurrency)},
  {"evaluation_concurrency",0x19,0,1,0,0,0,0,0,N_ifm(pint,asynchLocalEvalConcurrency)},
  {"local_evaluation_scheduling",8,2,2,0,kw,14}
}
12.1. DAKOTA NAMESPACE REFERENCE

KeyWord kw_16[2] [static]
Initial value:
= {
  {"dynamic", 8, 0, 1, 0, 0, 0, 0, 0, N, ifm(type, evalScheduling, PEER, DYNAMIC_SCHEDULING)},
  {"static", 8, 0, 1, 0, 0, 0, 0, 0, N, ifm(type, evalScheduling, PEER, STATIC_SCHEDULING)}
}

KeyWord kw_17[2] [static]
Initial value:
= {
  {"master", 8, 0, 1, 0, 0, 0, 0, 0, N, ifm(type, evalScheduling, MASTER_SCHEDULING)},
  {"peer", 8, 2, 1, 1, kw_16}
}

KeyWord kw_18[9] [static]
Initial value:
= {
  {"algebraic mappings", 11, 0, 2, 0, 0, 0, 0, 0, N, ifm(str, algebraicMappings)},
  {"analysis drivers", 15, 12, 3, 0, kw_12, 0, 0, 0, N, ifm(str, analysisDrivers)},
  {"analysis scheduling", 8, 2, 9, 0, kw_13},
  {"analysis servers", 0x19, 0, 8, 0, 0, 0, 0, 0, N, ifm(pint, analysisServers)},
  {"asynchronous", 8, 3, 4, 0, kw_15, 0, 0, 0, N, ifm(lit, interfaceSynchronization, asynchronous)},
  {"evaluation scheduling", 8, 2, 6, 0, kw_17},
  {"evaluation servers", 0x19, 0, 5, 0, 0, 0, 0, 0, N, ifm(pint, evalServers)},
  {"id interface", 11, 0, 1, 0, 0, 0, 0, 0, N, ifm(str, idInterface)},
  {"processors per evaluation", 0x19, 0, 7, 0, 0, 0, 0, 0, N, ifm(pint, procsPerEval)}
}

KeyWord kw_19[1] [static]
Initial value:
= {
  {"model pointer", 11, 0, 1, 0, 0, 0, 0, 0, N, mdm(str, modelPointer)}
}

KeyWord kw_20[2] [static]
Initial value:
= {
  {"samples", 9, 0, 1, 0, 0, 0, 0, 0, N, mdm(int, numSamples)},
  {"seed", 0x19, 0, 2, 0, 0, 0, 0, 0, N, mdm(pint, randomSeed)}
}

KeyWord kw_21[2] [static]
Initial value:
= {
  {"complementary", 8, 0, 1, 1, 0, 0, 0, 0, N, mdm(type, distributionType, COMPLEMENTARY)},
  {"cumulative", 8, 0, 1, 1, 0, 0, 0, 0, N, mdm(type, distributionType, CUMULATIVE)}
}
KeyWord kw_22[1]  [static]
Initial value:

```cpp
= {
    "num_gen_reliability_levels",13,0,1,0,0.,0.,0.,0,N_mdm(num_replevs,genReliabilityLevels)}
```

KeyWord kw_23[1]  [static]
Initial value:

```cpp
= {
    "num_probability_levels",13,0,1,0,0.,0.,0.,0,N_mdm(num_replevs,probabilityLevels)}
```

KeyWord kw_24[2]  [static]
Initial value:

```cpp
= {
    "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)}
}
```

KeyWord kw_25[4]  [static]
Initial value:

```cpp
= {
    "distribution",8,2,1,0,kw_21],
    "gen_reliability_levels",14,1,3,0,kw_22,0.,0.,0,N_mdm(resplevs,genReliabilityLevels),
    "probability_levels",14,1,2,0,kw_23,0.,0.,0,N_mdm(resplevs01,probabilityLevels),
    "rng",8,2,4,5,kw_24
}
```

KeyWord kw_26[4]  [static]
Initial value:

```cpp
= {
    "constant_liar",8,0,1,1,0,0.,0.,0,N_mdm(lit,batchSelectionType_constant_liar)
    "distance_penalty",8,0,1,1,0,0.,0.,0,N_mdm(lit,batchSelectionType_distance_penalty),
    "naive",8,0,1,1,0,0.,0.,0,N_mdm(lit,batchSelectionType_naive)
    "topology",8,0,1,1,0,0.,0.,0,N_mdm(lit,batchSelectionType_topoogy)
}
```

KeyWord kw_27[2]  [static]
Initial value:

```cpp
= {
    "annotated",8,0,1,0,0.,0.,0.,0,N_mdm(true,approxExportAnnotated)
    "freeform",8,0,1,0,0.,0.,0.,0,N_mdm(false,approxExportAnnotated)
}
```
KeyWord kw_28[3] [static]
Initial value:
= {
   "distance",8,0,1,1,0,0,0,0,N_mdm(lit,fitnessMetricType_distance),
   "gradient",8,0,1,1,0,0,0,0,N_mdm(lit,fitnessMetricType_gradient),
   "predicted_variance",8,0,1,1,0,0,0,0,N_mdm(lit,fitnessMetricType_predicted_variance)
}

KeyWord kw_29[2] [static]
Initial value:
= {
   "annotated",8,0,1,0,0,0,0,0,N_mdm(true,approxImportAnnotated),
   "freeform",8,0,1,0,0,0,0,0,N_mdm(false,approxImportAnnotated)
}

KeyWord kw_30[2] [static]
Initial value:
= {
   "parallel",8,0,1,1,0,0,0,0,N_mdm(type,responseLevelTargetReduce_SYSTEM_PARALLEL),
   "series",8,0,1,1,0,0,0,0,N_mdm(type,responseLevelTargetReduce_SYSTEM_SERIES)
}

KeyWord kw_31[3] [static]
Initial value:
= {
   "gen_reliabilities",8,0,1,1,0,0,0,0,N_mdm(type,responseLevelTarget_GEN_RELIABILITIES),
   "probabilities",8,0,1,1,0,0,0,0,N_mdm(type,responseLevelTarget_PROBABILITIES),
   "system",8,2,2,0,kw_30
}

KeyWord kw_32[2] [static]
Initial value:
= {
   "compute",8,3,2,0,kw_31,
   "num_response_levels",13,0,1,0,0,0,0,0,N_mdm(num_resplevs,responseLevels)
}

KeyWord kw_33[11] [static]
Initial value:
= {
   0,0,1,0,0,kw_19,
   0,0,2,0,0,kw_20,
   0,0,4,0,0,kw_25,
   "batch_selection",8,4,3,0,kw_26,
   "batch_size",9,0,4,0,0,0,0,0,N_mdm(int,batchSize),
   "emulator_samples",9,0,1,0,0,0,0,0,N_mdm(int,emulatorSamples),
   "export_points_file",11,2,6,0,kw_27,0,0,0,N_mdm(str,approxExportFile),
   "fitness_metric",8,3,2,0,kw_28,
   "import_points_file",11,2,5,0,kw_29,0,0,0,N_mdm(str,approxImportFile),
   "misc_options",15,0,8,0,0,0,0,0,N_mdm(str,miscOptions),
   "response_levels",14,2,7,0,kw_32,0,0,0,N_mdm(resplevs,responseLevels)
}
CHAPTER 12. NAMESPACE DOCUMENTATION

KeyWord kw_34[9] [static]

Initial value:

= {
    {"linear_equality_constraint_matrix",14,0,6,0,0,..,0,N_mdm(RealDL,
        linearEqConstraintCoeffs)},
    {"linear_equality_scale_types",15,0,8,0,0,..,0,N_mdm(strL,linearEqScaleTypes)},
    {"linear_equality_scales",14,0,9,0,0,..,0,N_mdm(RealDL,linearEqScales)},
    {"linear_equality_targets",14,0,7,0,0,..,0,N_mdm(RealDL,linearEqTargets)},
    {"linear_inequality_constraint_matrix",14,0,1,0,0,..,0,N_mdm(RealDL,
        linearIneqConstraintCoeffs)},
    {"linear_inequality_lower_bounds",14,0,2,0,0,..,0,N_mdm(RealDL,linearIneqLowerBnds)},
    {"linear_inequality_scale_types",15,0,4,0,0,..,0,N_mdm(strL,linearIneqScaleTypes)},
    {"linear_inequality_scales",14,0,5,0,0,..,0,N_mdm(RealDL,linearIneqScales)},
    {"linear_inequality_upper_bounds",14,0,3,0,0,..,0,N_mdm(RealDL,linearIneqUpperBnds)}
}

KeyWord kw_35[7] [static]

Initial value:

= {
    {"merit1",8,0,1,1,0,0,..,0,N_mdm(lit,meritFunction_merit1)},
    {"merit1_smooth",8,0,1,1,0,0,..,0,N_mdm(lit,meritFunction_merit1_smooth)},
    {"merit2",8,0,1,1,0,0,..,0,N_mdm(lit,meritFunction_merit2)},
    {"merit2_smooth",8,0,1,1,0,0,..,0,N_mdm(lit,meritFunction_merit2_smooth)},
    {"merit2_squared",8,0,1,1,0,0,..,0,N_mdm(lit,meritFunction_merit2_squared)},
    {"merit_max",8,0,1,1,0,0,..,0,N_mdm(lit,meritFunction_merit_max)},
    {"merit_max_smooth",8,0,1,1,0,0,..,0,N_mdm(lit,meritFunction_merit_max_smooth)}
}

KeyWord kw_36[2] [static]

Initial value:

= {
    {"blocking",8,0,1,1,0,0,..,0,N_mdm(lit,evalSynchronize_blocking)},
    {"nonblocking",8,0,1,1,0,0,..,0,N_mdm(lit,evalSynchronize_nonblocking)}
}

KeyWord kw_37[11] [static]

Initial value:

= {
    {0,0,1,0,0,kw_19},
    {0,0,9,0,kw_34},
    {"constraint_penalty",10,0,7,0,0,0,..,0,N_mdm(Real,constrPenalty)},
    {"contraction_factor",10,0,2,0,0,0,..,0,N_mdm(Real,contractStepLength)},
    {"initial_delta",10,0,1,0,0,0,..,0,N_mdm(Real,initStepLength)},
    {"merit_function",8,7,6,0,kw_35},
    {"smoothing_factor",10,0,8,0,0,0,..,0,N_mdm(Real,smoothFactor)},
    {"solution_accuracy",2,0,4,0,0,0,..,1,N_mdm(Real,solnTarget)},
    {"solution_target",10,0,4,0,0,0,..,0,N_mdm(Real,solnTarget)},
    {"synchronization",8,2,5,0,kw_36},
    {"threshold_delta",10,0,3,0,0,0,..,0,N_mdm(Real,threshStepLength)}
}
12.1. DAKOTA NAMESPACE REFERENCE

KeyWord kw_38[2] [static]
Initial value:

= {
  {"annotated", 8, 0, 1, 0, 0, 0, 0, 0, N, mdm(true, approxExportAnnotated)},
  {"freeform", 8, 0, 1, 0, 0, 0, 0, 0, N, mdm(false, approxExportAnnotated)}
}

KeyWord kw_39[2] [static]
Initial value:

= {
  {"annotated", 8, 0, 1, 0, 0, 0, 0, 0, N, mdm(true, approxImportAnnotated)},
  {"freeform", 8, 0, 1, 0, 0, 0, 0, 0, N, mdm(false, approxImportAnnotated)}
}

KeyWord kw_40[5] [static]
Initial value:

= {
  {"dakota", 8, 0, 1, 1, 0, 0, 0, 0, N, mdm(type, emulatorType, GP, EMULATOR)},
  {"emulator_samples", 9, 0, 2, 0, 0, 0, 0, 0, N, mdm(int, emulatorSamples)},
  {"export_points_file", 11, 2, 4, 0, kw_38, 0, 0, 0, N, mdm(str, approxExportFile)},
  {"import_points_file", 11, 2, 3, 0, kw_39, 0, 0, 0, N, mdm(str, approxImportFile)},
  {"surfpack", 8, 0, 1, 1, 0, 0, 0, 0, N, mdm(type, emulatorType, KRIGING, EMULATOR)}
}

KeyWord kw_41[1] [static]
Initial value:

= {
  {"sparse_grid_level", 13, 0, 1, 0, 0, 0, 0, 0, 0, N, mdm(usharray, sparseGridLevel)}
}

KeyWord kw_42[1] [static]
Initial value:

= {
  {"sparse_grid_level", 13, 0, 1, 0, 0, 0, 0, 0, 0, N, mdm(usharray, sparseGridLevel)}
}

KeyWord kw_43[4] [static]
Initial value:

= {
  {"gaussian_process", 8, 5, 1, 1, kw_40},
  {"kriging", 0, 5, 1, 1, kw_40, 0, 0, 0, 0},
  {"pce", 8, 1, 1, 1, kw_41, 0, 0, 0, 0, N, mdm(type, emulatorType, PCE, EMULATOR)},
  {"sc", 8, 1, 1, 1, kw_42, 0, 0, 0, 0, N, mdm(type, emulatorType, SC, EMULATOR)}
}
KeyWord kw_44[6]  [static]
Initial value:
= {
   {"chains",0x29,0,1,0,0,3,0,0,0,N_mdm(int,numChains)},
   {"crossover_chains_pairs",0x29,0,3,0,0,0,0,0,0,N_mdm(int,crossoverChainPairs)},
   {"gr_threshold",0x1a,0,0,0,0,0,0,0,0,N_mdm(Real,grThreshold)},
   {"jump_step",0x29,0,5,0,0,0,0,0,0,N_mdm(int,jumpStep)},
   {"num_cr",0x29,0,2,0,1,0,0,0,0,N_mdm(int,numCR)}
}

KeyWord kw_45[2]  [static]
Initial value:
= {
   {"adaptive",8,0,1,0,0,0,0,0,0,N_mdm(lit,metropolisType_adaptive)},
   {"hastings",8,0,1,0,0,0,0,0,0,N_mdm(lit,metropolisType_hastings)}
}

KeyWord kw_46[2]  [static]
Initial value:
= {
   {"delayed",8,0,1,0,0,0,0,0,0,N_mdm(lit,rejectionType_delayed)},
   {"standard",8,0,1,0,0,0,0,0,0,N_mdm(lit,rejectionType_standard)}
}

KeyWord kw_47[2]  [static]
Initial value:
= {
   {"metropolis",8,2,2,0,kw_45},
   {"rejection",8,2,1,0,kw_46} 
}

KeyWord kw_48[2]  [static]
Initial value:
= {
   {"dram",8,2,1,1,kw_47,0,0,0,0,N_mdm(lit,mcmcType_dram)},
   {"multilevel",8,0,1,1,0,0,0,0,0,N_mdm(lit,mcmcType_multilevel)}
}

KeyWord kw_49[2]  [static]
Initial value:
= {
   {"mt19937",8,0,1,1,0,0,0,0,0,N_mdm(lit,rngName_mt19937)},
   {"rnum2",8,0,1,1,0,0,0,0,0,N_mdm(lit,rngName_rnum2)}
}
12.1. DAKOTA NAMESPACE REFERENCE

**KeyWord kw_50[3]** [static]
Initial value:
= {
    "mcmc_type",8,2,1,0,kw_48,
    "proposal_covariance_scale",14,0,3,0,0,0,0,0,N_mdm(RealDL,proposalCovScale),
    "rng",8,2,2,0,kw_49
}

**KeyWord kw_51[2]** [static]
Initial value:
= {
    "annotated",8,0,1,0,0,0,0,0,N_mdm(true,approxExportAnnotated),
    "freeform",8,0,1,0,0,0,0,0,N_mdm(false,approxExportAnnotated)
}

**KeyWord kw_52[2]** [static]
Initial value:
= {
    "annotated",8,0,1,0,0,0,0,0,N_mdm(true,approxImportAnnotated),
    "freeform",8,0,1,0,0,0,0,0,N_mdm(false,approxImportAnnotated)
}

**KeyWord kw_53[4]** [static]
Initial value:
= {
    0,0,3,0,0,kw_50,
    "emulator_samples",9,0,1,0,0,0,0,0,N_mdm(int,emulatorSamples),
    "export_points_file",11,2,3,0,kw_51,0,0,0,N_mdm(str,approxExportFile),
    "import_points_file",11,2,2,0,kw_52,0,0,0,N_mdm(str,approxImportFile)
}

**KeyWord kw_54[2]** [static]
Initial value:
= {
    "annotated",8,0,1,0,0,0,0,0,N_mdm(true,approxExportAnnotated),
    "freeform",8,0,1,0,0,0,0,0,N_mdm(false,approxExportAnnotated)
}

**KeyWord kw_55[2]** [static]
Initial value:
= {
    "annotated",8,0,1,0,0,0,0,0,N_mdm(true,approxImportAnnotated),
    "freeform",8,0,1,0,0,0,0,0,N_mdm(false,approxImportAnnotated)
}
KeyWord kw_56[5] [static]
Initial value:
= {
    {"dakota", 8, 0, 1, 1, 0, 0, 0, 0, 0, N_mdm(type, emulatorType::GP_EMULATOR),
     "emulator_samples", 9, 0, 1, 1, 0, 0, 0, 0, N_mdm(int, emulatorSamples),
     "export_points_file", 11, 2, 4, 0, kw_54, 0, 0, 0, N_mdm(str, approxExportFile),
     "import_points_file", 11, 2, 3, 0, kw_55, 0, 0, 0, N_mdm(str, approxImportFile),
     "surfpack", 8, 0, 1, 1, 0, 0, 0, 0, N_mdm(type, emulatorType::KRIGING_EMULATOR)}
}

KeyWord kw_57[1] [static]
Initial value:
= {
    {"sparse_grid_level", 13, 0, 1, 0, 0, 0, 0, 0, N_mdm(usharray, sparseGridLevel)}
}

KeyWord kw_58[1] [static]
Initial value:
= {
    {"sparse_grid_level", 13, 0, 1, 0, 0, 0, 0, 0, N_mdm(usharray, sparseGridLevel)}
}

KeyWord kw_59[4] [static]
Initial value:
= {
    {"gaussian_process", 8, 5, 1, 1, kw_56},
    {"kriging", 0, 5, 1, 1, kw_56, 0, 0, -1},
    {"pce", 8, 1, 1, kw_57, 0, 0, 0, N_mdm(type, emulatorType::PCE_EMULATOR)},
    {"sc", 8, 1, 1, kw_58, 0, 0, 0, N_mdm(type, emulatorType::SC_EMULATOR)}
}

KeyWord kw_60[2] [static]
Initial value:
= {
    {0, 0, 3, 0, 0, kw_50},
    {"emulator", 8, 4, 1, 0, kw_59}
}

KeyWord kw_61[8] [static]
Initial value:
= {
    {0, 0, 1, 0, 0, kw_19},
    {0, 0, 2, 0, 0, kw_20},
    {"calibrate_sigma", 8, 0, 4, 0, 0, 0, 0, 0, N_mdm(true, calibrateSigmaFlag)},
    {"dream", 8, 6, 1, 1, kw_24, 0, 0, 0, N_mdm(utype, subMethod::SUBMETHOD::DREAM)},
    {"gpmsa", 8, 3, 1, 1, kw_33, 0, 0, 0, N_mdm(utype, subMethod::SUBMETHOD::GPMSA)},
    {"likelihood_scale", 10, 0, 3, 0, 0, 0, 0, 0, N_mdm(Real, likelihoodScale)},
    {"queso", 8, 1, 1, kw_60, 0, 0, 0, N_mdm(utype, subMethod::SUBMETHOD::QUESO)},
    {"use_derivatives", 8, 0, 2, 0, 0, 0, 0, 0, N_mdm(true, methodUseDerivsFlag)}
}
12.1. DAKOTA NAMESPACE REFERENCE

**KeyWord kw_62[4]** [static]
Initial value:
```cpp
= {
  0,0,1,0,kw,19,
  0,0,2,2,0,0,0,0,0,N_mdm(ivec,stepsPerVariable),
  "deltas_per_variable",5,0,2,2,0,0,0,0,0,N_mdm(ivec,stepsPerVariable),
  "step_vector",14,0,1,1,0,0,0,0,0,N_mdm(ivec,stepVector),
  "steps_per_variable",13,0,2,2,0,0,0,0,0,N_mdm(ivec,stepsPerVariable)
}
```

**KeyWord kw_63[5]** [static]
Initial value:
```cpp
= {
  "misc_options",15,0,4,0,0,0,0,N_mdm(strL,miscOptions),
  "seed",0x19,0,2,0,0,0,0,0,0,N_mdm(pint,randomSeed),
  "show_misc_options",8,0,3,0,0,0,0,0,0,N_mdm(true,showMiscOptions),
  "solution_accuracy",2,0,1,0,0,0,0,1,0,N_mdm(Real,solnTarget),
  "solution_target",10,0,1,0,0,0,0,0,0,N_mdm(Real,solnTarget)
}
```

**KeyWord kw_64[3]** [static]
Initial value:
```cpp
= {
  0,0,1,0,kw,19,
  0,0,5,0,kw,63,
  "beta_solver_name",11,0,1,1,0,0,0,0,0,N_mdm(str,betaSolverName)
}
```

**KeyWord kw_65[2]** [static]
Initial value:
```cpp
= {
  "initial_delta",10,0,1,0,0,0,0,0,0,N_mdm(Real,initDelta),
  "threshold_delta",10,0,2,0,0,0,0,0,0,N_mdm(Real,threshDelta)
}
```

**KeyWord kw_66[4]** [static]
Initial value:
```cpp
= {
  0,0,1,0,kw,19,
  0,0,5,0,kw,63,
  0,0,2,0,kw,65,
  "*
}
```

**KeyWord kw_67[2]** [static]
Initial value:
```cpp
= {
  "all_dimensions",8,0,1,1,0,0,0,0,0,N_mdm(lit,boxDivision_all_dimensions),
  "major_dimension",8,0,1,1,0,0,0,0,0,N_mdm(lit,boxDivision_major_dimension)
}
```
KeyWord kw.68[8] [static]
Initial value:

= {
    {0,0,1,0,0,kw.19},
    {0,0,5,0,0,kw.63},
    {"constraint_penalty",10,0,6,0,0,0,0,0,N,mdm(Real,constraintPenalty)},
    {"division",8,2,1,0,kw.67},
    {"global_balance_parameter",10,0,2,0,0,0,0,0,N,mdm(Real,globalBalanceParam)},
    {"local_balance_parameter",10,0,3,0,0,0,0,0,N,mdm(Real,localBalanceParam)},
    {"max_boxsize_limit",10,0,4,0,0,0,0,0,N,mdm(Real,maxBoxSize)},
    {"min_boxsize_limit",10,0,5,0,0,0,0,0,N,mdm(Real,minBoxSize)}
}

KeyWord kw.69[3] [static]
Initial value:

= {
    {"blend",8,0,1,1,0,0,0,0,N,mdm(lit,crossoverType_blend)},
    {"two_point",8,0,1,1,0,0,0,0,N,mdm(lit,crossoverType_two_point)},
    {"uniform",8,0,1,1,0,0,0,0,N,mdm(lit,crossoverType_uniform)}
}

KeyWord kw.70[2] [static]
Initial value:

= {
    {"linear_rank",8,0,1,1,0,0,0,0,N,mdm(lit,fitnessType_linear_rank)},
    {"merit_function",8,0,1,1,0,0,0,0,N,mdm(lit,fitnessType_proportional)}
}

KeyWord kw.71[3] [static]
Initial value:

= {
    {"flat_file",11,0,1,1,0,0,0,0,N,mdm(slit2,TYPE_DATA_initializationType_flat_file)},
    {"simple_random",8,0,1,1,0,0,0,0,N,mdm(lit,initializationType_random)},
    {"unique_random",8,0,1,1,0,0,0,0,N,mdm(lit,initializationType_unique_random)}
}

KeyWord kw.72[2] [static]
Initial value:

= {
    {"mutation_range",9,0,2,0,0,0,0,0,N,mdm(int,mutationRange)},
    {"mutation_scale",10,0,1,0,0,0,0,0,N,mdm(Real,mutationScale)}
}
KeyWord kw_73[5] [static]
Initial value:
= {
    "non_adaptive",8,0,2,0,0,0,0,N,mdm(false,mutationAdaptive),
    "offset_cauchy",8,2,1,1,kw_72,0,0,0,N,mdm(lit,mutationType_offset_cauchy),
    "offset_normal",8,2,1,1,kw_72,0,0,0,N,mdm(lit,mutationType_offset_normal),
    "replace_uniform",8,0,1,1,0,0,0,0,N,mdm(lit,mutationType_replace_uniform)
}

KeyWord kw_74[4] [static]
Initial value:
= {
    "chc",9,0,1,1,0,0,0,0,N,mdm(lit2,TYPE_DATA_replacementType_chc),
    "elitist",9,0,1,1,0,0,0,0,N,mdm(lit2,TYPE_DATA_replacementType_elitist),
    "new_solutions_generated",9,0,2,0,0,0,0,0,N,mdm(int,newSolnsGenerated),
    "random",9,0,1,1,0,0,0,0,N,mdm(lit2,TYPE_DATA_replacementType_random)
}

KeyWord kw_75[11] [static]
Initial value:
= {
    0,0,1,0,0,kw_19,
    0,0,5,0,0,kw_63,
    "constraint_penalty",10,0,9,0,0,0,0,0,N,mdm(Real,constraintPenalty),
    "crossover_rate",10,0,5,0,0,0,0,0,N,mdm(Real,crossoverRate),
    "crossover_type",8,3,6,0,kw_69,
    "fitness_type",8,2,3,0,kw_70,
    "initialization_type",8,3,2,0,kw_71,
    "mutation_rate",10,0,7,0,0,0,0,0,N,mdm(Real,mutationRate),
    "mutation_type",8,5,8,0,kw_73,
    "population_size",0x19,0,1,0,0,0,0,0,D,mdm(pint,populationSize),
    "replacement_type",8,6,4,0,kw_76
}

KeyWord kw_76[2] [static]
Initial value:
= {
    "constraint_penalty",10,0,2,0,0,0,0,0,N,mdm(Real,constraintPenalty),
    "contraction_factor",10,0,1,0,0,0,0,0,N,mdm(Real,contractFactor)
}

KeyWord kw_77[3] [static]
Initial value:
= {
    "adaptive_pattern",8,0,1,1,0,0,0,0,N,mdm(lit,exploratoryMoves_adaptive),
    "basic_pattern",8,0,1,1,0,0,0,0,N,mdm(lit,exploratoryMoves_simple),
    "multi_step",8,0,1,1,0,0,0,0,N,mdm(lit,exploratoryMoves_multi_step)
}
KeyWord kw_78[2] [static]
Initial value:
= {
    {"coordinate", 8, 0, 1, 0, 0, 0, 0, N, mdm(lit, patternBasis_coordinate)},
    {"simplex", 8, 0, 1, 0, 0, 0, 0, N, mdm(lit, patternBasis_simplex)}
}

KeyWord kw_79[2] [static]
Initial value:
= {
    {"blocking", 8, 0, 1, 0, 0, 0, 0, N, mdm(lit, evalSynchronize_blocking)},
    {"nonblocking", 8, 0, 1, 0, 0, 0, 0, N, mdm(lit, evalSynchronize_nonblocking)}
}

KeyWord kw_80[12] [static]
Initial value:
= {
    {0, 0, 1, 0, 0, kw_19},
    {0, 0, 5, 0, 0, kw_63},
    {0, 0, 2, 0, 0, kw_65},
    {0, 0, 2, 0, 0, kw_76},
    {"constant_penalty", 8, 0, 1, 0, 0, 0, 0, N, mdm(true, constantPenalty)},
    {"expand_after_success", 9, 0, 3, 0, 0, 0, 0, 0, N, mdm(int, expandAfterSuccess)},
    {"exploratory_moves", 8, 3, 7, 0, kw_77},
    {"no_expansion", 8, 0, 2, 0, 0, 0, 0, 0, N, mdm(false, expansionFlag)},
    {"pattern_basis", 8, 2, 4, 0, kw_78},
    {"stochastic", 8, 0, 5, 0, 0, 0, 0, 0, N, mdm(true, randomizeOrderFlag)},
    {"synchronization", 8, 2, 8, 0, kw_79},
    {"total_pattern_size", 9, 0, 6, 0, 0, 4, 0, 0, N, mdm(int, totalPatternSize)}
}

KeyWord kw_81[8] [static]
Initial value:
= {
    {0, 0, 1, 0, 0, kw_19},
    {0, 0, 5, 0, 0, kw_63},
    {0, 0, 2, 0, 0, kw_65},
    {0, 0, 2, 0, 0, kw_76},
    {"constant_penalty", 8, 0, 4, 0, 0, 0, 0, 0, 0, N, mdm(true, constantPenalty)},
    {"contract_after_failure", 9, 0, 1, 0, 0, 0, 0, 0, N, mdm(int, contractAfterFail)},
    {"expand_after_success", 9, 0, 3, 0, 0, 0, 0, 0, N, mdm(int, expandAfterSuccess)},
    {"no_expansion", 8, 0, 2, 0, 0, 0, 0, 0, N, mdm(false, expansionFlag)}
}

KeyWord kw_82[4] [static]
Initial value:
= {
    {0, 0, 1, 0, 0, kw_19},
    {0, 0, 9, 0, 0, kw_34},
    {"frcg", 8, 0, 1, 1, 0, 0, 0, 0, N, mdm(utype, methodName_CONMIN_FRCG)},
    {"mfd", 8, 0, 1, 1, 0, 0, 0, 0, N, mdm(utype, methodName_CONMIN_MFD)}
}
**KeyWord kw.83[3] [static]**
**Initial value:**
```cpp
= {
    {0,0,1,0,0,kw.19},
    {0,0,9,0,0,kw.34},
    *
}
```

**KeyWord kw.84[1] [static]**
**Initial value:**
```cpp
= {
    "drop_tolerance",10,0,1,0,0.,0.,0.,0,N_{mdm}(Real,vbdDropTolerance)
    ...
}
```

**KeyWord kw.85[14] [static]**
**Initial value:**
```cpp
= {
    {0,0,1,0,0,kw.19},
    {0,0,2,0,0,kw.20},
    "box_behnken",8,0,1,1,0,0.,0.,0.,0,N_{mdm}(utype,subMethod_SUBMETHOD_BOX_BEHNKEN),
    "central_composite",8,0,1,1,0,0.,0.,0.,0,N_{mdm}(utype,subMethod_SUBMETHOD_CENTRAL_COMPOSITE),
    "grid",8,0,1,1,0,0.,0.,0.,0,N_{mdm}(utype,subMethod_SUBMETHOD_GRID),
    "lhs",8,0,1,1,0,0.,0.,0.,0,N_{mdm}(utype,subMethod_SUBMETHOD_LHS),
    "main_effects",8,0,2,0,0.,0.,0.,0,N_{mdm}(true,mainEffectsFlag),
    "oa_lhs",8,0,1,1,0,0.,0.,0.,0,N_{mdm}(utype,subMethod_SUBMETHOD_OA_LHS),
    "oas",8,0,1,1,0,0.,0.,0.,0,N_{mdm}(utype,subMethod_SUBMETHOD_OAS),
    "quality_metrics",8,0,3,0,0.,0.,0.,0,N_{mdm}(true,volQualityFlag),
    "random",8,0,1,1,0,0.,0.,0.,0,N_{mdm}(utype,subMethod_SUBMETHOD_RANDOM),
    "symbols",9,0,6,0,0.,0.,0.,0,N_{mdm}(int,numSymbols)
    "variance_based_decomp",8,1,4,0,kw.84,0.,0.,0,N_{mdm}(true,vbdFlag)
    ...
}
```

**KeyWord kw.86[7] [static]**
**Initial value:**
```cpp
= {
    {0,0,1,0,0,kw.19},
    {0,0,9,0,0,kw.34},
    "bfgs",8,0,1,1,0,0.,0.,0.,0,N_{mdm}(utype,methodName_DOT_BFGS),
    "frcg",8,0,1,1,0,0.,0.,0.,0,N_{mdm}(utype,methodName_DOT_FRCG),
    "mmfd",8,0,1,1,0,0.,0.,0.,0,N_{mdm}(utype,methodName_DOT_MMFD),
    "slp",8,0,1,1,0,0.,0.,0.,0,N_{mdm}(utype,methodName_DOT_SLP),
    "scp",8,0,1,1,0,0.,0.,0.,0,N_{mdm}(utype,methodName_DOT_SCP)
    ...
}
```

**KeyWord kw.87[2] [static]**
**Initial value:**
```cpp
= {
    "annotated",8,0,1,0,0.,0.,0.,0,N_{mdm}(true,approxExportAnnotated),
    "freeform",8,0,1,0,0.,0.,0.,0,N_{mdm}(false,approxExportAnnotated)
    ...
}
```
KeyWord kw_88[2]  [static]
Initial value:
  = {
      {"dakota",8,0,1,0,0,0,0,N_mdm(type,emulatorType,GP_EMULATOR)},
      {"surfpack",8,0,1,0,0,0,0,N_mdm(type,emulatorType,KRIGING_EMULATOR)}
  }

KeyWord kw_89[2]  [static]
Initial value:
  = {
      {"annotated",8,0,1,0,0,0,0,N_mdm(true,approxImportAnnotated)},
      {"freeform",8,0,1,0,0,0,0,N_mdm(false,approxImportAnnotated)}
  }

KeyWord kw_90[7]  [static]
Initial value:
  = {
      {0,0,1,0,kw_19},
      {"export_points_file",11,2,4,0,kw_87,0,0,0,N_mdm(str,approxExportFile)},
      {"gaussian_process",8,2,1,0,kw_88},
      {"import_points_file",11,2,3,0,kw_89,0,0,0,N_mdm(str,approxImportFile)},
      {"kriging",0,2,1,0,kw_88,0,0,0,2},
      {"seed",0x19,0,1,0,0,0,0,0,N_mdm(pint,randomSeed)},
      {"use_derivatives",8,0,2,0,0,0,0,0,N_mdm(true,methodUseDerivsFlag)}
  }

KeyWord kw_91[5]  [static]
Initial value:
  = {
      {0,0,1,0,kw_19},
      {0,0,2,0,kw_20},
      {0,0,4,0,kw_25},
      {"batch_size",9,0,2,0,0,0,0,0,N_mdm(int,batchSize)},
      {"emulator_samples",9,0,1,0,0,0,0,0,N_mdm(int,emulatorSamples)}
  }

KeyWord kw_92[3]  [static]
Initial value:
  = {
      {"grid",8,0,1,0,0,0,0,N_mdm(lit,trialType,grid)},
      {"halton",8,0,1,0,0,0,0,N_mdm(lit,trialType,halton)},
      {"random",8,0,1,0,0,0,0,N_mdm(lit,trialType,random)}
  }

KeyWord kw_93[1]  [static]
Initial value:
  = {
      {"drop_tolerance",10,0,1,0,0,0,0,0,N_mdm(Real,vbdDropTolerance)}
  }
KeyWord kw_94[8] [static]
Initial value:
= {
    {0,0,1,0,0,kw.19},
    {0,0,2,0,0,kw.20},
    {"fixed_seed",8,0,4,0,0...0,N,mdm(true,fixedSeedFlag)},
    {"latinize",8,0,1,0,0...0,N,mdm(true,latinizeFlag)},
    {"num_trials",9,0,6,0,0...0,N,mdm(int,numTrials)},
    {"quality_metrics",8,0,2,0,0...0,N,mdm(true,volQualityFlag)},
    {"trial_type",8,3,5,0,kw.32},
    {"variance_based_decomp",8,1,3,0,kw.93,0...0,N,mdm(true,vbdFlag)}
}

KeyWord kw_95[1] [static]
Initial value:
= {
    {"drop_tolerance",10,0,1,0,0...0,N,mdm(Real,vbdDropTolerance)}
}

KeyWord kw_96[11] [static]
Initial value:
= {
    {0,0,1,0,0,kw.15},
    {"fixed_sequence",8,0,6,0,0...0,N,mdm(true,fixedSequenceFlag)},
    {"hammersley",8,0,1,0,0...0,N,mdm(utype,methodName_FSU_HAMMERSLEY)},
    {"prime_base",13,0,9,0,0...0,N,mdm(ivec,primeBase)},
    {"quality_metrics",8,0,3,0,0...0,N,mdm(true,volQualityFlag)},
    {"samples",9,0,5,0,0...0,N,mdm(int,numSamples)},
    {"sequence_leap",13,0,8,0,0...0,N,mdm(ivec,sequenceLeap)},
    {"sequence_start",13,0,7,0,0...0,N,mdm(ivec,sequenceStart)},
    {"variance_based_decomp",8,1,4,0,kw.95,0...0,N,mdm(true,vbdFlag)}
}

KeyWord kw_97[2] [static]
Initial value:
= {
    {"annotated",8,0,1,0,0...0,N,mdm(true,approxExportAnnotated)},
    {"freeform",8,0,1,0,0...0,N,mdm(false,approxExportAnnotated)}
}

KeyWord kw_98[2] [static]
Initial value:
= {
    {"annotated",8,0,1,0,0...0,N,mdm(true,approxImportAnnotated)},
    {"freeform",8,0,1,0,0...0,N,mdm(false,approxImportAnnotated)}
}
KeyWord kw_99[2] [static]
Initial value:
= {
    {"parallel",8,0,1,0,0,0,0,N_mdm(type,responseLevelTargetReduce_SYSTEM_PARALLEL)},
    {"series",8,0,1,0,0,0,0,N_mdm(type,responseLevelTargetReduce_SYSTEM_SERIES)}
}

KeyWord kw_100[3] [static]
Initial value:
= {
    {"gen_reliabilities",8,0,1,0,0,0,.0,N_mdm(type,responseLevelTarget_SEN_RELIABILITIES)},
    {"probabilities",8,0,1,0,0,0,.0,N_mdm(type,responseLevelTarget_PROBABILITIES)},
    {"system",8,2,2,0,kw_99}
}

KeyWord kw_101[2] [static]
Initial value:
= {
    {"compute",8,3,2,0,kw_100},
    {"num_response_levels",13,0,1,0,0,.0,0,N_mdm(num_resplevs,responseLevels)}
}

KeyWord kw_102[7] [static]
Initial value:
= {
    {0,0,1,0,0,kw_19},
    {0,0,2,0,0,kw_20},
    {0,0,4,0,0,kw_25},
    {"emulator_samples",9,0,1,0,0,0,.0,N_mdm(int,emulatorSamples)},
    {"export_points_file",11,2,3,0,kw_97,0,.0,0,N_mdm(str,approxExportFile)},
    {"import_points_file",11,2,2,0,kw_98,0,.0,0,N_mdm(str,approxImportFile)},
    {"response_levels",14,2,4,0,kw_101,0,.0,0,N_mdm(resplevs,responseLevels)}
}

KeyWord kw_103[2] [static]
Initial value:
= {
    {0,0,1,0,0,kw_19},
    {"seed",0x19,0,1,0,0,.0,0,N_mdm(pint,randomSeed)}
}

KeyWord kw_104[2] [static]
Initial value:
= {
    {"parallel",8,0,1,1,0,0,.0,0,N_mdm(type,responseLevelTargetReduce_SYSTEM_PARALLEL)},
    {"series",8,0,1,1,0,0,.0,0,N_mdm(type,responseLevelTargetReduce_SYSTEM_SERIES)}
}
KeyWord kw_105[3]  [static]
Initial value:

    = {
        {*"gen_reliabilities",8,0,1,0,0,0,0,0,N\_mdm(type, responseLevelTarget\_GEN, RELIABILITIES)*,}
        {*"probabilities",8,0,1,0,0,0,0,0,N\_mdm(type, responseLevelTarget\_PROBABILITIES)*,}
        {*"system",8,2,2,0,kw\_104*}
    }

KeyWord kw_106[2]  [static]
Initial value:

    = {
        {*"compute",8,3,2,0,kw\_105*},
        {*"num_response_levels",13,0,1,0,0,0,0,0,N\_mdm(num\_resplevs, responseLevels)*}
    }

KeyWord kw_107[2]  [static]
Initial value:

    = {
        {*"annotated",8,0,1,0,0,0,0,0,N\_mdm(true, approxExportAnnotated)*,}
        {*"freeform",8,0,1,0,0,0,0,0,N\_mdm(false, approxExportAnnotated)*}
    }

KeyWord kw_108[2]  [static]
Initial value:

    = {
        {*"dakota",8,0,1,0,0,0,0,0,N\_mdm(type, emulatorType\_GP, EMULATOR)*},
        {*"surfpack",8,0,1,0,0,0,0,0,N\_mdm(type, emulatorType\_KRIGING, EMULATOR)*}
    }

KeyWord kw_109[2]  [static]
Initial value:

    = {
        {*"annotated",8,0,1,0,0,0,0,0,N\_mdm(true, approxImportAnnotated)*,}
        {*"freeform",8,0,1,0,0,0,0,0,N\_mdm(false, approxImportAnnotated)*}
    }

KeyWord kw_110[5]  [static]
Initial value:

    = {
        {*"export_points_file",11,2,4,0,kw\_107,0,0,0,0,N\_mdm(str, approxExportFile)*},
        {*"gaussian_process",8,2,1,0,kw\_108*},
        {*"import_points_file",11,2,3,0,kw\_109,0,0,0,0,N\_mdm(str, approxImportFile)*},
        {*"kriging",0,2,1,0,kw\_108,0,0,-2*},
        {*"use_derivatives",8,0,2,0,0,0,0,0,0,N\_mdm(true, methodUseDerivsFlag)*}
    }
KeyWord kw_111[8] [static]
Initial value:
= {
    {0,0,1,0,0,kw_19},
    {0,0,2,0,0,kw_20},
    {0,0,4,0,0,kw_25},
    {"ea",8,0,1,0,0,0,0,0,0,0,N_mdm(utype,subMethod,SMETHOD_EA)},
    {"ego",8,5,1,0,kw_110,0,0,0,0,0,N_mdm(utype,subMethod,SMETHOD_EGO)},
    {"lhs",8,0,1,0,0,0,0,0,0,0,N_mdm(utype,subMethod,SMETHOD_LHS)},
    {"response_levels",14,2,0,kw_106,0,0,0,0,0,N_mdm(respLevels,respLevels)},
    {"sbo",8,1,0,kw_110,0,0,0,0,0,N_mdm(utype,subMethod,SMETHOD_SBO)}
}

KeyWord kw_112[2] [static]
Initial value:
= {
    {"mt19937",8,0,1,0,0,0,0,0,0,0,N_mdm(lit,rngName,mt19937)},
    {"rnum2",8,0,1,0,0,0,0,0,0,0,N_mdm(lit,rngName,rnum2)}
}

KeyWord kw_113[2] [static]
Initial value:
= {
    {"annotated",8,0,1,0,0,0,0,0,0,0,N_mdm(true,approxExportAnnotated)},
    {"freeform",8,0,1,0,0,0,0,0,0,0,N_mdm(false,approxExportAnnotated)}
}

KeyWord kw_114[2] [static]
Initial value:
= {
    {"dakota",8,0,1,0,0,0,0,0,0,0,N_mdm(type,emulatorType,GP_EMULATOR)},
    {"surfpack",8,0,1,0,0,0,0,0,0,0,N_mdm(type,emulatorType,KRIGING_EMULATOR)}
}

KeyWord kw_115[2] [static]
Initial value:
= {
    {"annotated",8,0,1,0,0,0,0,0,0,0,N_mdm(true,approxImportAnnotated)},
    {"freeform",8,0,1,0,0,0,0,0,0,0,N_mdm(false,approxImportAnnotated)}
}

KeyWord kw_116[5] [static]
Initial value:
= {
    {"export_points_file",11,2,4,0,kw_113,0,0,0,0,0,N_mdm(str,approxExportFile)},
    {"gaussian_process",8,2,1,0,kw_112},
    {"import_points_file",11,2,3,0,kw_115,0,0,0,0,0,N_mdm(str,approxImportFile)},
    {"kriging",0,2,1,0,kw_114,0,0,0,-2},
    {"use_derivatives",8,0,2,0,0,0,0,0,0,0,N_mdm(true,methodUseDerivsFlag)}
}
KeyWord kw_117[7] [static]
Initial value:
= {
    {0,0,1,0,0,kw\[19\]},
    {0,0,2,0,0,kw\[20\]},
    "ea\"/8,0,1,0,0,0,,0,,0,N,mdm(utype,subMethod_SUBMETHOD_EA)},
    "ego\"/8,5,1,0,kw\[116\],0,,0,,0,N,mdm(utype,subMethod_SUBMETHOD_EGO)},
    "lhs\"/8,0,1,0,0,0,,0,,0,N,mdm(utype,subMethod_SUBMETHOD_LHS)},
    "rng\"/8,2,2,0,kw\[112\]},
    "sbo\"/8,5,1,0,kw\[116\],0,,0,,0,N,mdm(utype,subMethod_SUBMETHOD_SBO)}

KeyWord kw_118[2] [static]
Initial value:
= {
    "complementary",8,0,1,0,0,0,,0,,0,N,mdm(type,distributionType_COMPLEMENTARY)},
    "cumulative",8,0,1,0,0,0,,0,,0,N,mdm(type,distributionType_CUMULATIVE)}

KeyWord kw_119[1] [static]
Initial value:
= {
    "num_gen_reliability_levels",13,0,1,0,0,0,,0,,0,N,mdm(num_resplevs,genReliabilityLevels)}

KeyWord kw_120[1] [static]
Initial value:
= {
    "num_probability_levels",13,0,1,0,0,0,,0,,0,N,mdm(num_resplevs,probabilityLevels)}

KeyWord kw_121[3] [static]
Initial value:
= {
    "distribution",8,2,1,0,kw\[118\]},
    "gen_reliability_levels",14,1,1,0,kw\[119\],0,,0,,0,N,mdm(resplevs,genReliabilityLevels)},
    "probability_levels",14,1,2,0,kw\[120\],0,,0,,0,N,mdm(resplevs01,probabilityLevels)}

KeyWord kw_122[2] [static]
Initial value:
= {
    "annotated",8,0,1,0,0,0,,0,,0,N,mdm(true,approxExportAnnotated)},
    "freeform",8,0,1,0,0,0,,0,,0,N,mdm(false,approxExportAnnotated)\}
KeyWord kw_123[2]  [static]
Initial value:
  = {
    {"annotated",8,0,1,0,0,0,0},N_mdm(true,approxImportAnnotated),
    {"freeform",8,0,1,0,0,0,0},N_mdm(false,approxImportAnnotated)
  }

KeyWord kw_124[2]  [static]
Initial value:
  = {
    {"parallel",8,0,1,1,0,0,0},N_mdm(type,responseLevelTargetReduce.SYSTEM_PARALLEL),
    {"series",8,0,1,1,0,0,0},N_mdm(type,responseLevelTargetReduce.SYSTEM_SERIES)
  }

KeyWord kw_125[3]  [static]
Initial value:
  = {
    {"gen_reliabilities",8,0,1,1,0,0,0},N_mdm(type,responseLevelTarget.JScrollPane.RELIABILITIES),
    {"probabilities",8,0,1,1,0,0,0},N_mdm(type,responseLevelTarget JScrollPane.PROBABILITIES),
    {"system",8,2,2,0},kw_124
  }

KeyWord kw_126[2]  [static]
Initial value:
  = {
    {"compute",8,3,2,0},kw_125,
    {"num_response_levels",13,0,1,0,0,0,0},N_mdm(num_resplevs, responseLevels)
  }

KeyWord kw_127[2]  [static]
Initial value:
  = {
    {"mt19937",8,0,1,1,0,0,0},N_mdm(lit,rngName_mt19937),
    {"rnum2",8,0,1,1,0,0,0},N_mdm(lit,rngName_rnum2)
  }

KeyWord kw_128[14]  [static]
Initial value:
  = {
    {0,0,1,0,0,0},kw_19,
    {0,0,3,0,0,0},kw_121,
    {"dakota",8,0,2,6,0,0,0},N_mdm(type,emulatorType GP_EMMULATOR),
    {"export_points_file",11,6,4,0},kw_122,0,0,0,N_mdm(str,approxExportFile),
    {"import_points_file",11,2,3,0},kw_123,0,0,0,N_mdm(str,approxImportFile),
    {"response_levels",14,2,8,0},kw_126,0,0,0,N_mdm(resplevs, responseLevels),
    {"rng",8,2,7,0},kw_127,
    {"seed",0x19,0,6,0,0,0,0},N_mdm(pint,randomSeed)},
12.1. DAKOTA NAMESPACE REFERENCE

```cpp
{"surfpack", 8, 0, 2, 0, 0, 0, 0, 0, N_mdm(type, emulatorType_KRIGING_EMULATOR)},
{"u.gaussian_process", 8, 0, 1, 0, 0, 0, 0, N_mdm(type, reliabilitySearchType_EGRA_U)},
{"u.kriging", 0, 0, 1, 0, 0, 0, 0, -1, N_mdm(type, reliabilitySearchType_EGRA_X)},
{"use_derivatives", 8, 0, 5, 0, 0, 0, 0, 0, N_mdm(true, methodUseDerivsFlag)},
{"x.gaussian_process", 8, 0, 1, 0, 0, 0, 0, N_mdm(type, reliabilitySearchType_EGRA_X)},
{"x.kriging", 0, 0, 1, 0, 0, 0, 0, -1, N_mdm(type, reliabilitySearchType_EGRA_X)}
```

**KeyWord kw129[2]** [static]

Initial value:
```
= {
   {"master", 8, 0, 1, 0, 0, 0, 0, 0, N_mdm(type, iteratorScheduling_MASTER_SCHEDULING)},
   {"peer", 8, 0, 1, 0, 0, 0, 0, 0, N_mdm(type, iteratorScheduling_PEER_SCHEDULING)}
}
```

**KeyWord kw130[3]** [static]

Initial value:
```
= {
   {"iterator_scheduling", 8, 2, 2, 0, kw129},
   {"iterator_servers", 0x19, 0, 1, 0, 0, 0, 0, 0, N_mdm(pint, iteratorServers)},
   {"processors_per_iterator", 0x19, 0, 3, 0, 0, 0, 0, 0, N_mdm(pint, procPerIterator)}
}
```

**KeyWord kw131[1]** [static]

Initial value:
```
= {
   {"model_pointer_list", 11, 0, 1, 0, 0, 0, 0, 0, N_mdm(strl, hybridModelPointers)}
}
```

**KeyWord kw132[2]** [static]

Initial value:
```
= {
   {"method_name_list", 15, 1, 1, 1, kw131, 0, 0, 0, N_mdm(strl, hybridMethodNames)},
   {"method_pointer_list", 15, 0, 1, 0, 0, 0, 0, 0, N_mdm(strl, hybridMethodPointers)}
}
```

**KeyWord kw133[1]** [static]

Initial value:
```
= {
   {"global_model_pointer", 11, 0, 1, 0, 0, 0, 0, 0, N_mdm(str, hybridGlobalModelPointer)}
}
```

**KeyWord kw134[1]** [static]

Initial value:
```
= {
   {"local_model_pointer", 11, 0, 1, 0, 0, 0, 0, 0, N_mdm(str, hybridLocalModelPointer)}
}
```
KeyWord kw_135[5]  [static]
Initial value:
= {
  "global_method_name":str,hybridGlobalMethodName,
  "global_method_pointer":str,hybridGlobalMethodPointer,
  "local_method_name":str,hybridLocalMethodName,
  "local_method_pointer":str,hybridLocalMethodPointer,
  "local_search_probability":Real,hybridLSProb
}

KeyWord kw_136[1]  [static]
Initial value:
= {
  "model_pointer_list":strL,hybridModelPointers
}

KeyWord kw_137[2]  [static]
Initial value:
= {
  "method_name_list":strL,hybridMethodNames,
  "method_pointer_list":strL,hybridMethodPointers
}

KeyWord kw_138[6]  [static]
Initial value:
= {
  0,0,3,0,0,kw_130,
  "collaborative":str,hybridMethodCollaborative,
  "coupled":strL,hybridMethodEmbedded,
  "sequential":strL,hybridMethodSequential,
  "uncoupled":strL,hybridMethodSequential,
  "parallel":str,hybridResponseParallel,
  "series":str,hybridResponseSeries
}

KeyWord kw_139[2]  [static]
Initial value:
= {
  "parallel":str,hybridResponseParallel,
  "sequential":str,hybridResponseSequential
}

KeyWord kw_140[3]  [static]
Initial value:
= {
  "gen_reliabilities":str,hybridGenReliabilities,
  "probabilities":str,hybridProbabilities,
  "system":strL,hybridSystemName
}
12.1. DAKOTA NAMESPACE REFERENCE

KeyWord kw_141[2]  [static]
Initial value:
= {
   {"compute",8,3,2,0,kw_140},
   {"num_response_levels",13,0,1,0,0,0,0,0,0,Nmdm(num_resplevs,responseLevels)}
}

KeyWord kw_142[8]  [static]
Initial value:
= {
    {0,0,1,0,0,kw_19},
    {0,0,2,0,kw_20},
    {0,0,4,0,kw_25},
    {"adapt_import",8,0,1,0,0,0,0,0,Nmdm(utype,integrationRefine_2),
     "mmadapt_import",8,0,1,0,0,0,0,0,Nmdm(utype,integrationRefine_2),
     "refinement_samples",9,0,2,0,0,0,0,0,Nmdm(int,refineSamples)},
    {"response_levels",14,2,3,0,kw_141,0,0,0,0,Nmdm(resplevs,responseLevels)}
}

KeyWord kw_143[2]  [static]
Initial value:
= {
    {"annotated",8,0,1,0,0,0,0,0,Nmdm(true,pstudyFileAnnotated)},
    {"freeform",8,0,1,0,0,0,0,0,Nmdm(false,pstudyFileAnnotated)}
}

KeyWord kw_144[3]  [static]
Initial value:
= {
    {0,0,1,0,0,kw_19},
    {"import_points_file",11,2,1,1,kw_143,0,0,0,0,Nmdm(str,pstudyFilename)},
    {"list_of_points",14,0,1,0,0,0,0,0,Nmdm(RealDL,listOfPoints)}
}

KeyWord kw_145[2]  [static]
Initial value:
= {
   {"complementary",8,0,1,1,0,0,0,0,0,Nmdm(type,distributionType_COMPLEMENTARY)},
   {"cumulative",8,0,1,1,0,0,0,0,0,Nmdm(type,distributionType_CUMULATIVE)}
}

KeyWord kw_146[1]  [static]
Initial value:
= {
   {"num_gen_reliability_levels",13,0,1,0,0,0,0,0,0,Nmdm(num_resplevs,genReliabilityLevels)}
}
KeyWord kw_147[1] [static]
Initial value:
= {
    "num_probability_levels", 13, 0, 1, 0, 0, 0, 0, 0, N, mdm(num, resplevs, probabilityLevels)
}

KeyWord kw_148[2] [static]
Initial value:
= {
    "parallel", 8, 0, 1, 1, 0, 0, 0, 0, 0, N, mdm(type, responseLevelTargetReduce, SYSTEM, PARALLEL),
    "series", 8, 0, 1, 1, 0, 0, 0, 0, N, mdm(type, responseLevelTargetReduce, SYSTEM, SERIES)
}

KeyWord kw_149[3] [static]
Initial value:
= {
    "gen_reliabilities", 8, 0, 1, 1, 0, 0, 0, 0, 0, N, mdm(type, responseLevelTarget, GEN_RELIABILITIES),
    "probabilities", 8, 0, 1, 1, 0, 0, 0, 0, N, mdm(type, responseLevelTarget, PROBABILITIES),
    "system", 8, 2, 2, 0, kw_148
}

KeyWord kw_150[2] [static]
Initial value:
= {
    "compute", 8, 3, 2, 0, kw_149,
    "num_response_levels", 13, 0, 1, 0, 0, 0, 0, 0, N, mdm(num, resplevs, responseLevels)
}

KeyWord kw_151[7] [static]
Initial value:
= {
    0, 0, 1, 0, 0, 0, kw_19,
    {"distribution", 8, 2, 5, 0, kw_145},
    {"gen_reliability_levels", 14, 1, 4, 0, kw_146, 0, 0, 0, 0, N, mdm(resplevs, genReliabilityLevels)},
    {"nip", 8, 0, 1, 0, 0, 0, 0, 0, N, mdm(utype, subMethod, SUBMETHOD, NIP)},
    {"probability_levels", 14, 1, 3, 0, kw_147, 0, 0, 0, 0, N, mdm(resplevs01, probabilityLevels)},
    {"response_levels", 14, 2, 2, 0, kw_150, 0, 0, 0, 0, N, mdm(resplevs, responseLevels)},
    {"sqp", 8, 0, 1, 0, 0, 0, 0, 0, N, mdm(utype, subMethod, SUBMETHOD, SQP)}
}

KeyWord kw_152[3] [static]
Initial value:
= {
    0, 0, 1, 0, 0, 0, kw_19,
    {"nip", 8, 0, 1, 0, 0, 0, 0, 0, N, mdm(utype, subMethod, SUBMETHOD, NIP)},
    {"sqp", 8, 0, 1, 0, 0, 0, 0, 0, N, mdm(utype, subMethod, SUBMETHOD, SQP)}
}
KeyWord kw_153[5]  [static]
Initial value:

= {
   "adapt_import", 8, 0, 1, 0, 0, 0, 0, 0, Nmdm(utype, integrationRefine_AIS),
   "import", 8, 0, 1, 0, 0, 0, 0, 0, Nmdm(utype, integrationRefine_IS),
   "mm_adapt_import", 8, 0, 1, 0, 0, 0, 0, 0, Nmdm(utype, integrationRefine_JMAIS),
   "refinement_samples", 8, 0, 2, 0, 0, 0, 0, 0, 0, Nmdm(int, refineSamples),
   "seed", 0x19, 0, 3, 0, 0, 0, 0, 0, Nmdm(pint, randomSeed)
}

KeyWord kw_154[4]  [static]
Initial value:

= {
   "first_order", 8, 0, 1, 0, 0, 0, 0, 0, Nmdm(lit, reliabilityIntegration_first_order),
   "probability_refinement", 8, 0, 2, kw_153},
   "sample_refinement", 8, 5, 2, kw_153, 0, 0, -1,
   "second_order", 8, 0, 1, 0, 0, 0, 0, Nmdm(lit, reliabilityIntegration_second_order)}

KeyWord kw_155[10]  [static]
Initial value:

= {
   "integration", 8, 4, 3, 0, kw_154,
   "nip", 8, 0, 2, 0, 0, 0, 0, 0, Nmdm(utype, subMethod_SUBMETHOD_NIP),
   "no_approx", 8, 0, 1, 0, 0, 0, 0, 0, Nmdm(utype, reliabilitySearchType_JMO_Approx),
   "sqp", 8, 0, 2, 0, 0, 0, 0, 0, 0, Nmdm(utype, subMethod_SUBMETHOD_SQP),
   "u_taylor_mean", 8, 0, 1, 0, 0, 0, 0, 0, Nmdm(utype, reliabilitySearchType_AMV_U),
   "u_taylor_mpp", 8, 0, 1, 0, 0, 0, 0, 0, 0, Nmdm(utype, reliabilitySearchType_AMV_MPP),
   "x_two_point", 8, 0, 1, 0, 0, 0, 0, 0, 0, Nmdm(utype, reliabilitySearchType_TANA_X),
   "x_two_point", 8, 0, 1, 0, 0, 0, 0, 0, 0, Nmdm(utype, reliabilitySearchType_TANA_X)}

KeyWord kw_156[1]  [static]
Initial value:

= {
   "num_reliability_levels", 13, 0, 1, 0, 0, 0, 0, 0, Nmdm(num_resevles, reliabilityLevels)
}

KeyWord kw_157[2]  [static]
Initial value:

= {
   "parallel", 8, 0, 1, 0, 0, 0, 0, 0, Nmdm(type, responseLevelTargetReduce_SYSTEM_PARALLEL),
   "series", 8, 0, 1, 0, 0, 0, 0, 0, Nmdm(type, responseLevelTargetReduce_SYSTEM_SERIES)}
KeyWord kw_158[4] [static]
Initial value:

= {
    "{gen_reliabilities",8,0,1,0,0,0.,0.,Ndm(type,responseLevelTarget,GEN_RELIABILITIES)},
    "{probabilities",8,0,1,0,0,0.,0.,Ndm(type,responseLevelTarget,PROBABILITIES)},
    "{reliabilities",8,0,1,0,0,0.,0.,0,Ndm(type,responseLevelTarget,RELIABILITIES)},
    "{system",8,2,2,0,kw_157}
}

KeyWord kw_159[2] [static]
Initial value:

= {
    "{compute",8,4,2,0,kw_158},
    "{num_response_levels",13,0,1,0,0.,0.,0,Ndm(num_resplevs,responseLevels)}
}

KeyWord kw_160[5] [static]
Initial value:

= {
    {0,0,1,0,0,kw_19},
    {0,0,3,0,0,kw_121},
    "{mpp_search",8,10,1,0,kw_155},
    "{reliability_levels",14,1,3,0,kw_156,0.,0.,0,Ndm(resplevs,relabilityLevels)},
    "{response_levels",14,2,2,0,kw_159,0.,0.,0,Ndm(resplevs,responseLevels)}
}

KeyWord kw_161[8] [static]
Initial value:

= {
    {0,0,1,0,0,kw_19},
    {0,0,9,0,0,kw_34},
    "{display_all_evaluations",8,0,6,0,0,0.,0.,0.,Ndm(true,showAllEval)},
    "{display_format",11,0,4,0,0,0.,0.,0,Ndm(str,displayFormat)},
    "{function_precision",10,0,1,0,0,0.,0.,0.,Ndm(Real,functionPrecision)},
    "{history_file",11,0,3,0,0,0,0.,0.,0,Ndm(str,historyFile)},
    "{seed",0x19,0,2,0,0,0.,0.,0,Ndm(pint,randomSeed)},
    "{variable_neighborhood_search",10,0,5,0,0,0.,0.,0,Ndm(Real,vns)}
}

KeyWord kw_162[2] [static]
Initial value:

= {
    "{num_offspring",0x19,0,2,0,0,0.,0.,0,Ndm(pintz,numOffspring)},
    "{num_parents",0x19,0,1,0,0,0.,0.,0,Ndm(pintz,numParents)}
}
12.1. DAKOTA NAMESPACE REFERENCE

KeyWord kw_163[5]  [static]

Initial value:

= {
    {"crossover_rate", 10, 0, 2, 0, 0, 0, 0, 0, N, mdm(litz, TYPE_DATA.crossoverType.null.crossover)},
    {"multi_point_binary", 9, 0, 1, 0, 0, 0, 0, N, mdm(lit2p, TYPE_DATA.crossoverType.multi_point_binary)},
    {"multi_point_parameterized_binary", 9, 0, 1, 0, 0, 0, 0, N, mdm(lit2p, TYPE_DATA.crossoverType.multi_point_parameterized_binary)},
    {"multi_point_real", 9, 0, 1, 0, 0, 0, 0, N, mdm(lit2p, TYPE_DATA.crossoverType.multi_point_real)},
    {"shuffle_random", 8, 2, 1, 1, kw_162, 0, 0, 0, N, mdm(litc, TYPE_DATA.crossoverType.shuffle_random)}
}

KeyWord kw_164[3]  [static]

Initial value:

= {
    {"flat_file", 11, 0, 1, 0, 0, 0, 0, 0, N, mdm(slit2, TYPE_DATA.initializationType.flat_file)},
    {"simple_random", 8, 0, 1, 0, 0, 0, 0, N, mdm(lit, initializationType.random)},
    {"unique_random", 8, 0, 1, 0, 0, 0, 0, N, mdm(lit, initializationType.unique_random)}
}

KeyWord kw_165[1]  [static]

Initial value:

= {
    {"mutation_scale", 10, 0, 1, 0, 0, 0, 0, N, mdm(Real01, mutationScale)}
}

KeyWord kw_166[6]  [static]

Initial value:

= {
    {"bit_random", 8, 0, 1, 0, 0, 0, 0, N, mdm(lit, mutationType.bit_random)},
    {"mutation_rate", 10, 0, 2, 0, 0, 0, 0, N, mdm(litz, TYPE_DATA.mutationType.null.mutation)},
    {"offset_cauchy", 8, 1, 1, kw_165, 0, 0, 0, N, mdm(litc, TYPE_DATA.mutationType.offset.cauchy)},
    {"offset.normal", 8, 1, 1, kw_165, 0, 0, 0, N, mdm(litc, TYPE_DATA.mutationType.offset.normal)},
    {"offset_uniform", 8, 1, 1, kw_165, 0, 0, 0, N, mdm(litc, TYPE_DATA.mutationType.offset.uniform)},
    {"replace_uniform", 8, 0, 1, 1, 0, 0, 0, 0, N, mdm(lit, mutationType.replace_uniform)}
}

KeyWord kw_167[7]  [static]

Initial value:

= {
    {"crossover_type", 8, 5, 5, 0, kw_163},
    {"initialization_type", 8, 3, 4, 0, kw_164},
    {"log_file", 11, 0, 2, 0, 0, 0, 0, N, mdm(str, logFile)},
    {"mutation_type", 8, 6, 6, 0, kw_166},
    {"population.size", 0x29, 0, 1, 0, 0, 0, 0, 0, N, mdm(nnint, populationSize)},
    {"print_each_pop", 8, 0, 3, 0, 0, 0, 0, 0, N, mdm(true, printPopFlag)},
    {"seed", 0x19, 0, 7, 0, 0, 0, 0, 0, N, mdm(pint, randomSeed)}
}
KeyWord kw_168[3]  [static]
Initial value:

```c
= {
    {"metric_tracker",8,0,1,0,0,0,0,,0,N_mdm(lit,convergenceType_metric_tracker)},
    {"num_generations",0x29,0,0,0,0,0,0,0,N_mdm(sizet,numGenerations)},
    {"percent_change",10,0,2,0,0,0,0,0,N_mdm(Realz,convergenceTolerance)}
} 
```

KeyWord kw_169[2]  [static]
Initial value:

```c
= {
    {"domination_count",8,0,1,0,0,0,0,0,N_mdm(lit,fitnessType_domination_count)},
    {"layer_rank",8,0,1,0,0,0,0,0,N_mdm(lit,fitnessType_layer_rank)}
} 
```

KeyWord kw_170[1]  [static]
Initial value:

```c
= {
    {"num_designs",0x29,0,1,0,0,2,.0,0,N_mdm(pintz,numDesigns)}
} 
```

KeyWord kw_171[3]  [static]
Initial value:

```c
= {
    {"distance",14,0,1,1,0,0,0,.0,0,N_mdm(RealLlit,TYPE_DATA_nichingType_distance)},
    {"max_designs",14,1,1,1,kw_170,0,0,0,N_mdm(RealLlit,TYPE_DATA_nichingType_max_designs)},
    {"radial",14,0,1,1,0,0,0,0,N_mdm(RealLlit,TYPE_DATA_nichingType_radial)}
} 
```

KeyWord kw_172[1]  [static]
Initial value:

```c
= {
    {"orthogonal_distance",14,0,1,1,0,0,0,.0,0,N_mdm(RealLlit,
            TYPE_DATA_postProcessorType_distance_postprocessor)}
} 
```

KeyWord kw_173[2]  [static]
Initial value:

```c
= {
    {"shrinkage_fraction",10,0,1,0,0,0,0,0,0,N_mdm(Real01,shrinkagePercent)},
    {"shrinkage_percentage",2,0,1,0,0,0,.0,-1,N_mdm(Real01,shrinkagePercent)}
} 
```
12.1. DAKOTA NAMESPACE REFERENCE

KeyWord kw_174[4] [static]
Initial value:

```c
{ "below_limit", 10, 2, 1, kw_173, 0, 0, 0, N_mdm(litp, TYPE_DATA, replacementType_below_limit) },
{ "elitist", 8, 0, 1, 1, 0, 0, 0, N_mdm(litp, replacementType_elitist) },
{ "roulette_wheel", 8, 0, 1, 1, 0, 0, 0, N_mdm(litp, replacementType_roulette_wheel) },
{ "unique_roulette_wheel", 8, 0, 1, 1, 0, 0, 0, N_mdm(litp, replacementType_unique_roulette_wheel) }
```

KeyWord kw_175[8] [static]
Initial value:

```c
{ 0, 0, 1, 0, 0, kw_18 },
{ 0, 0, 9, 0, 0, kw_34 },
{ 0, 0, 7, 0, 0, kw_167 },
{ "convergence_type", 8, 3, 4, 0, kw_168 },
{ "fitness_type", 8, 2, 1, 0, kw_169 },
{ "niching_type", 8, 3, 3, 0, kw_171 },
{ "postprocessor_type", 8, 1, 5, 0, kw_172 },
{ "replacement_type", 8, 2, 0, kw_174 }
```

KeyWord kw_176[1] [static]
Initial value:

```c
{ "model_pointer", 11, 0, 1, 0, 0, 0, 0, 0, N_mdm(str, subModelPointer) }
```

KeyWord kw_177[1] [static]
Initial value:

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

KeyWord kw_178[5] [static]
Initial value:

```c
{ 0, 0, 3, 0, 0, kw_130 },
{ "method_name", 11, 1, 1, kw_176, 0, 0, 0, 0, N_mdm(str, subMethodName) },
{ "method_pointer", 11, 0, 1, 1, 0, 0, 0, N_mdm(str, subMethodPointer) },
{ "random_starts", 5, 1, 2, 0, kw_177, 0, 0, 0, N_mdm(int, concurrentRandomJobs) },
{ "starting_point", 14, 0, 1, 0, 0, 0, 0, N_mdm(RealDL, concurrentParameterSets) }
```
CHAPTER 12. NAMESPACE DOCUMENTATION

KeyWord kw_179[2] [static]
Initial value:

= {
    {0,0,1,0,0,kw_19},
    {"partitions",13,0,1,0,0..0,0,0_Nmdm(usharray, varPartitions)}
}

KeyWord kw_180[5] [static]
Initial value:

= {
    {0,0,1,0,0,kw_19},
    {"min_boxsize_limit",10,0,2,0,0..0,0,0,0_Nmdm(Real, minBoxSize)},
    {"solution_accuracy",2,0,1,0,0..0,0,1,0_Nmdm(Real, solnTarget)},
    {"solution_target",10,0,1,0,0,0..0,0,0,0_Nmdm(Real, solnTarget)},
    {"volume_boxsize_limit",10,0,3,0,0,0..0,0,0,0_Nmdm(Real, volBoxSize)}
}

KeyWord kw_181[10] [static]
Initial value:

= {
    {0,0,1,0,0,kw_19},
    {"absolute_conv_tol",10,0,2,0,0..0,0,0,0_Nmdm(Real, absConvTol)},
    {"covariance",9,0,8,0,0..0,0,0,0_Nmdm(int, covarianceType)},
    {"false_conv_tol",10,0,6,0,0,0..0,0,0_Nmdm(Real, falseConvTol)},
    {"function_precision",10,0,1,0,0..0,0,0,0_Nmdm(Real, functionPrecision)},
    {"initial_trust_radius",10,0,7,0,0,0..0,0,0,0_Nmdm(Real, initTRRadius)},
    {"regression_diagnostics",8,0,9,0,0..0,0,0,0,0_Nmdm(true, regressDiag)},
    {"singular_conv_tol",10,0,4,0,0,0..0,0,0,0_Nmdm(Real, singConvTol)},
    {"singular_radius",10,0,5,0,0,0..0,0,0,0_Nmdm(Real, singRadius)},
    {"x_conv_tol",10,0,3,0,0,0..0,0,0,0_Nmdm(Real, xConvTol)}
}

KeyWord kw_182[2] [static]
Initial value:

= {
    {"parallel",8,0,1,1,0,0..0,0,0,0_Nmdm(type, responseLevelTargetReduce_SYSTEM_PARALLEL)},
    {"series",8,0,1,1,0,0..0,0,0,0_Nmdm(type, responseLevelTargetReduce_SYSTEM_SERIES)}
}

KeyWord kw_183[3] [static]
Initial value:

= {
    {"gen_reliabilities",8,0,1,1,0,0..0,0,0,0_Nmdm(type, responseLevelTarget_GEN_RELIABILITIES)},
    {"probabilities",8,0,1,1,0,0..0,0,0,0_Nmdm(type, responseLevelTarget_PROBABILITIES)},
    {"system",8,2,2,0,kw_192}
}
12.1. DAKOTA NAMESPACE REFERENCE

KeyWord kw_184[2] [static]
Initial value:

```cpp
= {
    {"compute", 8, 3, 2, kw_185[2]},
    {"num_response_levels", 13, 0, 1, 0, 0, 0, 0, 0, N, mdm(num_resplevs, responseLevels)}
}
```

KeyWord kw_185[4] [static]
Initial value:

```cpp
= {
    {0, 0, 1, 0, 0, kw_19[4]},
    {0, 0, 2, 0, 0, kw_20[4]},
    {0, 0, 4, 0, 0, kw_25[4]},
    {"response_levels", 14, 2, 1, 0, kw_184[2]}, 0, 0, 0, 0, N, mdm(resplevs, responseLevels)}
```

KeyWord kw_186[1] [static]
Initial value:

```cpp
= {
    {"num_reliability_levels", 13, 0, 1, 0, 0, 0, 0, 0, N, mdm(num_resplevs, reliabilityLevels)}
}
```

KeyWord kw_187[2] [static]
Initial value:

```cpp
= {
    {"parallel", 8, 0, 1, 1, 0, 0, 0, 0, 0, N, mdm(type, responseLevelTargetReduce_SYSTEM_PARALLEL)},
    {"series", 8, 0, 1, 0, 0, 0, 0, 0, N, mdm(type, responseLevelTargetReduce_SYSTEM_SERIES)}
}
```

KeyWord kw_188[4] [static]
Initial value:

```cpp
= {
    {"gen_reliabilities", 8, 0, 1, 1, 0, 0, 0, 0, 0, N, mdm(type, responseLevelTarget_GEN_RELIABILITIES)},
    {"probabilities", 8, 0, 1, 1, 0, 0, 0, 0, 0, N, mdm(type, responseLevelTarget_PROBABILITIES)},
    {"reliabilities", 8, 0, 1, 1, 0, 0, 0, 0, 0, N, mdm(type, responseLevelTarget_RELIABILITIES)},
    {"system", 8, 0, 2, 0, kw_187[2]}\}
```

KeyWord kw_189[2] [static]
Initial value:

```cpp
= {
    {"compute", 8, 4, 2, 0, kw_188[4]},
    {"num_response_levels", 13, 0, 1, 0, 0, 0, 0, 0, N, mdm(num_resplevs, responseLevels)}
}
```
KeyWord kw_190[2] [static]
Initial value:
= {
    "reliability_levels",14,1,0,kw_186,0.,0.,0.,N_mdm(resplevs,reliabilityLevels),
    "response_levels",14,2,0,kw_189,0.,0.,0.,N_mdm(resplevs,responseLevels)}

KeyWord kw_191[1] [static]
Initial value:
= {
    "fixed_seed",8,0,1,0,0.,0.,0.,N_mdm(true,fixedSeedFlag)}

KeyWord kw_192[2] [static]
Initial value:
= {
    "annotated",8,0,1,0,0.,0.,0.,N_mdm(true,approxImportAnnotated),
    "freeform",8,0,1,0,0.,0.,0.,N_mdm(false,approxImportAnnotated)}

KeyWord kw_193[1] [static]
Initial value:
= {
    "import_points_file",11,2,1,0,kw_192,0.,0.,0.,N_mdm(str,approxImportFile)}

KeyWord kw_194[3] [static]
Initial value:
= {
    "adapted",8,0,1,0,0.,0.,0.,N_mdm(type,expansionBasisType_ADAPTED_BASIS),
    "tensor_product",8,0,1,0,0.,0.,0.,N_mdm(type,expansionBasisType_TENSOR_PRODUCT_BASIS),
    "total_order",8,0,1,0,0.,0.,0.,N_mdm(type,expansionBasisType_TOTAL_ORDER_BASIS)}

KeyWord kw_195[1] [static]
Initial value:
= {
    "noise_tolerance",14,0,1,0,0.,0.,0.,N_mdm(RealDL,regressionNoiseTol)}
12.1. DAKOTA NAMESPACE REFERENCE

KeyWord kw_196[1] [static]
Initial value:
= {
   {"noise_tolerance",14,0,1,0,0,0,0,0,N_mdm(RealDL,regressionNoiseTol)}
}

KeyWord kw_197[2] [static]
Initial value:
= {
   {"l2_penalty",10,0,2,0,0,0,0,0,N_mdm(Real,regressionL2Penalty)},
   {"noise_tolerance",14,0,1,0,0,0,0,0,N_mdm(RealDL,regressionNoiseTol)}
}

KeyWord kw_198[2] [static]
Initial value:
= {
   {"equality_constrained",8,0,1,0,0,0,0,0,N_mdm(Real,lsRegressionType_EQ_CON_LS)},
   {"svd",8,0,1,0,0,0,0,0,N_mdm(Real,lsRegressionType_SVD_LS)}
}

KeyWord kw_199[1] [static]
Initial value:
= {
   {"noise_tolerance",14,0,1,0,0,0,0,0,N_mdm(RealDL,regressionNoiseTol)}
}

KeyWord kw_200[17] [static]
Initial value:
= {
   {"basis_pursuit",8,0,2,0,0,0,0,0,N_mdm(Real,regressionType_BASIS_Pursuit)},
   {"basis_pursuit_denoising",8,1,2,0,kw_195,0,0,0,N_mdm(Real,regressionType_BASIS_Pursuit_Denoising)},
   {"bpdn",0,0,2,0,0,0,0,0,N_mdm(Real,regressionType_BASIS_Pursuit_Denoising)},
   {"cross_validation",8,0,3,0,0,0,0,0,N_mdm(Real,lsRegressionType_CrossValidation)},
   {"lars",8,1,2,0,kw_196,0,0,0,0,N_mdm(Real,regressionType_LEAST_ANGLE_REGRESSION)},
   {"lasso",0,0,2,0,kw_197,0,0,0,1,N_mdm(Real,regressionType_LASSO_REGRESSION)},
   {"least_absolute_shrinkage",8,2,2,0,kw_197,0,0,0,0,N_mdm(Real,regressionType_LASSO_REGRESSION)},
   {"least_angle_regression",8,1,2,0,kw_196,0,0,0,0,N_mdm(Real,regressionType_LEAST_ANGLE_REGRESSION)},
   {"omp",8,0,2,0,kw_200,0,0,0,0,N_mdm(Real,regressionType_ORTHOGONAL_MATCH_Pursuit)},
   {"orthogonal_matching_pursuit",8,1,2,0,kw_199,0,0,0,0,N_mdm(Real,regressionType_ORTHOGONAL_MATCH_Pursuit)},
   {"ratio_order",10,0,1,0,0,0,0,0,N_mdm(Real,LSRegressionTermsRatioOrder)},{
   "reuse_points",8,0,6,0,0,0,0,0,0,N_mdm(lit,pointReuse_all)},
   {"reuse_samples",0,0,6,0,0,0,0,0,0,N_mdm(lit,pointReuse_all)},
   {"tensor_grid",8,0,5,0,0,0,0,0,0,N_mdm(true,tensorGridFlag)},
   {"use_derivatives",8,0,4,0,0,0,0,0,0,N_mdm(true,methodUseDerivsFlag)}
}
}
CHAPTER 12. NAMESPACE DOCUMENTATION

KeyWord kw_201[3] [static]
Initial value:

= {
    {
        "incremental_lhs", 8, 0, 2, 0, 0, 0, 1, 0, 0, 1, 0, 0, N_mdm(lit, expansionSampleType, incremental_lhs),
        "reuse_points", 8, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, N_mdm(lit, pointReuse_all),
        "reuse_samples", 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, N_mdm(lit, pointReuse_all)
    }
}

KeyWord kw_202[6] [static]
Initial value:

= {
    {0, 0, 1, 0, 0, 0, kw_193},
    {"basis_type", 8, 3, 2, 0, kw_194},
    {"collocation_points", 13, 17, 3, 1, kw_200, 0, 0, 0, 0, 0, 0, 0, N_mdm(szarray, collocationPoints)},
    {"collocation_ratio", 10, 17, 3, 1, kw_200, 0, 0, 0, 0, 0, 0, 0, N_mdm(Realp, collocationRatio)},
    {"dimension_preference", 14, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, N_mdm(RealDL, anisoDimPref)},
    {"expansion_samples", 13, 3, 3, 1, kw_201, 0, 0, 0, 0, 0, 0, 0, N_mdm(szarray, expansionSamples)}
}

KeyWord kw_203[2] [static]
Initial value:

= {
    {"annotated", 8, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, N_mdm(true, approxExportAnnotated),
    "freeform", 8, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, N_mdm(false, approxExportAnnotated)}
}

KeyWord kw_204[6] [static]
Initial value:

= {
    {0, 0, 1, 0, 0, 0, kw_193},
    {"collocation_points", 13, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, N_mdm(szarray, collocationPoints)},
    {"cross_validation", 8, 0, 2, 0, 0, 0, 0, 0, 0, 0, N_mdm(true, crossValidation)},
    {"reuse_points", 8, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, N_mdm(lit, pointReuse_all)},
    {"reuse_samples", 0, 0, 4, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, N_mdm(lit, pointReuse_all)},
    {"tensor_grid", 13, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, N_mdm(usharray, tensorGridOrder)}
}

KeyWord kw_205[3] [static]
Initial value:

= {
    {"decay", 8, 0, 1, 1, 0, 0, 0, 0, 0, N_mdm(type, refinementControl_DIMENSION_ADAPTIVE_CONTROL_DECAY),
    "generalized", 8, 0, 1, 1, 0, 0, 0, 0, 0, N_mdm(type, refinementControl_DIMENSION_ADAPTIVE_CONTROL_GENERALIZED),
    "sobol", 8, 0, 1, 1, 0, 0, 0, 0, 0, N_mdm(type, refinementControl_DIMENSION_ADAPTIVE_CONTROL_SOBOL)}
}
12.1. DAKOTA NAMESPACE REFERENCE

KeyWord kw_206[2] [static]
Initial value:

= {
  {"dimension_adaptive",8,3,1,1,kw_205},
  {"uniform",8,0,1,1,0,0,0,0,N_mdm(type,refinementControl,UNIFORM_CONTROL)}
}

KeyWord kw_207[4] [static]
Initial value:

= {
  {"adapt_import",8,0,1,1,0,0,0,0,N_mdm(type,integrationRefine_AIS)},
  {"import",8,0,1,1,0,0,0,0,N_mdm(type,integrationRefine_IS)},
  {"mm_adapt_import",8,0,1,1,0,0,0,0,N_mdm(type,integrationRefine_MMAIS)},
  {"refinement_samples",9,0,2,0,0,0,0,0,N_mdm(int,refineSamples)}
}

KeyWord kw_208[3] [static]
Initial value:

= {
  {"dimension_preference",14,0,1,0,0,0,0,0,N_mdm(RealDL,anisoDimPref)},
  {"nested",8,0,2,0,0,0,0,0,N_mdm(type,nestingOverride,NESTED)},
  {"non_nested",8,0,2,0,0,0,0,0,N_mdm(type,nestingOverride,NON_NESTED)}
}

KeyWord kw_209[2] [static]
Initial value:

= {
  {"lhs",8,0,1,1,0,0,0,0,N_mdm(type,sampleType,SUBMETHOD_LHS)},
  {"random",8,0,1,1,0,0,0,0,N_mdm(type,sampleType,SUBMETHOD_RANDOM)}
}

KeyWord kw_210[3] [static]
Initial value:

= {
  {0,0,3,0,0,kw_208},
  {"restricted",8,0,1,0,0,0,0,0,N_mdm(type,growthOverride,RESTRICTED)},
  {"unrestricted",8,0,1,0,0,0,0,0,N_mdm(type,growthOverride,UNRESTRICTED)}
}

KeyWord kw_211[2] [static]
Initial value:

= {
  {"drop_tolerance",10,0,2,0,0,0,0,0,N_mdm(Real,vbdDropTolerance)},
  {"interaction_order",0x19,0,1,0,0,0,0,0,N_mdm(ushint,vbdOrder)}
}
### KeyWord `kw_212[25]` [static]

Initial value:

```plaintext
= {
    {0,0,1,0,0,kw_19},
    {0,0,2,0,0,kw_20},
    {0,0,4,0,0,kw_25},
    {0,0,2,0,0,kw_190},
    {0,0,1,0,0,kw_191},
    {"askey",8,0,2,0,0,0,0,0,0,N,mdm(type,expansionType,ASKEY)},
    {"cubature_integrand",3,0,1,0,0,0,0,0,0,N,mdm(ushint,cubIntOrder)},
    {"diagonal_covariance",8,0,5,0,0,0,0,0,0,N,mdm(type,covarianceControl,DIAONAL,COVARIANCE)},
    {"expansion_order",13,5,3,1,0,0,0,0,0,N,mdm(usharray,expansionOrder)},
    {"full_covariance",8,0,5,0,0,0,0,0,0,N,mdm(type,covarianceControl,FULL,COVARIANCE)},
    {"import_expansion_file",11,0,3,1,0,0,0,0,0,N,mdm(str,expansionImportFile)},
    {"least_interpolation",0,5,3,1,0,0,0,0,0,N,mdm(type,regressionType,ORTHOG,LEAST,INTERPOLATION)},
    {"normalized",8,0,6,0,0,0,0,0,0,N,mdm(true,normalizedCoeffs)},
    {"oli",0,5,3,1,kw_204,0,0,0,0,N,mdm(type,regressionType,ORTHOG,LEAST,INTERPOLATION)},
    {"orthogonal_least_interpolation",8,5,3,1,kw_204,0,0,0,0,N,mdm(type,regressionType,ORTHOG,LEAST,INTERPOLATION)},
    {"p_refinement",8,2,1,0,kw_206,0,0,0,0,N,mdm(type,refinementType,P,REFINEMENT)},
    {"probability_refinement",8,4,8,0,kw_207},
    {"quadrature_order",13,3,1,kw_208,0,0,0,0,N,mdm(usharray,quadratureOrder)},
    {"sample_refinement",0,4,8,0,kw_207,0,0,0,-2},
    {"sample_type",8,2,1,0,kw_209},
    {"sparse_grid_level",13,2,3,1,kw_210,0,0,0,0,N,mdm(usharray,sparseGridLevel)},
    {"variance_based_decomp",8,2,4,0,kw_211,0,0,0,0,N,mdm(true,vbdFlag)},
    {"wiener",8,0,2,0,0,0,0,0,0,N,mdm(type,expansionType,STD,NORMAL,U)}
}
```

### KeyWord `kw_213[1]` [static]

Initial value:

```plaintext
= {
    {"previous_samples",9,0,1,1,0,0,0,0,0,N,mdm(int,previousSamples)}
}
```

### KeyWord `kw_214[4]` [static]

Initial value:

```plaintext
= {
    {"incremental_lhs",8,1,1,1,kw_213,0,0,0,0,N,mdm(utype,sampleType,INCREMENTAL,LHS)},
    {"incremental_random",8,1,1,1,kw_213,0,0,0,0,N,mdm(utype,sampleType,INCREMENTAL,RANDOM)},
    {"lhs",8,0,1,1,0,0,0,0,0,N,mdm(utype,sampleType,LMETHOD,LHS)},
    {"random",8,0,1,1,0,0,0,0,0,N,mdm(utype,sampleType,LMETHOD,RANDOM)}
}
```

### KeyWord `kw_215[1]` [static]

Initial value:

```plaintext
= {
    {"drop_tolerance",10,0,1,0,0,0,0,0,0,N,mdm(Real,vbdDropTolerance)}
}
```
KeyWord kw_216[7] [static]  
Initial value:  
= {
    {0,0,1,0,0,kw_19},
    {0,0,2,0,0,kw_20},
    {0,0,4,0,0,kw_25},
    {0,0,2,0,0,kw_190},
    {0,0,1,0,0,kw_191},
    {"sample_type",8,4,1,0,kw_214},
    {"variance_based_decomp",8,1,2,0,kw_215,0,,0,N_{mdm}(true,vbdFlag)}
}

KeyWord kw_217[2] [static]  
Initial value:  
= {
    {"annotated",8,0,1,0,0,,0,N_{mdm}(true,approxExportAnnotated)},
    {"freeform",8,0,1,0,0,,0,N_{mdm}(false,approxExportAnnotated)}
}

KeyWord kw_218[2] [static]  
Initial value:  
= {
    {"generalized",8,0,1,1,0,0,,0,N_{mdm}(type,refinementControl_DIMENSION_ADAPTIVE_CONTROL_GENERALIZED)},
    {"sobol",8,0,1,1,0,0,,0,N_{mdm}(type,refinementControl_DIMENSION_ADAPTIVE_CONTROL_SOBOL)}
}

KeyWord kw_219[3] [static]  
Initial value:  
= {
    {"dimension_adaptive",8,2,1,1,kw_218},
    {"local_adaptive",8,0,1,1,0,0,,0,N_{mdm}(type,refinementControl_LOCAL_ADAPTIVE_CONTROL)},
    {"uniform",8,0,1,1,0,0,,0,N_{mdm}(type,refinementControl_UNIFORMCONTROL)}
}

KeyWord kw_220[2] [static]  
Initial value:  
= {
    {"generalized",8,0,1,1,0,0,,0,N_{mdm}(type,refinementControl_DIMENSION_ADAPTIVE_CONTROL_GENERALIZED)},
    {"sobol",8,0,1,1,0,0,,0,N_{mdm}(type,refinementControl_DIMENSION_ADAPTIVE_CONTROL_SOBOL)}
}

KeyWord kw_221[2] [static]  
Initial value:  
= {
    {"dimension_adaptive",8,2,1,1,kw_220},
    {"uniform",8,0,1,1,0,0,,0,N_{mdm}(type,refinementControl_UNIFORMCONTROL)}
}
KeyWord kw_222[4] [static]
Initial value:
= {
    "adapt_import",8,0,1,0,0,0,0,0,Nmdm(utype,integrationRefineAIS),
    "import",8,0,1,0,0,0,0,0,Nmdm(utype,integrationRefineAIS),
    "mm_adapt_import",8,0,1,0,0,0,0,0,Nmdm(utype,integrationRefineMAIS),
    "refinement_samples",9,0,2,0,0,0,0,0,Nmdm(int,refineSamples)
}

KeyWord kw_223[2] [static]
Initial value:
= {
    "lhs",8,0,1,0,0,0,0,0,Nmdm(utype,sampleType_SUBMETHOD_LHS),
    "random",8,0,1,0,0,0,0,0,Nmdm(utype,sampleType_SUBMETHOD_RANDOM)
}

KeyWord kw_224[4] [static]
Initial value:
= {
    "hierarchical",8,0,2,0,0,0,0,0,Nmdm(type,expansionBasisType_HIERARCHICAL_INTERPOLANT),
    "nodal",8,0,2,0,0,0,0,0,Nmdm(type,expansionBasisType_NODAL_INTERPOLANT),
    "restricted",8,0,1,0,0,0,0,0,Nmdm(type,growthOverride_RESTRICTED),
    "unrestricted",8,0,1,0,0,0,0,0,Nmdm(type,growthOverride_UNRESTRICTED)
}

KeyWord kw_225[2] [static]
Initial value:
= {
    "drop_tolerance",10,0,2,0,0,0,0,0,Nmdm(Real,vbdDropTolerance),
    "interaction_order",0x19,0,1,0,0,0,0,0,Nmdm(ushint,vbdOrder)
}

KeyWord kw_226[23] [static]
Initial value:
= {
    0,0,1,0,0,kw_19,
    0,0,2,0,0,kw_20,
    0,0,4,0,0,kw_25,
    0,0,2,0,0,kw_190,
    0,0,1,0,0,kw_191,
    "askey",8,0,2,0,0,0,0,0,Nmdm(type,expansionType_ASKEY_U),
    "diagonal_covariance",8,0,8,0,0,0,0,0,Nmdm(type,covarianceControl_DIAGONAL_COVARIANCE),
    "dimension_preference",14,0,4,0,0,0,0,0,Nmdm(RealDL,anisoDimPref),
    "export_points_file",11,2,11,0,kw_217,0,0,0,0,Nmdm(str,approxExportFile),
    "full_covariance",8,0,8,0,0,0,0,0,Nmdm(type,covarianceControl_FULL_COVARIANCE),
    "h_refinement",8,3,1,0,kw_219,0,0,0,0,Nmdm(type,refinementType_HREFINEMENT),
    "nested",8,0,6,0,0,0,0,0,Nmdm(type,nestingOverride_NESTED),
    "non_nested",8,0,6,0,0,0,0,0,Nmdm(type,nestingOverride_NON_NESTED),
    "p_refinement",8,2,1,0,kw_221,0,0,0,0,Nmdm(type,refinementType_PREFINEMENT),
    "piecewise",8,0,2,0,0,0,0,0,NIDRProblemDescDB::method::piecewise,
    "probability_refinement",8,4,10,0,kw_222"}
12.1. DAKOTA NAMESPACE REFERENCE

```c++
{"quadrature_order",13,0,3,1,0,0,0,0,N,mdm(usharray,quadratureOrder),
"sample.refinement",0,4,10,0,kw,222,0,0,-2},
"sample_type",8,2,0,0,kw,223},
"sparse_grid_level",13,4,3,1,kw,224,0,0,0,N,mdm(usharray,sparseGridLevel),
"use_derivatives",8,0,5,0,0,0,0,0,N,mdm(true,methodUseDerivsFlag),
"variance_based_decomp",8,2,7,0,kw,225,0,0,0,N,mdm(true,vbdFlag),
"wiener",8,0,2,0,0,0,0,0,N,mdm(type,expansionType,STD_NORMAL,0)}
```

**KeyWord kw_227[2]** [static]
Initial value:

```c++
= {
   {0,0,1,0,0,kw,19},
   {"misc_options",15,0,1,0,0,0,0,0,N,mdm(strL,miscOptions)}
}
```

**KeyWord kw_228[3]** [static]
Initial value:

```c++
= {
   {"function_precision",10,0,2,0,0,0,0,0,N,mdm(Real,functionPrecision)},
   {"linesearch_tolerance",10,0,3,0,0,0,0,0,N,mdm(Real,lineSearchTolerance)},
   {"verify_level",9,0,1,0,0,0,0,0,N,mdm(int,verifyLevel)}
}
```

**KeyWord kw_229[4]** [static]
Initial value:

```c++
= {
   {0,0,1,0,0,kw,19},
   {0,0,9,0,0,kw,34},
   {0,0,3,0,0,kw,226},
   {""}
}
```

**KeyWord kw_230[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)}
}
```

**KeyWord kw_231[4]** [static]
Initial value:

```c++
= {
   {0,0,1,0,0,kw,19},
   {0,0,9,0,0,kw,34},
   {0,0,2,0,0,kw,230},
   {""}
}
```
KeyWord kw_232[3] [static]
Initial value:

= {
    {0,0,1,0,0,kw_19},
    {0,0,9,0,0,kw_34},
    {"search_scheme_size",9,0,1,0,0,0.,0,N,mдоб(int,searchSchemeSize)}
}

KeyWord kw_233[3] [static]
Initial value:

= {
    {"argaez_tapia",8,0,1,1,0,.0,.0,N,mдоб(type,meritFn_ArgaezTapia)},
    {"el_bakry",8,0,1,1,0,.0,.0,N,mдоб(type,meritFn_NormFmu)},
    {"van_shanno",8,0,1,1,0,.0,.0,N,mдоб(type,meritFn_VanShanno)}
}

KeyWord kw_234[4] [static]
Initial value:

= {
    {"gradient_based_line_search",8,0,1,1,0,.0,.0,N,mдоб(lit, searchMethod_gradient_based_line_search)},
    {"tr_pds",8,0,1,1,0,.0,.0,N,mдоб(lit,searchMethod_tr_pds)},
    {"trust_region",8,0,1,1,0,.0,.0,N,mдоб(lit,searchMethod_trust_region)},
    {"value_based_line_search",8,0,1,1,0,.0,.0,N,mдоб(lit,searchMethod_value_based_line_search)}
}

KeyWord kw_235[7] [static]
Initial value:

= {
    {0,0,1,0,0,kw_19},
    {0,0,9,0,0,kw_34},
    {0,0,2,0,0,kw_230},
    {"centering_parameter",10,0,4,0,0,.0,.0,N,mдоб(Real,centeringParam)},
    {"merit_function",8,3,2,0,kw_233},
    {"search_method",8,4,1,0,kw_234},
    {"steplength_to_boundary",10,0,3,0,0,.0,.0,N,mдоб(Real,stepLenToBoundary)}
}

KeyWord kw_236[5] [static]
Initial value:

= {
    {"debug",8,0,1,1,0,.0,.0,N,mдоб(type,methodOutput_DEBUG_OUTPUT)},
    {"normal",8,0,1,1,0,.0,.0,N,mдоб(type,methodOutput_NORMAL_OUTPUT)},
    {"quiet",8,0,1,1,0,.0,.0,N,mдоб(type,methodOutput_QUIET_OUTPUT)},
    {"silent",8,0,1,1,0,.0,.0,N,mдоб(type,methodOutput_SILENT_OUTPUT)},
    {"verbose",8,0,1,1,0,.0,.0,N,mдоб(type,methodOutput_VERBOSE_OUTPUT)}
}
KeyWord kw_237[2] [static]
Initial value:
= {
    "model_pointer", 11, 0, 1, 0, 0, 0, 0, 0, N_mdmd(str, subModelPointer),
    "opt_model_pointer", 3, 0, 1, 0, 0, 0, -1, N_mdmd(str, subModelPointer)
}

KeyWord kw_238[1] [static]
Initial value:
= {
    "seed", 9, 0, 1, 0, 0, 0, 0, N_mdmd(int, randomSeed)
}

KeyWord kw_239[8] [static]
Initial value:
= {
    0, 0, 3, 0, 0, kw_130,
    {"method_name", 11, 2, 1, kw_237, 0, 0, 0, 0, N_mdmd(str, subMethodName),
     "method_pointer", 11, 0, 1, 0, 0, 0, 0, N_mdmd(str, subMethodPointer),
     "multi_objective_weight_sets", 6, 0, 3, 0, 0, 0, 0, 4, N_mdmd(RealDL, concurrentParameterSets),
     "opt_method_name", 3, 2, 1, 1, kw_237, 0, 0, -3, N_mdmd(str, subMethodName),
     "opt_method_pointer", 3, 0, 1, 0, 0, 0, -3, N_mdmd(str, subMethodPointer),
     "random_weight_sets", 9, 1, 2, 0, kw_238, 0, 0, 0, N_mdmd(int, concurrentRandomJobs),
     "weight_sets", 14, 0, 3, 0, 0, 0, 0, 0, N_mdmd(RealDL, concurrentParameterSets)
}

KeyWord kw_240[3] [static]
Initial value:
= {
    0, 0, 1, 0, 0, kw_19,
    0, 0, 2, 0, 0, kw_20,
    {"partitions", 13, 0, 1, 0, 0, 0, 0, 0, N_mdmd(usharray, varPartitions)}
}

KeyWord kw_241[5] [static]
Initial value:
= {
    0, 0, 1, 0, 0, kw_19,
    "converge_order", 8, 0, 1, 1, 0, 0, 0, 0, N_mdmd(utype, subMethod_SUBMETHOD_CONVERGE_ORDER),
    "converge_qoi", 8, 0, 1, 1, 0, 0, 0, 0, N_mdmd(utype, subMethod_SUBMETHOD_CONVERGE_QOI),
    "estimate_order", 8, 0, 1, 1, 0, 0, 0, 0, N_mdmd(utype, subMethod_SUBMETHOD_ESTIMATE_ORDER),
    "refinement_rate", 10, 0, 2, 0, 0, 0, 0, 0, N_mdmd(Real, refinementRate)
}

KeyWord kw_242[2] [static]
Initial value:
= {
    "num_generations", 0x29, 0, 2, 0, 0, 0, 0, 0, N_mdmd(sizet, numGenerations),
    "percent_change", 10, 0, 1, 0, 0, 0, 0, 0, N_mdmd(Realz, convergenceTolerance)
}
KeyWord kw_243[2] [static]
Initial value:

```c
= {
    {"num_generations",0x29,0,0,0,0,0,0,0,Nmdm(sizet,numGenerations)},
    {"percent_change",10,0,1,0,0,0,0,0,Nmdm(Realz,convergenceTolerance)}
}
```

KeyWord kw_244[2] [static]
Initial value:

```c
= {
    {"average_fitness_tracker",8,2,1,1,kw_242,0,0,0,0,Nmdm(lit,
      convergenceType_average_fitness_tracker)},
    {"best_fitness_tracker",8,2,1,1,kw_243,0,0,0,0,Nmdm(lit,
      convergenceType_best_fitness_tracker)}
}
```

KeyWord kw_245[2] [static]
Initial value:

```c
= {
    {"constraint_penalty",10,0,2,0,0,0,0,0,Nmdm(Realp,constraintTolerance)},
    {"merit_function",8,0,1,1,0,0,0,0,Nmdm(lit,fitnessType_merit_function)}
}
```

KeyWord kw_246[4] [static]
Initial value:

```c
= {
    {"elitist",8,0,1,1,0,0,0,0,Nmdm(lit,replacementType_elitist)},
    {"favor_feasible",8,0,1,1,0,0,0,0,Nmdm(lit,replacementType_favor_feasible)},
    {"roulette_wheel",8,0,1,1,0,0,0,0,Nmdm(lit,replacementType_roulette_wheel)},
    {"unique_roulette_wheel",8,0,1,1,0,0,0,0,Nmdm(lit,replacementType_unique_roulette_wheel)}
}
```

KeyWord kw_247[6] [static]
Initial value:

```c
= {
    {0,0,1,0,0,kw_19},
    {0,0,0,0,0,kw_34},
    {0,0,7,0,0,kw_167},
    {"convergence_type",8,2,3,0,kw_244},
    {"fitness_type",8,2,1,0,kw_245},
    {"replacement_type",8,1,2,0,kw_246}
}
```
KeyWord kw_248[5]  [static]
Initial value:
= {
  {0,0,1,0,0,kw},
  {0,0,9,0,0,kw},
  {0,0,3,0,0,kw},
  {*nlssol*,8,0,1,1,0,0.,0.,0.,N_{mdm}(type,methodName,NLSSOL_SQP)},
  {*npsol*,8,0,1,1,0,0.,0.,0.,N_{mdm}(type,methodName,NPSOL_SQP)}
}

KeyWord kw_249[7]  [static]
Initial value:
= {
  {approx_method_name*,3,0,1,1,0,0.,0.,0.,3,N_{mdm}(str,subMethodName)},
  {approx_method_pointer*,3,0,1,1,0,0.,0.,0.,3,N_{mdm}(str,subMethodPointer)},
  {method_name*,11,0,1,1,0,0.,0.,0.,N_{mdm}(str,subMethodName)},
  {method_pointer*,11,0,1,1,0,0.,0.,0.,N_{mdm}(str,subMethodPointer)},
  {replace_points*,8,0,3,0,0.,0.,0.,0.,N_{mdm}(true,surrBasedGlobalReplacePts)}
}

KeyWord kw_250[2]  [static]
Initial value:
= {
  {filter*,8,0,1,1,0,0.,0.,0.,0.,N_{mdm}(type,surrBasedLocalAcceptLogic_FILTER)},
  {tr_ratio*,8,0,1,1,0,0.,0.,0.,0.,N_{mdm}(type,surrBasedLocalAcceptLogic_TR_RATIO)}
}

KeyWord kw_251[7]  [static]
Initial value:
= {
  {augmented_lagrangian_objective*,8,0,1,1,0,0.,0.,0.,0.,N_{mdm}(type,}
   {surrBasedLocalSubProbObj,AUGMENTED_LAGRANGIAN_OBJECTIVE)},
  {lagrangian_objective*,8,0,1,1,0,0.,0.,0.,0.,N_{mdm}(type,}
   {surrBasedLocalSubProbObj,LAGRANGIAN_OBJECTIVE)},
  {linearized_constraints*,8,0,2,2,0,0.,0.,0.,0.,N_{mdm}(type,}
   {surrBasedLocalSubProbCon,LINEARIZED_CONSTRAINTS)},
  {no_constraints*,8,0,2,2,0,0.,0.,0.,0.,N_{mdm}(type,}
   {surrBasedLocalSubProbCon,NO_CONSTRAINTS)},
  {original_constraints*,8,0,2,2,0,0.,0.,0.,0.,N_{mdm}(type,}
   {surrBasedLocalSubProbCon,ORIGINAL_CONSTRAINTS)},
  {original_primary*,8,0,1,1,0,0.,0.,0.,0.,N_{mdm}(type,}
   {surrBasedLocalSubProbObj,ORIGINAL_PRIMARY)},
  {single_objective*,8,0,1,1,0,0.,0.,0.,0.,N_{mdm}(type,}
   {surrBasedLocalSubProbObj,SINGLE_OBJECTIVE)}
}

KeyWord kw_252[1]  [static]
Initial value:
= {
  {homotopy*,8,0,1,1,0,0.,0.,0.,0.,N_{mdm}(type,surrBasedLocalConstrRelax,HOMOTOPIY)}
}
**KeyWord kw_253[4]**  [static]  
Initial value:

```
{"adaptive_penaltyMerit",8,0,1,1,0,0,0,0,N\_mdm(type,
surrBasedLocalMeritFn\_ADAPTIVE\_PENALTY\_MERIT)}
{"augmented_lagrangian_merit",8,0,1,1,0,0,0,0,N\_mdm(type,
surrBasedLocalMeritFn\_ADAPTED\_LAGRANGIAN\_MERIT)}
{"lagrangian_merit",8,0,1,1,0,0,0,0,N\_mdm(type,surrBasedLocalMeritFn\_LAGRANGIAN\_MERIT)}
{"penalty_merit",8,0,1,1,0,0,0,0,N\_mdm(type,surrBasedLocalMeritFn\_PENALTY\_MERIT)}
```

**KeyWord kw_254[6]**  [static]  
Initial value:

```
{"contract_threshold",10,0,3,0,0,0,0,0,N\_mdm(Real,surrBasedLocalTRContractTrigger)}
{"expansion_factor",10,0,6,0,0,0,0,0,N\_mdm(Real,surrBasedLocalTRExpand)}
{"initial_size",10,0,1,0,0,0,0,0,N\_mdm(Real,surrBasedLocalTRInitSize)}
{"minimum_size",10,0,2,0,0,0,0,0,N\_mdm(Real,surrBasedLocalTRMinSize)}
```

**KeyWord kw_255[13]**  [static]  
Initial value:

```
{"acceptance_logic",8,2,8,0,kw_250},
{"approx_method_name",3,0,1,1,0,0,0,0,6,N\_mdm(str,subMethodName)},
{"approx_method_pointer",3,0,1,1,0,0,0,0,6,N\_mdm(str,subMethodPointer)},
{"approx_model_pointer",3,0,2,2,0,0,0,0,6,N\_mdm(str,modelPointer)},
{"approx_subproblem",8,7,6,0,kw_251},
{"constraint_relax",8,1,9,0,kw_252},
{"merit_function",8,4,7,0,kw_253},
{"method_name",11,0,1,1,0,0,0,0,6,N\_mdm(str,subMethodName)},
{"method_pointer",11,0,1,1,0,0,0,0,6,N\_mdm(str,subMethodPointer)},
{"model_pointer",11,0,2,2,0,0,0,0,0,N\_mdm(str,modelPointer)},
{"soft_convergence_limit",9,0,3,0,0,0,0,0,0,N\_mdm(int,surrBasedLocalSoftConvLimit)},
{"trust_region",8,6,5,0,kw_254,0,0,0,0,0,NIDRProblemDescDB\_method_tr\_final},
{"truth_surrogate_bypass",8,0,4,0,0,0,0,0,0,N\_mdm(true,surrBasedLocalLayerBypass)}
```

**KeyWord kw_256[4]**  [static]  
Initial value:

```
{"final_point",14,0,1,1,0,0,0,0,N\_mdm(RealDL,finalPoint)},
{"num_steps",9,0,2,2,0,0,0,0,0,N\_mdm(int,numSteps)},
{"step_vector",14,0,1,1,0,0,0,0,0,N\_mdm(RealDL,stepVector)}
```

**KeyWord kw_258[1]**  [static]  
Initial value:

```
{"optional_interface_responses_pointer",11,0,1,0,0,0,0,0,0,N\_mom(str,
optionalInterfRespPointer)}
```
12.1. DAKOTA NAMESPACE REFERENCE

KeyWord kw_259[2] [static]
Initial value:
= {
    {"master"},8,0,1,0,0,0,0,0,N_mom(type,subMethodScheduling,MASTER_SCHEDULING)},
    {"peer"},8,0,1,0,0,0,0,0,N_mom(type,subMethodScheduling,PEER_SCHEDULING)}

KeyWord kw_260[7] [static]
Initial value:
= {
    {"iterator_scheduling"},8,2,2,0,kw_259},
    {"iterator_servers"},0x19,0,1,0,0,0,0,0,N_mom(pint,subMethodServers)},
    {"primary_response_mapping"},14,0,6,0,0,0,0,0,N_mom(RealDL,primaryRespCoeffs)},
    {"primary_variable_mapping"},15,0,4,0,0,0,0,0,N_mom(strl,primaryVarMaps)},
    {"processors_per_iterator"},0x19,0,3,0,0,0,0,0,N_mom(pint,subMethodProcs)},
    {"secondary_response_mapping"},14,0,7,0,0,0,0,0,N_mom(RealDL,secondaryRespCoeffs)},
    {"secondary_variable_mapping"},15,0,5,0,0,0,0,0,N_mom(strl,secondaryVarMaps)}

KeyWord kw_261[2] [static]
Initial value:
= {
    {"optional_interface_pointer"},11,1,1,0,kw_258,0,0,0,N_mom(str,interfacePointer)},
    {"sub_method_pointer"},11,7,2,1,kw_260,0,0,0,N_mom(str,subMethodPointer)}

KeyWord kw_262[1] [static]
Initial value:
= {
    {"interface_pointer"},11,0,1,0,0,0,0,0,N_mom(str,interfacePointer)}

KeyWord kw_263[2] [static]
Initial value:
= {
    {"annotated"},8,0,1,0,0,0,0,0,N_mom(true,approxChallengeAnnotated)},
    {"freeform"},8,0,1,0,0,0,0,0,N_mom(false,approxChallengeAnnotated)}

KeyWord kw_264[6] [static]
Initial value:
= {
    {"additive"},8,0,2,2,0,0,0,0,N_mom(type,approxCorrectionType,ADDITIVE_CORRECTION)},
    {"combined"},8,0,2,2,0,0,0,0,N_mom(type,approxCorrectionType,COMBINED_CORRECTION)},
    {"first_order"},8,0,1,1,0,0,0,0,N_mom(order,approxCorrectionOrder,1)},
    {"multiplicative"},8,0,2,2,0,0,0,0,N_mom(type,approxCorrectionType,MULTIPLICATIVE_CORRECTION)},
    {"second_order"},8,0,1,1,0,0,0,0,N_mom(order,approxCorrectionOrder,2)},
    {"zeroth_order"},8,0,1,1,0,0,0,0,N_mom(order,approxCorrectionOrder,0)}
KeyWord kw_265[2]  [static]
Initial value:
= {
    {"folds",9,0,1,0,0.,0.,0.,0.,0.,mom(int,numFolds)},
    {"percent",10,0,1,0,0.,0.,0.,0.,0.,N,mom(Real,percentFold)}
}

KeyWord kw_266[2]  [static]
Initial value:
= {
    {"cross_validate",8,2,1,0,kw_265,0.,0.,0.,0.,N,mom(true,crossValidateFlag)},
    {"press",8,0,2,0,0.,0.,0.,0.,0.,N,mom(true,pressFlag)}
}

KeyWord kw_267[2]  [static]
Initial value:
= {
    {"annotated",8,0,1,0,0.,0.,0.,0.,0.,N,mom(true,approxExportAnnotated)},
    {"freeform",8,0,1,0,0.,0.,0.,0.,0.,N,mom(false,approxExportAnnotated)}
}

KeyWord kw_268[3]  [static]
Initial value:
= {
    {"constant",8,0,1,1,0,0.,0.,0.,0.,N,mom(lit,trendOrder_constant)},
    {"linear",8,0,1,1,0,0.,0.,0.,0.,N,mom(lit,trendOrder_linear)},
    {"reduced_quadratic",8,0,1,1,0,0.,0.,0.,0.,N,mom(lit,trendOrder_reduced_quadratic)}
}

KeyWord kw_269[2]  [static]
Initial value:
= {
    {"point_selection",8,0,1,0,0.,0.,0.,0.,0.,N,mom(true,pointSelection)},
    {"trend",8,3,2,0,kw_268}
}

KeyWord kw_270[4]  [static]
Initial value:
= {
    {"constant",8,0,1,1,0,0.,0.,0.,0.,N,mom(lit,trendOrder_constant)},
    {"linear",8,0,1,1,0,0.,0.,0.,0.,N,mom(lit,trendOrder_linear)},
    {"quadratic",8,0,1,1,0,0.,0.,0.,0.,N,mom(lit,trendOrder_quadratic)},
    {"reduced_quadratic",8,0,1,1,0,0.,0.,0.,0.,N,mom(lit,trendOrder_reduced_quadratic)}
}
KeyWord kw_271[7]  [static]
Initial value:
= {
   {"correlation_lengths",14,0,5,0,0,0,0,0,0,Nmom(RealDL,krigingCorrelations)},
   {"export_model_file",11,0,6,0,0,0,0,0,0,Nmom(str, approxExportModelFile)},
   {"find_nugget",9,0,4,0,0,0,0,0,0,Nmom(shint,krigingFindNugget)},
   {"max_trials",0x19,0,3,0,0,0,0,0,0,Nmom(shint,krigingMaxTrials)},
   {"nugget",0x1a,0,4,0,0,0,0,0,0,Nmom(Real,krigingNugget)},
   {"optimization_method",11,0,2,0,0,0,0,0,0,Nmom(str,krigingOptMethod)},
   {"trend",8,4,1,0,kw_270}
}

KeyWord kw_272[2]  [static]
Initial value:
= {
   {"dakota",8,2,1,1,kw_269,0,0,0,0,0,Nmom(lit,surrogateType,global,gaussian)},
   {"surfpack",8,7,1,1,kw_271,0,0,0,0,0,Nmom(lit,surrogateType,global,kriging)}
}

KeyWord kw_273[2]  [static]
Initial value:
= {
   {"annotated",8,0,1,0,0,0,0,0,0,0,Nmom(true,approxImportAnnotated)},
   {"freeform",8,0,1,0,0,0,0,0,0,0,Nmom(false,approxImportAnnotated)}
}

KeyWord kw_274[2]  [static]
Initial value:
= {
   {"cubic",8,0,1,1,0,0,0,0,0,0,Nmom(lit,marsInterpolation,cubic)},
   {"linear",8,0,1,1,0,0,0,0,0,0,Nmom(lit,marsInterpolation,linear)}
}

KeyWord kw_275[3]  [static]
Initial value:
= {
   {"export_model_file",11,0,3,0,0,0,0,0,0,Nmom(str, approxExportModelFile)},
   {"interpolation",8,2,2,0,kw_274},
   {"max_bases",9,0,1,0,0,0,0,0,0,Nmom(shint,marsMaxBases)}
}

KeyWord kw_276[3]  [static]
Initial value:
= {
   {"export_model_file",11,0,3,0,0,0,0,0,0,Nmom(str, approxExportModelFile)},
   {"poly_order",9,0,1,0,0,0,0,0,0,Nmom(shint,mlsPolyOrder)},
   {"weight_function",9,0,2,0,0,0,0,0,0,Nmom(shint,mlsWeightFunction)}
}
CHAPTER 12. NAMESPACE DOCUMENTATION

KeyWord kw_277[4]  [static]
Initial value:
= {
  {"export_model_file",11,0,4,0,0,0,0,0,Nmom(str, approxExportModelFile)},
  {"nodes",9,0,1,0,0,0,0,0,Nmom(shint,annNodes)},
  {"random_weight",9,0,1,0,0,0,0,0,Nmom(shint,annRandomWeight)},
  {"range",10,0,2,0,0,0,0,0,Nmom(Real,annRange)}
}

KeyWord kw_278[4]  [static]
Initial value:
= {
  {"cubic",8,0,1,0,0,0,0,0,Nmom(order,polynomialOrder3)},
  {"export_model_file",11,0,2,0,0,0,0,0,Nmom(str, approxExportModelFile)},
  {"linear",8,0,1,0,0,0,0,0,Nmom(order,polynomialOrder1)},
  {"quadratic",8,0,1,0,0,0,0,0,Nmom(order,polynomialOrder2)}
}

KeyWord kw_279[5]  [static]
Initial value:
= {
  {"bases",9,0,1,0,0,0,0,0,Nmom(shint,rbfBases)},
  {"export_model_file",11,0,5,0,0,0,0,0,Nmom(str, approxExportModelFile)},
  {"max_pts",9,0,2,0,0,0,0,0,Nmom(shint,rbfMaxPts)},
  {"max_subsets",9,0,4,0,0,0,0,0,Nmom(shint,rbfMaxSubsets)},
  {"min_partition",9,0,3,0,0,0,0,0,Nmom(shint,rbfMinPartition)}
}

KeyWord kw_280[3]  [static]
Initial value:
= {
  {"all",8,0,1,0,0,0,0,0,Nmom(lit,approxPointReuse_all)},
  {"none",8,0,1,0,0,0,0,0,Nmom(lit,approxPointReuse_none)},
  {"region",8,0,1,0,0,0,0,0,Nmom(lit,approxPointReuse_region)}
}

KeyWord kw_281[21]  [static]
Initial value:
= {
  {"challenge_points_file",11,2,10,0,kw_263,0,0,0,0,Nmom(str,approxChallengeFile)},
  {"correction",8,0,8,0,kw_264},
  {"dace_method_pointer",11,0,3,0,0,0,0,0,0,Nmom(str,subMethodPointer)},
  {"diagnostics",7,2,9,0,kw_266,0,0,0,0,Nmom(strL,diagMetrics)},
  {"export_points_file",11,2,6,0,kw_267,0,0,0,0,Nmom(str,approxExportFile)},
  {"gaussian_process",8,2,11,1,kw_272},
  {"import_points_file",11,2,5,0,kw_273,0,0,0,0,Nmom(str,approxImportFile)},
  {"kriging",0,2,11,1,kw_272,0,0,0,0,Nmom(lit,surrogateType_globalKriging)},
  {"metrics",15,2,9,0,kw_266,0,0,0,0,Nmom(str,diagMetrics)},
  {"minimum_points",8,0,2,0,0,0,0,0,0,Nmom(type,pointsManagement_MINIMUM_POINTS)},
  {"moving_least_squares",8,3,1,1,kw_276,0,0,0,0,Nmom(lit, surrogateType_globalMovingLeastSquares)}
}
12.1. DAKOTA NAMESPACE REFERENCE

```
{"neural_network", 8, 4, 1, 1, kw_277, 0, 0, 0, 0, Nmom(lit, surrogateType_global_neural_network)},
{"polynomial", 8, 4, 1, 1, kw_278, 0, 0, 0, 0, Nmom(lit, surrogateType_global_polynomial)},
{"radial_basis", 8, 5, 1, 1, kw_279, 0, 0, 0, 0, Nmom(lit, surrogateType_global_radial_basis)},
{"recommended_points", 8, 0, 2, 0, 0, 0, 0, 0, Nmom(type, pointsManagement_RECOMMENDED_POINTS)},
{"reuse_points", 8, 3, 4, 0, kw_280},
{"reuse_samples", 0, 3, 4, 0, kw_280, 0, 0, -1},
{"samples_file", 3, 2, 5, 0, kw_273, 0, 0, -12, Nmom(str, approxImportFile)},
{"total_points", 9, 0, 2, 0, 0, 0, 0, 0, Nmom(int, pointsTotal)},
{"use_derivatives", 8, 0, 7, 0, 0, 0, 0, 0, Nmom(true, modelUseDerivsFlag)}
```

**KeyWord kw_282[6] [static]**

Initial value:

```
= {
  {"additive", 8, 0, 2, 0, 0, 0, 0, 0, Nmom(type, approxCorrectionType_ADDITIVE_CORRECTION)},
  {"combined", 8, 0, 2, 0, 0, 0, 0, 0, Nmom(type, approxCorrectionType_COMBINED_CORRECTION)},
  {"first_order", 8, 0, 1, 1, 0, 0, 0, 0, Nmom(order, approxCorrectionOrder_1)},
  {"multiplicative", 8, 0, 2, 2, 0, 0, 0, 0, Nmom(type, approxCorrectionType_MULTIPLICATIVE_CORRECTION)},
  {"second_order", 8, 0, 1, 1, 0, 0, 0, 0, Nmom(order, approxCorrectionOrder_2)},
  {"zeroth_order", 8, 0, 1, 1, 0, 0, 0, 0, Nmom(order, approxCorrectionOrder_0)}
}
```

**KeyWord kw_283[3] [static]**

Initial value:

```
= {
  {"correction", 8, 6, 3, 3, kw_282},
  {"high_fidelity_model_pointer", 11, 0, 2, 2, 0, 0, 0, 0, Nmom(str, truthModelPointer)},
  {"low_fidelity_model_pointer", 11, 0, 1, 1, 0, 0, 0, 0, Nmom(str, lowFidelityModelPointer)}
}
```

**KeyWord kw_284[1] [static]**

Initial value:

```
= {
  {"actual_model_pointer", 11, 0, 1, 1, 0, 0, 0, 0, Nmom(str, truthModelPointer)}
}
```

**KeyWord kw_285[2] [static]**

Initial value:

```
= {
  {0, 0, 1, 0, 0, kw_284},
  {"taylor_series", 8, 0, 1, 1}
}
```

**KeyWord kw_286[2] [static]**

Initial value:

```
= {
  {0, 0, 1, 0, 0, kw_284},
  {"tana", 8, 0, 1, 1}
}
```
KeyWord kw_287[5]  [static]
Initial value:

= {
    {"global",8,21,2,1,kw_281},
    {"hierarchical",8,3,2,1,kw_283,0,0,0,0,Nmom(lit,surrogateType,hierarchical)},
    {"id_surrogates",13,0,1,0,0,0,0,0,Nmom(intsetml,surrogateFnIndices)},
    {"local",8,1,2,1,kw_285,0,0,0,0,Nmom(lit,surrogateType_local,taylor)},
    {"multipoint",8,1,2,1,kw_286,0,0,0,0,Nmom(lit,surrogateType_multipoint_tana)}
}

KeyWord kw_288[7]  [static]
Initial value:

= {
    {"hierarchical_tagging",8,0,4,0,0,0,0,0,Nmom(true,hierarchicalTags)},
    {"id_model",11,0,1,0,0,0,0,0,Nmom(str,idModel)},
    {"nested",8,2,5,1,kw_261,0,0,0,0,Nmom(lit,modelType_nested)},
    {"responses_pointer",11,0,3,0,0,0,0,0,Nmom(str,responsesPointer)},
    {"single",8,1,5,1,kw_262,0,0,0,0,Nmom(lit,modelType_single)},
    {"surrogate",8,1,5,1,kw_287,0,0,0,0,Nmom(lit,modelType_surrogate)},
    {"variables_pointer",11,0,2,0,0,0,0,0,Nmom(str,variablesPointer)}
}

KeyWord kw_289[6]  [static]
Initial value:

= {
    {"annotated",8,0,3,0,0,0,0,0,Nrem(true,expDataFileAnnotated)},
    {"freeform",8,0,3,0,0,0,0,0,Nrem(false,expDataFileAnnotated)},
    {"num_config_variables",0x29,0,4,0,0,0,0,0,Nrem(sizet,numExpConfigVars)},
    {"num_experiments",0x29,0,1,0,0,0,0,0,Nrem(sizet,numExperiments)},
    {"num_replicates",13,0,2,0,0,0,0,0,Nrem(ivec,numReplicates)},
    {"num_std_deviations",0x29,0,5,0,0,0,0,0,Nrem(sizet,numExpStdDeviations)}
}

KeyWord kw_290[6]  [static]
Initial value:

= {
    {"nonlinear_equality_scale_types",0x807,0,2,0,0,0,0,3,Nrem(strL,nonlinearEqScaleTypes)},
    {"nonlinear_equality_scales",0x806,0,3,0,0,0,0,3,Nrem(RealDL,nonlinearEqScales)},
    {"nonlinear_equality_targets",6,0,1,0,0,0,0,3,Nrem(RealDL,nonlinearEqTargets)},
    {"scale_types",0x805,0,2,0,0,0,0,0,Nrem(strL,nonlinearEqScaleTypes)},
    {"scales",0x804,0,3,0,0,0,0,0,Nrem(RealDL,nonlinearEqScales)},
    {"targets",14,0,1,0,0,0,0,3,Nrem(RealDL,nonlinearEqTargets)}
}

KeyWord kw_291[8]  [static]
Initial value:
12.1. DAKOTA NAMESPACE REFERENCE

= {
  ["lower_bounds",14,0,1,0,0,0,0,0,0,N_rem(RealDL,nonlinearIneqLowerBnds)],
  "nonlinear_inequality_lower_bounds",6,0,1,0,0,0,0,0,1,N_rem(RealDL,nonlinearIneqLowerBnds),
  "nonlinear_inequality_scale_types",0x807,0,3,0,0,0,0,0,3,N_rem(strL,nonlinearIneqScaleTypes),
  ["nonlinear_inequality_scales",0x806,0,4,0,0,0,0,0,3,N_rem(RealDL,nonlinearIneqScales)],
  "nonlinear_inequality_upper_bounds",6,0,2,0,0,0,0,0,3,N_rem(RealDL,nonlinearIneqUpperBnds),
  "scale_types",0x80f,0,1,0,0,0,0,0,0,N_rem(strL,nonlinearIneqScaleTypes),
  ["scales",0x80e,0,4,0,0,0,0,0,3,N_rem(RealDL,nonlinearIneqScales)],
  "upper_bounds",14,0,2,0,0,0,0,0,0,N_rem(RealDL,nonlinearIneqUpperBnds)}

KeyWord kw_292[15] [static]
Initial value:

= {
  ["calibration_data_file",11,6,4,0,kw_289,0,0,0,0,0,N_rem(str,expDataFileName)],
  ["calibration_term_scale_types",0x807,0,1,0,0,0,0,11,N_rem(strL,primaryRespFnScaleTypes)],
  ["calibration_term_scales",0x806,0,2,0,0,0,0,11,N_rem(RealDL,primaryRespFnScales)],
  ["calibration_weights",6,0,3,0,0,0,0,11,N_rem(RealDL,primaryRespFnWeights)],
  ["least_squares_data_file",3,6,4,0,kw_289,0,0,0,0,0,N_rem(str,expDataFileName)],
  ["least_squares_term_scale_types",0x807,0,1,0,0,0,0,7,N_rem(strL,primaryRespFnScaleTypes)],
  ["least_squares_term_scales",0x806,0,2,0,0,0,0,7,N_rem(RealDL,primaryRespFnScales)],
  ["least_squares_weights",6,0,3,0,0,0,0,7,N_rem(RealDL,primaryRespFnWeights)],
  ["nonlinear_inequality_constraints",0x29,6,6,0,kw_290,0,0,0,0,0,N_rem(sizet, numNonlinearEqConstraints)],
  ["num_nonlinear_inequality_constraints",0x29,8,6,0,kw_291,0,0,0,0,0,N_rem(sizet, numNonlinearEqConstraints)],
  ["primary_scale_types",0x80f,0,1,0,0,0,0,0,0,N_rem(strL,primaryRespFnScaleTypes)],
  ["primary_scales",0x80e,0,2,0,0,0,0,0,0,N_rem(RealDL,primaryRespFnScales)],
  ["weights",14,0,2,0,0,0,0,0,0,N_rem(RealDL,primaryRespFnWeights)]
}

KeyWord kw_293[4] [static]
Initial value:

= {
  ["absolute",8,0,2,0,0,0,0,0,0,N_rem(lit,fdGradStepType_absolute)],
  ["bounds",8,0,2,0,0,0,0,0,0,N_rem(lit,fdGradStepType_bounds)],
  ["central",8,0,4,0,0,0,0,0,0,N_rem(lit,intervalType_central)],
  ["fd_gradient_step_size",6,0,5,0,0,0,0,0,1,N_rem(RealL,fdGradStepSize)],
  ["fd_step_size",14,0,5,0,0,0,0,0,0,N_rem(RealL,fdGradStepSize)],
  ["forward",8,0,4,0,0,0,0,0,0,N_rem(lit,intervalType_forward)],
  ["method_source",8,0,3],
  ["vendor",8,0,2,0,0,0,0,0,0,N_rem(lit,methodSource_vendor)]
}

KeyWord kw_294[8] [static]
Initial value:

= {
  ["central",8,0,4,0,0,0,0,0,0,N_rem(lit,intervalType_central)],
  ["dakota",8,4,2,0,kw_293,5,0,0,0,N_rem(lit,methodSource_dakota)],
  ["fd_gradient_step_size",6,0,5,0,0,0,0,0,1,N_rem(RealL,fdGradStepSize)],
  ["fd_step_size",14,0,5,0,0,0,0,0,0,N_rem(RealL,fdGradStepSize)],
  ["forward",8,0,4,0,0,0,0,0,0,N_rem(lit,intervalType_forward)],
  ["interval_type",8,0,3],
  ["method_source",8,0,1],
  ["vendor",8,0,2,0,0,0,0,0,0,N_rem(lit,methodSource_vendor)]
}
KeyWord kw_295[3]  [static]
Initial value:
= {
    {0,0,8,0,0,kw_294},
    {"id_analytic_gradients",13,0,2,2,0,0,0,0,N_rem(intset,idAnalyticGrads)},
    {"id_numerical_gradients",13,0,1,1,0,0,0,0,N_rem(intset,idNumericalGrads)}
}

KeyWord kw_296[2]  [static]
Initial value:
= {
    {"fd_hessian_step_size",6,0,1,0,0,0,0,1,N_rem(RealL,fdHessStepSize)},
    {"fd_step_size",14,0,1,0,0,0,0,0,N_rem(RealL,fdHessStepSize)}
}

KeyWord kw_297[1]  [static]
Initial value:
= {
    {"damped",8,0,1,0,0,0,0,0,N_rem(lit,quasiHessianType_damped_bfgs)}
}

KeyWord kw_298[2]  [static]
Initial value:
= {
    {"bfgs",8,0,1,1,0,0,0,0,N_rem(lit,quasiHessianType_bfgs)},
    {"sr1",8,0,1,1,0,0,0,0,N_rem(lit,quasiHessianType_sr1)}
}

KeyWord kw_299[8]  [static]
Initial value:
= {
    {"absolute",8,0,2,0,0,0,0,0,N_rem(lit,fdHessStepType_absolute)},
    {"bounds",8,0,2,0,0,0,0,0,N_rem(lit,fdHessStepType_bounds)},
    {"central",8,0,3,0,0,0,0,0,N_rem(true,centralHess)},
    {"forward",8,0,3,0,0,0,0,0,N_rem(false,centralHess)},
    {"id_analytic_hessians",13,0,5,0,0,0,0,0,N_rem(intset,idAnalyticHessians)},
    {"id_numerical_hessians",13,2,1,0,kw_297,0,0,0,N_rem(intset,idNumericalHessians)},
    {"id_quasi_hessians",13,2,4,0,kw_298,0,0,0,N_rem(intset,idQuasiHessians)},
    {"relative",8,0,2,0,0,0,0,0,N_rem(lit,fdHessStepType_relative)}
}

KeyWord kw_300[6]  [static]
Initial value:
= {
    {"nonlinear_equality_scale_types","0x807,0,2,0,0,0,0,0,3,N_rem(strl,nonlinearEqScaleTypes)}},
    {"nonlinear_equality_scales","0x806,0,3,0,0,0,0,0,3,N_rem(RealDL,nonlinearEqScales)}},
    {"nonlinear_equality_targets",6,0,1,0,0,0,0,3,N_rem(RealDL,nonlinearEqTargets)},
    {"scale_types","0x805,0,2,0,0,0,0,0,0,N_rem(strl,nonlinearEqScaleTypes)}},
    {"scales","0x804,0,3,0,0,0,0,0,0,N_rem(RealDL,nonlinearEqScales)}},
    {"targets",14,0,1,0,0,0,0,0,N_rem(RealDL,nonlinearEqTargets)}
}
12.1. DAKOTA NAMESPACE REFERENCE

KeyWord kw_301[8] [static]
Initial value:
= {
    "lower_bounds", 14, 0, 1, 0, 0, 0, 0, 0, rem(RealDL, nonlinearIneqLowerBnds),
    "nonlinear_inequality_lower_bounds", 6, 0, 1, 0, 0, 0, 0, -1, rem(RealDL, nonlinearIneqLowerBnds),
    "nonlinear_inequality_lower_bounds", 0, x807, 0, 0, 0, 0, 0, 0, rem(RealDL, nonlinearIneqScaleTypes),
    "nonlinear_inequality_scales", 0, x806, 0, 4, 0, 0, 0, 0, rem(RealDL, nonlinearIneqScales),
    "nonlinear_inequality_upper_bounds", 6, 0, 2, 0, 0, 0, 0, 3, N, rem(RealDL, nonlinearIneqUpperBnds)
}

KeyWord kw_302[11] [static]
Initial value:
= {
    "multi_objective_weights", 6, 0, 1, 0, 0, 0, 0, 10, N, rem(RealDL, primaryRespFnWeights),
    "nonlinear_inequality_constraints", 0, x29, 6, 6, 0, kw_300, 0, 0, 0, N, rem(sizet, numNonlinearEqConstraints),
    "nonlinear_inequality_constraints", 0, x29, 8, 5, 0, kw_301, 0, 0, 0, N, rem(sizet, numNonlinearIneqConstraints),
    "objective_function_scale_types", 0, x807, 0, 2, 0, 0, 0, 0, 2, N, rem(strl, primaryRespFnScaleTypes),
    "objective_function_scales", 0, x806, 0, 3, 0, 0, 0, 0, 2, N, rem(RealDL, primaryRespFnScales),
    "primary_scale_types", 0, x80f, 0, 3, 0, 0, 0, 0, N, rem(strl, nonlinearIneqScaleTypes),
    "primary_scales", 0, x80e, 0, 4, 0, 0, 0, 0, N, rem(RealDL, nonlinearIneqScales),
    "upper_bounds", 14, 0, 2, 0, 0, 0, 0, 0, N, rem(RealDL, nonlinearIneqUpperBnds)
}

KeyWord kw_303[7] [static]
Initial value:
= {
    "absolute", 8, 0, 2, 0, 0, 0, 0, 0, N, rem(lit, fdHessStepType_absolute),
    "bounds", 8, 0, 2, 0, 0, 0, 0, N, rem(lit, fdHessStepType_bounds),
    "central", 8, 0, 3, 0, 0, 0, 0, N, rem(true, centralHess),
    "fd_hessian_step_size", 6, 0, 1, 0, 0, 0, 0, 1, N, rem(RealL, fdHessStepSize),
    "fd_step_size", 14, 0, 1, 0, 0, 0, 0, N, rem(RealL, fdHessStepSize),
    "forward", 8, 0, 3, 0, 0, 0, 0, N, rem(false, centralHess),
    "relative", 8, 0, 2, 0, 0, 0, 0, N, rem(lit, fdHessStepType_relative)
}

KeyWord kw_304[1] [static]
Initial value:
= {
    "damped", 8, 0, 1, 0, 0, 0, 0, N, rem(lit, quasiHessianType_damped_bfgs)
}
KeyWord kw_305[2]  [static]
Initial value:
= {
   {"bfgs",8,1,1,1,kw_304,0,0,0,N_rem(lit,quasiHessianType_bfgs)},
   {"sr1",8,0,1,1,0,0,0,0,N_rem(lit,quasiHessianType_sr1)}
}

KeyWord kw_306[19]  [static]
Initial value:
= {
   {"analytic_gradients",8,0,4,2,0,0,0,0,N_rem(lit,gradientType_analytic)},
   {"analytic_hessians",8,0,5,3,0,0,0,0,N_rem(lit,hessianType_analytic)},
   {"calibration_terms",0x29,15,3,1,kw_292,0,0,0,0,N_rem(sizet,numLeastSqTerms)},
   {"descriptors",15,0,2,0,0,0,0,0,0,N_rem(strL,responseLabels)},
   {"least_squares_terms",0x21,15,3,1,kw_292,0,0,0,0,N_rem(sizet,numLeastSqTerms)},
   {"mixed_gradients",8,0,4,2,0,0,0,0,N_rem(lit,gradientType_mixed)},
   {"mixed_hessians",8,0,5,3,0,0,0,0,N_rem(lit,hessianType_mixed)},
   {"quasi_hessians",8,0,5,3,0,0,0,0,N_rem(lit,hessianType_quasi)},
   {"analytic_gradients",8,0,4,2,0,0,0,0,N_rem(lit,gradientType_analytic)},
   {"analytic_hessians",8,0,5,3,0,0,0,0,N_rem(lit,hessianType_analytic)},
   {"calibration_terms",0x29,15,3,1,kw_292,0,0,0,0,N_rem(sizet,numLeastSqTerms)},
   {"descriptors",15,0,2,0,0,0,0,0,0,N_rem(strL,responseLabels)},
   {"least_squares_terms",0x21,15,3,1,kw_292,0,0,0,0,N_rem(sizet,numLeastSqTerms)},
   {"mixed_gradients",8,0,4,2,0,0,0,0,N_rem(lit,gradientType_mixed)},
   {"mixed_hessians",8,0,5,3,0,0,0,0,N_rem(lit,hessianType_mixed)},
   {"quasi_hessians",8,0,5,3,0,0,0,0,N_rem(lit,hessianType_quasi)},
   {"objective_functions",0x29,11,3,1,kw_302,0,0,0,0,N_rem(sizet,numObjectiveFunctions)},
   {"quasi_hessians",8,0,5,3,0,0,0,0,N_rem(lit,hessianType_quasi)},
   {"response_descriptors",7,0,2,0,0,0,0,0,0,N_rem(strL,responseLabels)},
   {"response_functions",0x29,0,3,1,0,0,0,0,0,N_rem(sizet,numResponseFunctions)}
}

KeyWord kw_307[6]  [static]
Initial value:
= {
   {"aleatory",8,0,1,1,0,0,0,0,N_vam(type,varsView_ALEATORY,UNCERTAIN_VIEW)},
   {"design",8,0,1,1,0,0,0,0,N_vam(type,varsView_ALL_VIEW)},
   {"epistemic",8,0,1,1,0,0,0,0,N_vam(type,varsView_EPISTEMIC,UNCERTAIN_VIEW)},
   {"state",8,0,1,1,0,0,0,0,N_vam(type,varsView_STATE_VIEW)},
   {"uncertain",8,0,1,1,0,0,0,0,N_vam(type,varsView_UNCERTAIN_VIEW)}
}

KeyWord kw_308[11]  [static]
Initial value:
= {
   {"alphas",14,0,1,1,0,0,0,0,N_vam(RealLb,RealUncAlphas)},
   {"betas",14,0,2,2,0,0,0,0,N_vam(RealLb,RealUncBetas)},
   {"buv_alphas",6,0,1,1,0,0,0,0,-2,N_vam(RealLb,RealUncAlphas)},
   {"buv_betas",6,0,2,2,0,0,0,0,-2,N_vam(RealLb,RealUncBetas)},
   {"buv_descriptors",7,0,6,0,0,0,0,0,3,N_vam(caulbl,CAUVar_beta)},
   {"buv_lower_bounds",6,0,3,3,0,0,0,0,4,N_vam(rvec,RealUncLowerBnds)},
   {"buv_upper_bounds",6,0,3,3,0,0,0,0,4,N_vam(rvec,RealUncUpperBnds)},
   {"initial_point",14,0,5,5,0,0,0,0,N_vam(rvec,RealUncVars)},
   {"lower_bounds",14,0,3,3,0,0,0,0,0,N_vam(rvec,RealUncLowerBnds)},
   {"upper_bounds",14,0,4,4,0,0,0,0,0,N_vam(rvec,RealUncUpperBnds)}
}
12.1. DAKOTA NAMESPACE REFERENCE

**KeyWord kw_309[5]** [static]

Initial value:

```
= {
    {'descriptors*:15,0,4,0,0,0,0,0,N_vae(duailbl,DAUIVar_binaryal)},
    {'initial_point*:13,0,3,0,0,0,0,0,N_vam(IniLb,binomialUncVars)},
    {'num_trials*:13,0,2,0,0,0,0,0,N_vam(IniLb,binomialUncNumTrials)},
    {'prob_per_trial*:6,0,1,1,0,0,0,0,0,N_vam(binomialUncProbPerTrial),
     'probability_per_trial*:14,0,1,1,0,0,0,0,0,N_vam(binomialUncProbPerTrial)}
}
```

**KeyWord kw_310[12]** [static]

Initial value:

```
= {
    {'cdv_descriptors*:7,0,6,0,0,0,0,0,N_vam(strL,continuousDesignLabels)},
    {'cdv_initial_point*:6,0,1,0,0,0,0,0,6,N_vam(rvec,continuousDesignVars)},
    {'cdv_lower_bounds*:6,0,2,0,0,0,0,0,6,N_vam(rvec,continuousDesignLowerBnds)},
    {'cdv_scale_types*:0x807,0,0,0,0,6,N_vam(strL,continuousDesignScaleTypes)},
    {'cdv_scales*:0x806,0,0,0,0,6,N_vam(rvec,continuousDesignScales)},
    {'cdv_upper_bounds*:6,0,3,0,0,0,0,0,6,N_vam(rvec,continuousDesignUpperBnds)},
    {'descriptors*:15,0,6,0,0,0,0,0,N_vam(strL,continuousDesignLabels)},
    {'initial_point*:14,0,1,0,0,0,0,0,0,N_vam(rvec,continuousDesignVars)},
    {'lower_bounds*:14,0,2,0,0,0,0,0,0,N_vam(rvec,continuousDesignLowerBnds)},
    {'scale_types*:0x80f,0,0,0,0,0,0,N_vam(strL,continuousDesignScaleTypes)},
    {'scales*:0x80e,0,0,0,0,0,0,N_vam(rvec,continuousDesignScales)},
    {'upper_bounds*:14,0,3,0,0,0,0,0,0,N_vam(rvec,continuousDesignUpperBnds)}
}
```

**KeyWord kw_311[10]** [static]

Initial value:

```
= {
    {'descriptors*:15,0,6,0,0,0,0,0,0,N_vae(ceulbl,CEUVar_interval)},
    {'initial_point*:14,0,5,0,0,0,0,0,0,N_vam(rvec,continuousIntervalUncVars)},
    {'interval_probabilities*:14,0,2,0,0,0,0,0,0,N_vam(newrvec,Var_Info_CIp)},
    {'interval_probs*:6,0,2,0,0,0,0,0,0,N_vam(newrvec,Var_Info_CIp)},
    {'iuv_descriptors*:7,0,6,0,0,0,0,0,0,0,N_vae(ceulbl,CEUVar_interval)},
    {'iuv_interval_probs*:6,0,2,0,0,0,0,0,0,N_vam(newrvec,Var_Info_CIp)},
    {'iuv_num_intervals*:5,0,1,0,0,0,0,0,2,N_vam(newiarray,Var_Info_CI)},
    {'iuv_intervals*:13,0,1,0,0,0,0,0,0,N_vam(newiarray,Var_Info_CI)},
    {'upper_bounds*:14,0,4,2,0,0,0,0,0,N_vam(newrvec,Var_Info_CIlb)}
}
```

**KeyWord kw_312[8]** [static]

Initial value:

```
= {
    {'csv_descriptors*:7,0,4,0,0,0,0,0,4,N_vam(strL,continuousStateLabels)},
    {'csv_initial_state*:6,0,1,0,0,0,0,0,4,N_vam(rvec,continuousStateVars)},
    {'csv_lower_bounds*:6,0,2,0,0,0,0,0,4,N_vam(rvec,continuousStateLowerBnds)},
    {'csv_upper_bounds*:6,0,3,0,0,0,0,0,4,N_vam(rvec,continuousStateUpperBnds)},
    {'descriptors*:15,0,4,0,0,0,0,0,0,N_vae(strL,continuousStateLabels)},
    {'initial_state*:14,0,1,0,0,0,0,0,0,N_vam(rvec,continuousStateVars)},
    {'lower_bounds*:14,0,2,0,0,0,0,0,0,N_vam(rvec,continuousStateLowerBnds)},
    {'upper_bounds*:14,0,3,0,0,0,0,0,0,N_vam(rvec,continuousStateUpperBnds)}
}
```
KeyWord kw_313[8]  [static]
Initial value:

= 

{ "ddv_descriptors",7,0,4,0,0,0,0,0,4,N_vam(strL,discreteDesignRangeLabels) },
{ "ddv_initial_point",5,0,1,0,0,0,0,4,N_vam(ivec,discreteDesignRangeVars) },
{ "ddv_lower_bounds",5,0,2,0,0,0,0,4,N_vam(ivec,discreteDesignRangeLowerBnds) },
{ "ddv_upper_bounds",5,0,3,0,0,0,0,4,N_vam(ivec,discreteDesignRangeUpperBnds) },
{ "descriptors",15,0,4,0,0,0,0,0,4,N_vam(strL,discreteDesignRangeLabels) },
{ "initial_point",13,0,1,0,0,0,0,0,0,N_vam(ivec,discreteDesignRangeVars) },
{ "lower_bounds",13,0,2,0,0,0,0,0,0,N_vam(ivec,discreteDesignRangeLowerBnds) },
{ "upper_bounds",13,0,3,0,0,0,0,0,0,N_vam(ivec,discreteDesignRangeUpperBnds) }

KeyWord kw_314[4]  [static]
Initial value:

= 

{ "descriptors",15,0,4,0,0,0,0,0,0,N_vam(strL,discreteDesignSetIntLabels) },
{ "initial_point",13,0,1,0,0,0,0,0,0,N_vam(ivec,discreteDesignSetIntVars) },
{ "num_set_values",13,0,2,0,0,0,0,0,0,N_vam(newiarray,Var_Info_nddsi) },
{ "set_values",13,0,3,1,0,0,0,0,0,N_vam(newivec,Var_Info_ddsi) }

KeyWord kw_315[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(rvec,discreteDesignSetRealVars) },
{ "num_set_values",13,0,2,0,0,0,0,0,0,N_vam(newiarray,Var_Info_nddsr) },
{ "set_values",14,0,3,1,0,0,0,0,0,N_vam(newrvec,Var_Info_ddsr) }

KeyWord kw_316[9]  [static]
Initial value:

= 

{ "descriptors",15,0,6,0,0,0,0,0,0,N_vae(deuilbl,DEUIVar_interval) },
{ "interval_probabilities",14,0,2,0,0,0,0,0,0,N_vam(newrvec,Var_Info_DIp) },
{ "lower_bounds",13,0,2,0,0,0,0,0,0,N_vam(newrvec,Var_Info_DIlb) },
{ "num_intervals",13,0,1,0,0,0,0,0,0,N_vam(newiarray,Var_Info_DII) },
{ "range_probs",6,0,2,0,0,0,0,0,0,N_vam(newrvec,Var_Info_DIp) },
{ "range_values",6,0,0,0,0,0,0,0,0,N_vam(newrvec,Var_Info_DVal) },
{ "upper_bounds",13,0,4,2,0,0,0,0,0,N_vam(newrvec,Var_Info_DIub) }

KeyWord kw_317[8]  [static]
Initial value:
12.1. DAKOTA NAMESPACE REFERENCE

= {
  "descriptors":15,0,4,0,0,0,0,N_vam(strL,discreteStateRangeLabels),
  "dsv_descriptors":7,0,4,0,0,0,0,-1,N_vam(strL,discreteStateRangeLabels),
  "dsv_initial_state":5,0,1,0,0,0,0,3,N_vam(ivec,discreteStateRangeVars),
  "dsv_lower_bounds":5,0,2,0,0,0,0,3,N_vam(ivec,discreteStateRangeLowerBnds),
  "dsv_upper_bounds":5,0,3,0,0,0,0,3,N_vam(ivec,discreteStateRangeUpperBnds),
  "initial_state":13,0,1,0,0,0,0,0,N_vam(ivec,discreteStateRangeVars),
  "lower_bounds":13,0,2,0,0,0,0,0,N_vam(ivec,discreteStateRangeLowerBnds),
  "upper_bounds":13,0,3,0,0,0,0,0,N_vam(ivec,discreteStateRangeUpperBnds)
}

KeyWord kw_318[4] [static]
Initial value:

= {
  "descriptors":15,0,4,0,0,0,0,0,N_vam(strL,discreteStateSetIntLabels),
  "initial_state":13,0,1,0,0,0,0,0,N_vam(ivec,discreteStateSetIntVars),
  "num_set_values":13,0,2,0,0,0,0,0,N_vam(newiarray,Var_Info.ndssi),
  "set_values":13,0,3,1,0,0,0,0,N_vam(newivec,Var_Info_dssi)
}

KeyWord kw_319[4] [static]
Initial value:

= {
  "descriptors":15,0,4,0,0,0,0,0,N_vam(strL,discreteStateSetRealLabels),
  "initial_state":14,0,1,0,0,0,0,0,N_vam(rvec,discreteStateSetRealVars),
  "num_set_values":13,0,2,0,0,0,0,0,N_vam(newiarray,Var_Info.ndssr),
  "set_values":14,0,3,1,0,0,0,0,N_vam(newrvec,Var_Info_dssr)
}

KeyWord kw_320[6] [static]
Initial value:

= {
  "descriptors":15,0,5,0,0,0,0,0,N_vae(deuilbl,DEUIVar_set_int),
  "initial_point":13,0,4,0,0,0,0,0,0,N_vam(ivec,discreteUncSetIntVars),
  "num_set_values":13,0,1,0,0,0,0,0,N_vam(newiarray,Var_Info.ndusi),
  "set_probs":6,0,3,0,0,0,0,-1,N_vam(newrvec,Var_Info_DSIP),
  "set_values":13,0,2,1,0,0,0,0,N_vam(newivec,Var_Info_dusi)
}

KeyWord kw_321[6] [static]
Initial value:

= {
  "descriptors":15,0,5,0,0,0,0,0,N_vae(deurlbl,DEURVar_set_real),
  "initial_point":14,0,4,0,0,0,0,0,0,N_vam(rvec,discreteUncSetRealVars),
  "num_set_values":13,0,1,0,0,0,0,0,N_vam(newiarray,Var_Info ndusr),
  "set_probs":6,0,3,0,0,0,0,-1,N_vam(newrvec,Var_Info_DSRP),
  "set_values":14,0,2,1,0,0,0,0,N_vam(newrvec,Var_Info_dusr)
}
KeyWord kw_322[5] [static]

Initial value:
= {
  {"betas",14,0,1,1,0,0,0,0,0,N_vam(RealLb,exponentialUncBetas)},
  {"descriptors",15,0,1,0,0,0,0,0,N_vae(caulbl,CAUVar_exponential)},
  {"euv_betas",6,0,1,1,0,0,0,0,-2,N_vam(RealLb,exponentialUncBetas)},
  {"euv_descriptors",7,0,3,0,0,0,0,0,-2,N_vae(caulbl,CAUVar_exponential)},
  {"initial_point",14,0,2,0,0,0,0,0,0,N_vam(RealLb,exponentialUncVars)}
}

KeyWord kw_323[7] [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(rvec,frechetUncBetas)},
  {"descriptors",15,0,4,0,0,0,0,0,N_vae(caulbl,CAUVar_frechet)},
  {"fuv_alphas",6,0,1,1,0,0,0,0,-3,N_vam(RealLb,frechetUncAlphas)},
  {"fuv_betas",6,0,2,2,0,0,0,0,-3,N_vam(rvec,frechetUncBetas)},
  {"fuv_descriptors",7,0,4,0,0,0,0,0,-3,N_vae(caulbl,CAUVar_frechet)},
  {"initial_point",14,0,3,0,0,0,0,0,0,N_vam(rvec,frechetUncVars)}
}

KeyWord kw_324[7] [static]

Initial value:
= {
  {"alphas",14,0,1,1,0,0,0,0,0,N_vam(RealLb,gammaUncAlphas)},
  {"betas",14,0,2,2,0,0,0,0,0,N_vam(RealLb,gammaUncBetas)},
  {"descriptors",15,0,4,0,0,0,0,0,N_vae(caulbl,CAUVar_gamma)},
  {"gauv_alphas",6,0,1,1,0,0,0,0,-3,N_vam(RealLb,gammaUncAlphas)},
  {"gauv_betas",6,0,2,2,0,0,0,0,-3,N_vam(RealLb,gammaUncBetas)},
  {"gauv_descriptors",7,0,4,0,0,0,0,0,-3,N_vae(caulbl,CAUVar_gamma)},
  {"initial_point",14,0,3,0,0,0,0,0,0,N_vam(rvec,gammaUncVars)}
}

KeyWord kw_325[4] [static]

Initial value:
= {
  {"descriptors",15,0,3,0,0,0,0,0,0,N_vae(dauilbl,DAUIVar_geometric)},
  {"initial_point",13,0,2,0,0,0,0,0,0,N_vam(intlb,geometricUncVars)},
  {"prob_per_trial",6,0,1,1,0,0,0,1,N_vam(rvec,geometricUncProbPerTrial)},
  {"probability_per_trial",14,0,1,1,0,0,0,0,0,N_vam(rvec,geometricUncProbPerTrial)}
}

KeyWord kw_326[7] [static]

Initial value:
= {
  {"alphas",14,0,1,1,0,0,0,0,0,N_vam(RealLb,gumbelUncAlphas)},
  {"betas",14,0,2,2,0,0,0,0,0,N_vam(rvec,gumbelUncBetas)},
  {"descriptors",15,0,4,0,0,0,0,0,N_vae(caulbl,CAUVar_gumbel)},
  {"gauv_alphas",6,0,1,1,0,0,0,0,-3,N_vam(RealLb,gumbelUncAlphas)},
  {"gauv_betas",6,0,2,2,0,0,0,0,-3,N_vam(rvec,gumbelUncBetas)},
  {"gauv_descriptors",7,0,4,0,0,0,0,0,-3,N_vae(caulbl,CAUVar_gumbel)},
  {"initial_point",14,0,3,0,0,0,0,0,0,N_vam(rvec,gumbelUncVars)}
}
KeyWord kw_327[11]  [static]
Initial value:
= {
  "abscissas", 14, 0, 2, 1, 0, 0, 0, 0, 0, 0, N_vam(newrvec, Var_Info_hba),
  "counts", 14, 0, 3, 2, 0, 0, 0, 0, 0, 0, N_vam(newrvec, Var_Info_hbc),
  "descriptors", 15, 0, 5, 0, 0, 0, 0, 0, 0, N_vae(caulbl, CAUVar_histogram_bin),
  "huv_bin_abscissas", 6, 0, 2, 1, 0, 0, 0, 0, -3, N_vam(newrvec, Var_Info_hba),
  "huv_bin_counts", 6, 0, 3, 2, 0, 0, 0, 0, -3, N_vam(newrvec, Var_Info_hbc),
  "huv_bin_descriptors", 7, 0, 5, 0, 0, 0, 0, -3, N_vae(caulbl, CAUVar_histogram_bin),
  "huv_binordinates", 6, 0, 3, 2, 0, 0, 0, 0, 4, N_vam(newrvec, Var_Info_hbo),
  "huv_numbin_pairs", 5, 0, 1, 0, 0, 0, 0, 2, N_vam(newarray, Var_Info_nhbp),
  "initial_point", 14, 0, 4, 0, 0, 0, 0, 0, 0, N_vam(rvec, histogramBinUncVars),
  "ordinates", 14, 0, 3, 2, 0, 0, 0, 0, 0, N_vam(newrvec, Var_Info_hbo),
}

KeyWord kw_328[9]  [static]
Initial value:
= {
  "abscissas", 14, 0, 2, 1, 0, 0, 0, 0, 0, 0, N_vam(newrvec, Var_Info_hpa),
  "counts", 14, 0, 3, 2, 0, 0, 0, 0, 0, 0, N_vam(newrvec, Var_Info_hpc),
  "descriptors", 15, 0, 5, 0, 0, 0, 0, 0, 0, 0, N_vae(daurlbl, DAURVar_histogram_point),
  "huv_num_point_pairs", 5, 0, 1, 0, 0, 0, 0, 5, N_vam(newarray, Var_Info_nhpp),
  "huv_point_abscissas", 6, 0, 2, 1, 0, 0, 0, 0, -4, N_vam(newrvec, Var_Info_hpa),
  "huv_point_counts", 6, 0, 3, 2, 0, 0, 0, 0, -4, N_vam(newrvec, Var_Info_hpc),
  "huv_point_descriptors", 7, 0, 5, 0, 0, 0, 0, -4, N_vae(daurlbl, DAURVar_histogram_point),
  "initial_point", 14, 0, 4, 0, 0, 0, 0, 0, 0, N_vam(rvec, histogramPointUncVars),
  "num_pairs", 13, 0, 1, 0, 0, 0, 0, 0, 0, N_vam(newarray, Var_Info_nhpp),
}

KeyWord kw_329[5]  [static]
Initial value:
= {
  "descriptors", 15, 0, 5, 0, 0, 0, 0, 0, 0, 0, N_vae(dauilbl, DAUIVar_hypergeometric),
  "initial_point", 13, 0, 4, 0, 0, 0, 0, 0, 0, N_vam(InIntLb, hyperGeomUncVars),
  "selected_population", 13, 0, 2, 2, 0, 0, 0, 0, 0, N_vam(InIntLb, hyperGeomUncSelectedPop),
  "total_population", 13, 0, 1, 1, 0, 0, 0, 0, 0, N_vam(InIntLb, hyperGeomUncTotalPop),
}

KeyWord kw_330[2]  [static]
Initial value:
= {
  "lnuv_zetas", 6, 0, 1, 1, 0, 0, 0, 0, 1, N_vam(RealLb, lognormalUncZetas),
  "zetas", 14, 0, 1, 1, 0, 0, 0, 0, 0, N_vam(RealLb, lognormalUncZetas),
}

KeyWord kw_331[4]  [static]
Initial value:
= {
  "error_factors", 14, 0, 1, 1, 0, 0, 0, 0, 0, N_vam(RealLb, lognormalUncErrFacts),
  "lnuv_std_deviations", 6, 0, 1, 1, 0, 0, 0, 0, 1, N_vam(RealLb, lognormalUncStdDevs),
  "std_deviations", 14, 0, 1, 1, 0, 0, 0, 0, 0, N_vam(RealLb, lognormalUncStdDevs),
}
KeyWord kw_332[11]  [static]
Initial value:

= {
  {"descriptors",15,0,5,0,0,0,0,0,N_vae(caulbl,CAUVar,lognormal)},
  {"initial_point",14,0,4,0,0,0,0,0,N_vam(RealLb,lognormalUncVars)},
  {"lambdas",14,2,11,kw_330,0,0,0,0,0,N_vae(caulbl,CAUVar,lognormal)},
  {"lnuv_descriptors",7,0,5,0,0,0,0,0,3,N_vae(caulbl,CAUVar,lognormal)},
  {"lnuv_lambdas",6,2,11,kw_330,0,0,0,0,0,N_vam(rvec,lognormalUncLambdas)},
  {"lnuv_lower_bounds",6,0,2,0,0,0,0,0,3,N_vam(RealLb,lognormalUncLowerBnds)},
  {"lnuv_means",6,4,1,1,kw_331,0,0,0,0,0,N_vam(RealLb,lognormalUncMeans)},
  {"lnuv_upper_bounds",6,0,3,0,0,0,0,0,3,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_331,0,0,0,0,0,N_vam(RealLb,lognormalUncMeans)},
  {"upper_bounds",14,0,3,0,0,0,0,0,0,N_vam(RealUb,lognormalUncUpperBnds)}
}

KeyWord kw_333[7]  [static]
Initial value:

= {
  {"descriptors",15,0,4,0,0,0,0,0,N_vae(caulbl,CAUVar,loguniform)},
  {"initial_point",14,0,3,0,0,0,0,0,N_vam(RealLb,loguniformUncVars)},
  {"lower_bounds",14,0,1,1,0,0,0,0,0,N_vam(RealLb,loguniformUncLowerBnds)},
  {"luuv_descriptors",7,0,4,0,0,0,0,0,3,N_vae(caulbl,CAUVar,loguniform)},
  {"luuv_lower_bounds",6,0,1,1,0,0,0,0,0,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)}
}

KeyWord kw_334[5]  [static]
Initial value:

= {
  {"descriptors",15,0,4,0,0,0,0,0,N_vae(daulbl,DAUIVar,negative_binomial)},
  {"initial_point",13,0,3,0,0,0,0,0,N_vam(IntLb,negBinomialUncVars)},
  {"num_trials",13,0,2,2,0,0,0,0,0,N_vam(IntLb,negBinomialUncNumTrials)},
  {"prob_per_trial",6,0,1,1,0,0,0,0,1,N_vam(rvec,negBinomialUncProbPerTrial)},
  {"probability_per_trial",14,0,1,1,0,0,0,0,0,N_vam(rvec,negBinomialUncProbPerTrial)}
}

KeyWord kw_335[11]  [static]
Initial value:

= {
  {"descriptors",15,0,6,0,0,0,0,0,0,N_vae(caulbl,CAUVar,normal)},
  {"initial_point",14,0,5,0,0,0,0,0,0,N_vam(rvec,normalUncVars)},
  {"lower_bounds",14,0,3,0,0,0,0,0,0,N_vam(rvec,normalUncLowerBnds)},
  {"means",14,0,1,1,0,0,0,0,0,N_vam(rvec,normalUncMeans)},
  {"nuv_descriptors",7,0,6,0,0,0,0,0,4,N_vae(caulbl,CAUVar,normal)},
  {"nuv_lower_bounds",6,0,3,0,0,0,0,0,3,N_vam(rvec,normalUncLowerBnds)},
  {"nuv_means",6,0,1,1,0,0,0,0,3,N_vam(rvec,normalUncMeans)},
  {"nuv_std_deviations",6,0,2,2,0,0,0,0,2,N_vam(RealLb,normalUncStdDevs)},
  {"nuv_upper_bounds",6,0,4,0,0,0,0,0,2,N_vam(rvec,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(rvec,normalUncUpperBnds)}
}
KeyWord kw_336[3] [static]
Initial value:
= {
  "descriptors", 15, 0, 3, 0, 0, 0, 0, 0, N_vae(dauilbl, DAUVar_poisson),
  "initial_point", 13, 0, 2, 0, 0, 0, 0, 0, N_vam[IntLb, poissonUncVars],
  "lambdas", 14, 0, 1, 1, 0, 0, 0, 0, N_vam(RealLb, poissonUncLambdas)
}

KeyWord kw_337[9] [static]
Initial value:
= {
  "descriptors", 15, 0, 5, 0, 0, 0, 0, 0, N_vae(caulbl, CAUVar_triangular),
  "initial_point", 14, 0, 4, 0, 0, 0, 0, 0, N_vam(rvec, triangularUncVars),
  "lower_bounds", 14, 0, 1, 1, 0, 0, 0, 0, -3, N_vam(RealLb, triangularUncLowerBnds),
  "modes", 14, 0, 2, 2, 0, 0, 0, 0, -3, N_vam(RealLb, triangularUncModes),
  "tuv_descriptors", 7, 0, 1, 1, 0, 0, 0, 0, -3, N_vam(rvec, triangularUncTuvDescriptors),
  "tuv_lower_bounds", 6, 0, 1, 1, 0, 0, 0, 0, -3, N_vam(RealLb, triangularUncLowerBnds),
  "tuv_upper_bounds", 6, 0, 2, 2, 0, 0, 0, 0, -3, N_vam(RealLb, triangularUncUpperBnds),
  "upper_bounds", 14, 0, 3, 0, 0, 0, 0, 0, 0, N_vam(RealUb, triangularUncUpperBnds)
}

KeyWord kw_338[7] [static]
Initial value:
= {
  "descriptors", 15, 0, 4, 0, 0, 0, 0, 0, N_vae(caulbl, CAUVar_uniform),
  "initial_point", 14, 0, 3, 0, 0, 0, 0, 0, N_vam(rvec, uniformUncVars),
  "lower_bounds", 14, 0, 1, 1, 0, 0, 0, 0, N_vam(RealLb, uniformUncLowerBnds),
  "upper_bounds", 14, 0, 2, 2, 0, 0, 0, 0, N_vam(RealUb, uniformUncUpperBnds),
  "uuv_descriptors", 7, 0, 4, 0, 0, 0, 0, 0, -4, N_vae(caulbl, CAUVar_uniform),
  "uuv_lower_bounds", 6, 0, 1, 1, 0, 0, 0, 0, -4, N_vam(RealLb, uniformUncLowerBnds),
  "uuv_upper_bounds", 6, 0, 2, 2, 0, 0, 0, 0, -4, N_vam(RealUb, uniformUncUpperBnds)
}

KeyWord kw_339[7] [static]
Initial value:
= {
  "alphas", 14, 0, 1, 1, 0, 0, 0, 0, N_vam(RealLb, weibullUncAlphas),
  "betas", 14, 0, 2, 2, 0, 0, 0, 0, N_vam(RealLb, weibullUncBetas),
  "descriptors", 15, 0, 4, 0, 0, 0, 0, 0, N_vae(caulbl, CAUVar_weibull),
  "initial_point", 14, 0, 3, 0, 0, 0, 0, 0, N_vam(RealLb, weibullUncVars),
  "wuv_alphas", 6, 0, 1, 1, 0, 0, 0, 0, -4, N_vam(RealLb, weibullUncAlphas),
  "wuv_betas", 6, 0, 2, 2, 0, 0, 0, 0, -4, N_vam(RealLb, weibullUncBetas),
  "wuv_descriptors", 7, 0, 4, 0, 0, 0, 0, 0, -4, N_vae(caulbl, CAUVar_weibull)
}

KeyWord kw_341[6] [static]
Initial value:
= {
  "environment", 0x108, 6, 1, 1, kw_3, 0, 0, 0, 0, NIDRProblemDescDB::env_start,
  "interface", 0x308, 9, 5, 5, kw_18, 0, 0, 0, 0, N_ifm3(start, 0, stop),
  "method", 0x308, 90, 2, 2, kw_257, 0, 0, 0, N_mdm3(start, 0, stop),
  "model", 8, 7, 3, kw_288, 0, 0, 0, N_mcm3(start, 0, stop),
  "responses", 0x308, 19, 6, 6, kw_306, 0, 0, 0, N_rem3(start, 0, stop),
  "variables", 0x308, 37, 4, 4, kw_340, 0, 0, 0, 0, N_vam3(start, 0, stop)
}
Var_uinfo CAUVLbl[CAUVar_Nkinds]  [static]

Initial value:

= {
  VarLabelInfo(nuv", NormalUnc),
  VarLabelInfo(lnuv", LognormalUnc),
  VarLabelInfo(uuv", UniformUnc),
  VarLabelInfo(luuv", LoguniformUnc),
  VarLabelInfo(tuv", TriangularUnc),
  VarLabelInfo(euv", ExponentialUnc),
  VarLabelInfo(beuv", BetaUnc),
  VarLabelInfo(gauv", GammaUnc),
  VarLabelInfo(guuv", GumbelUnc),
  VarLabelInfo(fuv", FrechetUnc),
  VarLabelInfo(wuv", WeibullUnc),
  VarLabelInfo(hbuv", HistogramBinUnc)
}

Var_uinfo DAUIVLbl[DAUIVar_Nkinds]  [static]

Initial value:

= {
  VarLabelInfo(puv", PoissonUnc),
  VarLabelInfo(biuv", BinomialUnc),
  VarLabelInfo(nbuv", NegBinomialUnc),
  VarLabelInfo(geuv", GeometricUnc),
  VarLabelInfo(hguv", HyperGeomUnc)
}

Var_uinfo DAURVLbl[DAURVar_Nkinds]  [static]

Initial value:

= {
  VarLabelInfo(hpuv", HistogramPtUnc)
}

Var_uinfo CEUVLbl[CEUVar_Nkinds]  [static]

Initial value:

= {
  VarLabelInfo(ciuv", ContinuousIntervalUnc)
}

Var_uinfo DEUIVLbl[DEUIVar_Nkinds]  [static]

Initial value:

= {
  VarLabelInfo(diuv", DiscreteIntervalUnc),
  VarLabelInfo(disiv", DiscreteUncSetInt)
12.1. DAKOTA NAMESPACE REFERENCE

Var_uinfo DEURVLbl[DEURVar_Nkinds] [static]

Initial value:

= {
    VarLabelInfo(duusrv, DiscreteUncSetReal)
}

Var_uinfo DiscSetLbl[DiscSetVar_Nkinds] [static]

Initial value:

= {
    VarLabelInfo(ddsisv, DiscreteDesSetInt),
    VarLabelInfo(ddsrv, DiscreteDesSetReal),
    VarLabelInfo(dssisv, DiscreteStateSetInt),
    VarLabelInfo(dssrv, DiscreteStateSetReal)
}

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 numContinuousStateVars, AVI continuousStateLabels, "csv", "csv_descriptors" },
    AVI numDiscreteStateRangeVars, AVI discreteStateRangeLabels, "dsriv", "dsriv_descriptors" },
    AVI numDiscreteStateSetIntVars, AVI discreteStateSetIntLabels, "dssiv", "dssiv_descriptors" },
    AVI numContinuousDesVars, AVI continuousDesignScaleTypes, 0, "cdv_scale_types"
}

VLreal VLR[N_VLR] [static]

Initial value:

= {
    {CAUVar_Nkinds, AVI CAUv, CAUVLabel, 
        DVR continuousAleatoryUncLabels, 
        DVR continuousAleatoryUncLowerBnds, 
        DVR continuousAleatoryUncUpperBnds, 
        DVR continuousAleatoryUncVars},
    {CEUVar_Nkinds, AVI CEUv, CEUVLbl, 
        DVR continuousEpistemicUncLabels, 
        DVR continuousEpistemicUncLowerBnds, 
        DVR continuousEpistemicUncUpperBnds, 
        DVR continuousEpistemicUncVars},
    {DAURVar_Nkinds, AVI DAURv, DAURVLbl, 
        DVR discreteRealAleatoryUncLabels, 
        DVR discreteRealAleatoryUncLowerBnds, 
        DVR discreteRealAleatoryUncUpperBnds, 
        DVR discreteRealAleatoryUncVars},
    {DEURVar_Nkinds, AVI DEURv, DEURVLbl, 
        DVR discreteRealEpistemicUncLabels, 
        DVR discreteRealEpistemicUncLowerBnds, 
        DVR discreteRealEpistemicUncUpperBnds, 
        DVR discreteRealEpistemicUncVars}
VLInt VLI[N_VLI] [static]
Initial value:
= \{
  \{DAUIVarNkinds, AVI DAUIv, DAUIVLbl,
  DVR discreteIntAleatoryUncLabels,
  DVR discreteIntAleatoryUncLowerBnds,
  DVR discreteIntAleatoryUncUpperBnds,
  DVR discreteIntAleatoryUncVars\},
  \{DEUIVarNkinds, AVI DEUIv, DEUIVLbl,
  DVR discreteIntEpistemicUncLabels,
  DVR discreteIntEpistemicUncLowerBnds,
  DVR discreteIntEpistemicUncUpperBnds,
  DVR discreteIntEpistemicUncVars\}\}

Var_check var_mp_check_cv[] [static]
Initial value:
= \{
  Vchk3(continuous_design, ContinuousDes),
  Vchk3(continuous_state, ContinuousState) \}

Var_check var_mp_check_dset[] [static]
Initial value:
= \{
  Vchk3(discrete_design_set_integer, DiscreteDesSetInt),
  Vchk3(discrete_design_set_real, DiscreteDesSetReal),
  Vchk3(discrete_state_set_integer, DiscreteStateSetInt),
  Vchk3(discrete_state_set_real, DiscreteStateSetReal) \}

Var_check var_mp_check_cau[] [static]
Initial value:
= \{
  Vchk3(normal_uncertain, NormalUnc),
  Vchk3(lognormal_uncertain, LognormalUnc),
  Vchk3(uniform_uncertain, UniformUnc),
  Vchk3(loguniform_uncertain, LoguniformUnc),
  Vchk3(triangular_uncertain, TriangularUnc),
  Vchk3(exponential_uncertain, ExponentialUnc),
  Vchk3(beta_uncertain, BetaUnc),
  Vchk3(gamma_uncertain, GammaUnc),
  Vchk3(gumbel_uncertain, GumbelUnc),
  Vchk3(frechet_uncertain, FrechetUnc),
  Vchk3(weibull_uncertain, WeibullUnc),
  Vchk3(histogram_bin_uncertain, HistogramBinUnc) \}

Var_check var_mp_check_daui[] [static]
Initial value:
= \{
  Vchk3(poisson_uncertain, PoissonUnc),
  Vchk3(binomial_uncertain, BinomialUnc),
  Vchk3(negative_binomial_uncertain, NegBinomialUnc),
  Vchk3(geometric_uncertain, GeometricUnc),
  Vchk3(hypergeometric_uncertain, HyperGeomUnc) \}
12.1. DAKOTA NAMESPACE REFERENCE

Var_check var_mp_check_daur[] [static]
Initial value:

= { Vchk_3(histogram_point_uncertain, HistogramPtUnc) }

Var_check var_mp_check_ceu[] [static]
Initial value:

= { Vchk_3(continuous_interval_uncertain, ContinuousIntervalUnc) }

Var_check var_mp_check_deui[] [static]
Initial value:

= { Vchk_3(discrete_interval_uncertain, DiscreteIntervalUnc),
    Vchk_3(discrete_uncertain_set_integer, DiscreteUncSetInt) }

Var_check var_mp_check_deur[] [static]
Initial value:

= { Vchk_3(discrete_uncertain_set_real, DiscreteUncSetReal) }

Var_rcheck var_mp_cbound[] [static]
Initial value:

= { Vchk_7(continuous_design, ContinuousDes, continuousDesign),
    Vchk_7(continuous_state, ContinuousState, continuousState),
    Vchk_5(normal_uncertain, NormalUnc, normalUnc),
    Vchk_5(lognormal_uncertain, LognormalUnc, lognormalUnc),
    Vchk_5(uniform_uncertain, UniformUnc, uniformUnc),
    Vchk_5(loguniform_uncertain, LoguniformUnc, loguniformUnc),
    Vchk_5(triangular_uncertain, TriangularUnc, triangularUnc),
    Vchk_5(beta_uncertain, BetaUnc, betaUnc) }

Var_icheck var_mp_drange[] [static]
Initial value:

= { Vchk_7(discrete_design_range, DiscreteDesRange, discreteDesignRange),
    Vchk_7(discrete_state_range, DiscreteStateRange, discreteStateRange) }
const char* SCI_FIELD_NAMES[]

Initial value:

```c
= { "dakota_type", "numFns", "numVars", "numACV", "numADIV",
  "numADRV", "numDerivVars", "xC", "xD1",
  "xD2", "xLabels", "xDILabels",
  "xDRLabels", "directFnASV", "directFnASM",
  "directFnDVV", "directFnDVV_bool",
  "fnFlag", "gradFlag", "hessFlag",
  "fnVals", "fnGrads", "fnHessians",
  "fnLabels", "failure", "currEvalId" }
```

fields to pass to Scilab in Dakota structure
Referenced by ScilabInterface::scilab_engine_run().

```c
const int SCI_NUMBER_OF_FIELDS = 26
```

number of fields in above structure
Referenced by ScilabInterface::scilab_engine_run().

## 12.2 SIM Namespace Reference

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

### Classes

- class **ParallelDirectApplicInterface**
  Sample derived interface class for testing parallel simulator plug-ins using assign_rep().
- class **SerialDirectApplicInterface**
  Sample derived interface class for testing serial simulator plug-ins using assign_rep().

### 12.2.1 Detailed Description

A sample namespace for derived classes that use assign_rep() to plug facilities into DAKOTA. A typical use of plug-ins with assign_rep() is to publish a simulation interface for use in library mode See Interfacing with Dakota as a Library for more information.
Chapter 13

Class Documentation

13.1 ActiveSet Class Reference

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

Public Member Functions

- **ActiveSet ()**
  default constructor
- **ActiveSet (size_t num_fns, size_t num_deriv_vars)**
  standard constructor
- **ActiveSet (size_t num_fns)**
  partial constructor
- **ActiveSet (const ActiveSet &set)**
  copy constructor
- **~ActiveSet ()**
  destructor
- **ActiveSet & operator= (const ActiveSet &set)**
  assignment operator
- **void reshape (size_t num_fns, size_t num_deriv_vars)**
  reshape requestVector and derivVarsVector
- **void reshape (size_t num_fns)**
  reshape requestVector
- **const ShortArray & request_vector () const**
  return the request vector
- **void request_vector (const ShortArray &rv)**
  set the request vector
- **void request_values (const short rv_val)**
  set all request vector values
- **short request_value (const size_t index) const**
  get the value of an entry in the request vector
• void request_value (const short rv_val, const size_t index)
  set the value of an entry in the request vector
• const SizetArray & derivative_vector () const
  return the derivative variables vector
• void derivative_vector (const SizetArray &dvv)
  set the derivative variables vector from a SizetArray
• void derivative_vector (SizetMultiArrayConstView dvv)
  set the derivative variables vector from a SizetMultiArrayConstView
• void derivative_start_value (size_t dvv_start_val)
  set the derivative variables vector values
• void read (std::istream &s)
  read an active set object from an std::istream
• void write (std::ostream &s) const
  write an active set object to an std::ostream
• void write.annotated (std::ostream &s) const
  write an active set object to an std::ostream in annotated format
• void read (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 Member Functions
• template<class Archive >
  void serialize (Archive &ar, const unsigned int version)
  implementation of Boost serialize for ActiveSet

Private Attributes
• ShortArray requestVector
  the vector of response requests
• SizetArray derivVarsVector
  the vector of variable ids used for computing derivatives

Friends
• class boost::serialization::access
• bool operator== (const ActiveSet &set1, const ActiveSet &set2)
  equality operator
• bool operator!= (const ActiveSet &set1, const ActiveSet &set2)
  inequality operator
13.2 ANALYZER CLASS REFERENCE

13.1.1 Detailed Description

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

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

13.1.2 Member Data Documentation

**ShortArray requestVector** [private]

the vector of response requests

It uses a 0 value for inactive functions and sums 1 (value), 2 (gradient), and 4 (Hessian) for active functions.

Referenced by ActiveSet::ActiveSet(), ActiveSet::operator=(), ActiveSet::read(), ActiveSet::request_value(), ActiveSet::request_values(), ActiveSet::request_vector(), ActiveSet::reshape(), ActiveSet::write(), and ActiveSet::write_annotated().

**SizetArray derivVarsVector** [private]

the vector of variable ids used for computing derivatives

These ids will generally identify either the active continuous variables or the inactive continuous variables.

Referenced by ActiveSet::ActiveSet(), ActiveSet::derivative_start_value(), ActiveSet::derivative_vector(), ActiveSet::operator=(), ActiveSet::read(), ActiveSet::reshape(), ActiveSet::write(), and ActiveSet::write_annotated().

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

- DakotaActiveSet.hpp

13.2 Analyzer Class Reference

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

Inheritance diagram for Analyzer:
Public Member Functions

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

- const RealMatrix & all_samples ()
  
  return the complete set of evaluated samples

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

- int num_samples () const

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

Protected Member Functions

- Analyzer ()
  
  default constructor

- Analyzer (ProblemDescDB &problem_db, Model &model)
  
  standard constructor

- Analyzer (unsigned short method_name, Model &model)
  
  alternate constructor for instantiations "on the fly" with a Model

- Analyzer (unsigned short method_name)
  
  alternate constructor for instantiations "on the fly" without a Model

- ~Analyzer ()
  
  destructor

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

- virtual void update_model_from_sample (Model &model, const Real *sample_vars)
13.2. ANALYZER CLASS REFERENCE

update model’s current variables with data from sample

- virtual void update_model_from_variables (Model &model, const Variables &vars)
  update model’s current variables with data from vars

- void update_from_model (const Model &model)
  set inherited data attributes based on extractions from incoming model

- void initialize_run ()
  utility function to perform common operations prior to pre_run(); typically memory initialization; setting of instance pointers

- void post_run (std::ostream &s)
  post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/-Responses and perform final analysis phase in a standalone way

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

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

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

- const VariablesArray & variables_array_results ()
  return multiple final iterator solutions (variables). This should only be used if returns_multiple_points() returns true.

- const ResponseArray & response_array_results ()
  return multiple final iterator solutions (response). This should only be used if returns_multiple_points() returns true.

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

- bool compact_mode () const
  returns Analyzer::compactMode

- bool returns_multiple_points () const
  indicates if this iterator returns multiple final points. Default return is false. Override to return true if appropriate.

- void evaluate_parameter_sets (Model &model, bool log_resp_flag, bool log_best_flag)
  perform function evaluations to map parameter sets (allVariables) into response sets (allResponses)

- void variance_based_decomp (int ncont, int ndiscint, int ndiscreal, int num_samples)
- void read_variables_responses (int num_evals, size_t num_vars)
  convenience function for reading variables/responses (used in derived classes post_input)

- void print_sobol_indices (std::ostream &s) const
  Printing of VBD results.

- void sample_to_variables (const Real *sample_c_vars, Variables &vars)
  convert samples array to variables array; e.g., allSamples to allVariables

- void samples_to_variables_array (const RealMatrix &sample_matrix, VariablesArray &vars_array)
  convert samples array to variables array; e.g., allSamples to allVariables

- void variables_array_to_samples (const VariablesArray &vars_array, RealMatrix &sample_matrix)
  convert variables array to samples array; e.g., allVariables to allSamples
Protected Attributes

- `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
- `bool compactMode`
  switch for `allSamples` (compact mode) instead of `allVariables` (normal mode)
- `VariablesArray allVariables`
  array of all variables to be evaluated in `evaluate_parameter_sets()`
- `RealMatrix allSamples`
  compact alternative to `allVariables`
- `IntResponseMap allResponses`
  array of all responses to be computed in `evaluate_parameter_sets()`
- `StringArray allHeaders`
  array of headers to insert into output while evaluating `allVariables`
- `size_t numObjFns`
  number of objective functions
- `size_t numLSqTerms`
  number of least squares terms
- `RealPairPRPMultiMap bestVarsRespMap`
  map which stores best set of solutions

Private Member Functions

- `void compute_best_metrics (const Response &response, std::pair<Real, Real> &metrics)`
  compares current evaluation to best evaluation and updates best
- `void update_best (const Variables &vars, int eval_id, const Response &response)`
  compares current evaluation to best evaluation and updates best
- `void update_best (const Real *sample_c_vars, int eval_id, const Response &response)`
  compares current evaluation to best evaluation and updates best

Private Attributes

- `int writePrecision`
  write precision as specified by the user
- `Real vbdDropTol`
  tolerance for omitting output of small VBD indices
- `RealVectorArray S4`
  VBD main effect indices.
- `RealVectorArray T4`
  VBD total effect indices.
13.2. ANALYZER CLASS REFERENCE

13.2.1 Detailed Description

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

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

13.2.2 Member Function Documentation

int num_samples ( ) const [inline], [virtual]

Return current number of evaluation points. Since the calculation of samples, collocation points, etc. might be
costly, provide a default implementation here that backs out from the maxEvalConcurrency.

Reimplemented from Iterator.
Reimplemented in NonDQuadrature, NonDSparseGrid, NonDSampling, NonDCubature, DDAECDesign-CompExp, FSUDesignCompExp, and PSUADEDesignCompExp.

References Model::derivative_concurrency(), Iterator::iteratedModel, and Iterator::maxEvalConcurrency.
Referenced by NonDGlobalReliability::get_best_sample(), Analyzer::samples_to_variables_array(), Analyzer::variables_array_to_samples(), and Analyzer::variance_based_decomp().

void initialize_run ( ) [protected], [virtual]

utility function to perform common operations prior to pre_run(); typically memory initialization; setting of in-
stance pointers

Perform initialization phases of run sequence, like allocating memory and setting instance pointers. Com-
monly used in sub-iterator executions. This is a virtual function; when re-implementing, a derived class must call
its nearest parent’s initialize_run(), typically before performing its own implementation steps.

Reimplemented from Iterator.
Reimplemented in NonD.
References Model::is_null(), Iterator::iteratedModel, Iterator::maxEvalConcurrency, Model::set_communicators(),
Model::set_evaluation_reference(), and Iterator::summaryOutputFlag.
Referenced by NonD::initialize_run().

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

post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/-
Responses and perform final analysis phase in a standalone way

Post-run phase, which a derived iterator may optionally reimplement; when not present, post-run is likely
integrated into run. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s
post_run(), typically after performing its own implementation steps.

Reimplemented from Iterator.
Reimplemented in NonDLHSSampling, DDAECDesignCompExp, ParamStudy, FSUDesignCompExp, and
PSUADEDesignCompExp.

References Model::is_null(), Iterator::iteratedModel, Model::print_evaluation_summary(), Analyzer::print_-results(), Iterator::resultsDB, Iterator::summaryOutputFlag, and ResultsManager::write_databases().
Referenced by PSUADEDesignCompExp::post_run(), FSUDesignCompExp::post_run(), ParamStudy::post_-run(), DDAECDesignCompExp::post_run(), and NonDLHSSampling::post_run().

void pre_output ( ) [protected], [virtual]

Generate tabular output with active variables (compactMode) or all variables with their labels and response labels,
with no data. Variables are sequenced \{cv, div, drv\}
Reimplemented from `Iterator`.
References `Analyzer::allSamples`, `Analyzer::allVariables`, `ParallelLibrary::command_line_pre_run_output()`, `ParallelLibrary::command_line_user_modes()`, `Analyzer::compactMode`, `Model::current_response()`, `Model::current_variables()`, `Iterator::iteratedModel`, `Iterator::outputLevel`, `Model::parallel_library()`, `Dakota::write_precision`, and `Analyzer::writePrecision`.

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

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

Reimplemented from `Iterator`.

Reimplemented in `NonDLHSSampling`, `NonDPOFDarts`, `NonDLocalReliability`, `NonDAdaptiveSampling`, `NonDGPImpSampling`, `NonDAdaptImpSampling`, `NonDIncremLHSSampling`, `NonDGlobalReliability`, `NonDInterval`, `NonDExpansion`, `PStudyDACE`, `RichExtrapVerification`, and `Verification`.

References `Analyzer::bestVarsRespMap`, `ParamResponsePair::eval_id()`, `Response::function_values()`, `Analyzer::numLSqTerms`, `Analyzer::numObjFns`, `ParamResponsePair::prp_parameters()`, and `ParamResponsePair::prp_response()`.

Referenced by `Analyzer::post_run()`, `Verification::print_results()`, `PStudyDACE::print_results()`, and `NonDLHSSampling::print_results()`.

```cpp
const Model & algorithm_space_model ( ) const [inline], [protected], [virtual]
```

default definition that gets redefined in selected derived Minimizers

Reimplemented from `Iterator`.

Reimplemented in `NonDExpansion`, `NonDGlobalInterval`, `NonDBayesCalibration`, and `NonDReliability`.

References `Iterator::iteratedModel`.

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

perform function evaluations to map parameter sets (`allVariables`) into response sets (`allResponses`)

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

References `Iterator::activeSet`, `Analyzer::allHeaders`, `Analyzer::allResponses`, `Analyzer::allSamples`, `Analyzer::allVariables`, `Model::asynch_compute_response()`, `Model::asynch_flag()`, `Analyzer::compactMode`, `Model::compute_response()`, `Response::copy()`, `Model::current_response()`, `Model::current_variables()`, `Model::evaluation_id()`, `Model::synchronize()`, `Analyzer::update_best()`, `Analyzer::update_model_from_sample()`, and `Analyzer::update_model_from_variables()`.

Referenced by `NonDSparseGrid::evaluate_grid_increment()`, `NonDSparseGrid::evaluate_set()`, `PSUADEDesignCompExp::extract_trends()`, `PSUADEDesignCompExp::extract_trends()`, `ParamStudy::extract_trends()`, `DDACEDesignCompExp::extract_trends()`, `NonDIncremLHSSampling::quantify_uncertainty()`, `NonDAdaptImpSampling::quantify_uncertainty()`, `NonDLHSSampling::quantify_uncertainty()`, `NonDIntegration::quantify_uncertainty()`, and `Analyzer::variance_based_decomp()`.

```cpp
void variance_based_decomp ( int ncont, int ndiscint, int ndiscreal, int num_samples ) [protected]
```

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

References Dakota::abort_handler(), Analyzer::allResponses, Analyzer::allSamples, Analyzer::allVariables, Analyzer::compactMode, Variables::continuous_variables(), Dakota::copy_data(), Variables::discrete_int_variables(), Variables::discrete_real_variables(), Analyzer::evaluate_parameter_sets(), Analyzer::get_parameter_sets(), Iterator::iteratedModel, Analyzer::num_samples(), Analyzer::numFunctions, Analyzer::S4, Analyzer::T4, and Analyzer::vary_pattern().

Referenced by FSUDesignCompExp::extract_trends(), DDACEDesignCompExp::extract_trends(), and NonDLHSSampling::quantify_uncertainty().

```cpp
void read_variables_responses ( int num_evals, size_t num_vars ) [protected]
```

Convenience function for reading variables/responses (used in derived classes post_input)

Read num_evals variables/responses from file

References Dakota::abort_handler(), Analyzer::allResponses, Analyzer::allSamples, Analyzer::allVariables, ParallelLibrary::command_line_post_run_input(), ParallelLibrary::command_line_user_modes(), Analyzer::compactMode, Variables::copy(), Response::copy(), Model::current_response(), Model::current_variables(), Iterator::iteratedModel, Iterator::outputLevel, Model::parallel_library(), and Analyzer::update_best().

Referenced by PSUADEDesignCompExp::post_input(), FSUDesignCompExp::post_input(), ParamStudy::post_input(), DDACEDesignCompExp::post_input(), and NonDLHSSampling::post_input().

```cpp
void print_sobol_indices ( std::ostream & s ) const [protected]
```

Printing of VBD results.

Printing of variance based decomposition indices.

References Model::continuous_variable_labels(), Model::discrete_int_variable_labels(), Model::discrete_real_variable_labels(), Iterator::iteratedModel, Analyzer::numContinuousVars, Analyzer::numDiscreteIntVars, Analyzer::numDiscreteRealVars, Analyzer::numFunctions, Model::response_labels(), Analyzer::S4, Analyzer::T4, Analyzer::vbdDropTol, and Dakota::write_precision.

Referenced by PStudyDACE::print_results(), and NonDLHSSampling::print_results().

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

- DakotaAnalyzer.hpp
- DakotaAnalyzer.cpp

13.3 ApplicationInterface Class Reference

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

Inheritance diagram for ApplicationInterface:
Public Member Functions

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

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

Protected Member Functions

- **void init_communicators** (const IntArray &message_lengths, int max_eval_concurrency)
  
  Allocate communicator partitions for concurrent evaluations within an iterator and concurrent multiprocessor analyses within an evaluation.

- **void set_communicators** (const IntArray &message_lengths, int max_eval_concurrency)
  
  Set the local parallel partition data for an interface (the partitions are already allocated in ParallelLibrary).

- **void free_communicators** ()
  
  Deallocate communicator partitions for concurrent evaluations within an iterator and concurrent multiprocessor analyses within an evaluation.

- **void init_serial** ()

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

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

- **bool evaluation_cache** () const
  
  Return evalCacheFlag

- **void map** (const Variables &vars, const ActiveSet &set, Response &response, bool asynch_flag=false)
  
  Provides a "mapping" of variables to responses using a simulation. Protected due to Interface letter-envelope idiom.

- **void manage_failure** (const Variables &vars, const ActiveSet &set, Response &response, int failed_eval_id)
  
  Manages a simulation failure using abort/retry/recover/continuation
• const IntResponseMap & synch ()
  executes a blocking schedule for asynchronous evaluations in the beforeSynchCorePRPQueue and returns all jobs
• const IntResponseMap & synch_nowait ()
  executes a nonblocking schedule for asynchronous evaluations in the beforeSynchCorePRPQueue and returns a partial set of completed jobs
• void serve_evaluations ()
  run on evaluation servers to serve the iterator master
• void stop_evaluation_servers ()
  used by the iterator master to terminate evaluation servers
• bool check_multiprocessor_analysis (bool warn)
  checks on multiprocessor analysis configuration
• bool check_asynchronous (bool warn, int max_eval_concurrency)
  checks on asynchronous configuration (for direct interfaces)
• bool check_multiprocessor_asynchronous (bool warn, int max_eval_concurrency)
  checks on asynchronous settings for multiprocessor partitions
• virtual void derived_map (const Variables &vars, const ActiveSet &set, Response &response, int fn_eval_id)
  Called by map() and other functions to execute the simulation in synchronous mode. The portion of performing an evaluation that is specific to a derived class.
• virtual void derived_map_asynch (const ParamResponsePair &pair)
  Called by map() and other functions to execute the simulation in asynchronous mode. The portion of performing an asynchronous evaluation that is specific to a derived class.
• virtual void wait_local_evaluations (PRPQueue &prp_queue)
  For asynchronous function evaluations, this method is used to detect completion of jobs and process their results. It provides the processing code that is specific to derived classes. This version waits for at least one completion.
• virtual void test_local_evaluations (PRPQueue &prp_queue)
  For asynchronous function evaluations, this method is used to detect completion of jobs and process their results. It provides the processing code that is specific to derived classes. This version is nonblocking and will return without any completions if none are immediately available.
• virtual void init_communicators_checks (int max_eval_concurrency)
  perform construct-time error checks on the parallel configuration
• virtual void set_communicators_checks (int max_eval_concurrency)
  perform run-time error checks on the parallel configuration
• void master_dynamic_schedule_analyses ()
  blocking dynamic schedule of all analyses within a function evaluation using message passing
• void serve_analyses_synch ()
  serve the master analysis scheduler and manage one synchronous analysis job at a time
• virtual int synchronous_local_analysis (int analysis_id)
  Execute a particular analysis (identified by analysis_id) synchronously on the local processor. Used for the derived class specifics within ApplicationInterface::serve_analyses_synch().
Protected Attributes

- **ParallelLibrary & parallelLib**
  reference to the ParallelLibrary object used to manage MPI partitions for the concurrent evaluations and concurrent analyses parallelism levels
- **bool suppressOutput**
  flag for suppressing output on slave processors
- **int evalCommSize**
  size of evalComm
- **int evalCommRank**
  processor rank within evalComm
- **int evalServerId**
  evaluation server identifier
- **bool eaDedMasterFlag**
  flag for dedicated master partitioning at ea level
- **int analysisCommSize**
  size of analysisComm
- **int analysisCommRank**
  processor rank within analysisComm
- **int analysisServerId**
  analysis server identifier
- **int numAnalysisServers**
  current number of analysis servers
- **bool multiProcAnalysisFlag**
  flag for multiprocessor analysis partitions
- **bool asynchLocalAnalysisFlag**
  flag for asynchronous local parallelism of analyses
- **int asynchLocalAnalysisConcurrency**
  limits the number of concurrent analyses in asynchronous local scheduling and specifies hybrid concurrency when message passing
- **int numAnalysisDrivers**
  the number of analysis drivers used for each function evaluation (from the analysis drivers interface specification)
- **IntSet completionSet**
  the set of completed fn_eval_id’s populated by wait_local_evaluations() and test_local_evaluations()

Private Member Functions

- **bool duplication_detect (const Variables &vars, Response &response, bool asynch_flag)**
  checks data pairs and beforeSynchCorePRPQueue to see if the current evaluation request has already been performed or queued
- **void master_dynamic_schedule_evaluations ()**
  blocking dynamic schedule of all evaluations in beforeSynchCorePRPQueue using message passing on a dedicated master partition; executes on iteratorComm master
- **void peer_static_schedule_evaluations ()**
13.3. APPLICATIONINTERFACE CLASS REFERENCE

blocking static schedule of all evaluations in beforeSynchCorePRPQueue using message passing on a peer partition; executes on iteratorComm master

• void peer\_dynamic\_schedule\_evaluations ()

blocking dynamic schedule of all evaluations in beforeSynchCorePRPQueue using message passing on a peer partition; executes on iteratorComm master

• void asynchronous\_local\_evaluations (PRPQueue &prp\_queue)

perform all jobs in prp\_queue using asynchronous approaches on the local processor

• void synchronous\_local\_evaluations (PRPQueue &prp\_queue)

perform all jobs in prp\_queue using synchronous approaches on the local processor

• void master\_dynamic\_schedule\_evaluations\_nowait ()

execute a nonblocking dynamic schedule in a master-slave partition

• void peer\_dynamic\_schedule\_evaluations\_nowait ()

execute a nonblocking static/dynamic schedule in a peer partition

• void asynchronous\_local\_evaluations\_nowait (PRPQueue &prp\_queue)

launch new jobs in prp\_queue asynchronously (if capacity is available), perform nonblocking query of all running jobs, and process any completed jobs (handles both local master- and local peer-scheduling cases)

• void broadcast\_evaluation (const ParamResponsePair &pair)

convenience function for broadcasting an evaluation over an evalComm

• void broadcast\_evaluation (int fn\_eval\_id, const Variables &vars, const ActiveSet &set)

convenience function for broadcasting an evaluation over an evalComm

• void send\_evaluation (PRPQueueIter &prp\_it, size\_t buff\_index, int server\_id, bool peer\_flag)

helper function for sending sendBuffers[buff\_index] to server

• void receive\_evaluation (PRPQueueIter &prp\_it, size\_t buff\_index, int server\_id, bool peer\_flag)

helper function for processing recvBuffers[buff\_index] within scheduler

• void launch\_asynch\_local (PRPQueueIter &prp\_it)

launch an asynchronous local evaluation

• void process\_asynch\_local (int fn\_eval\_id)

process a completed asynchronous local evaluation

• void process\_synch\_local (PRPQueueIter &prp\_it)

process a completed synchronous local evaluation

• void assign\_asynch\_local\_queue (PRPQueue &local\_prp\_queue, PRPQueueIter &local\_prp\_iter)

helper function for creating an initial active local queue by launching asynch local jobs from local\_prp\_queue, as limited by server capacity

• void assign\_asynch\_local\_queue\_nowait (PRPQueue &local\_prp\_queue, PRPQueueIter &local\_prp\_iter)

helper function for updating an active local queue by backfilling asynch local jobs from local\_prp\_queue, as limited by server capacity

• size\_t test\_local\_backfill (PRPQueue &assign\_queue, PRPQueueIter &assign\_iter)

helper function for testing active asynch local jobs and then backfilling

• size\_t test\_receives\_backfill (PRPQueueIter &assign\_iter, bool peer\_flag)

helper function for testing receive requests and then backfilling jobs

• void serve\_evaluations\_synch ()

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

• void serve\_evaluations\_synch\_peer ()
serve the evaluation message passing schedulers and perform one synchronous evaluation at a time as part of the 1st peer

- **void serve_evaluations_asynch ();**
  
  serve the evaluation message passing schedulers and manage multiple asynchronous evaluations

- **void serve_evaluations_asynch_peer ();**
  
  serve the evaluation message passing schedulers and perform multiple asynchronous evaluations as part of the 1st peer

- **void set_evaluation_communicators (const IntArray &message_lengths);**
  
  convenience function for updating the local evaluation partition data following `ParallelLibrary::init_evaluation_communicators()`.

- **void set_analysis_communicators ();**
  
  convenience function for updating the local analysis partition data following `ParallelLibrary::init_analysis_communicators()`.

- **void init_serial_evaluations ();**
  
  set concurrent evaluation configuration for serial operations

- **void init_serial_analyses ();**
  
  set concurrent analysis configuration for serial operations (e.g., for local executions on a dedicated master)

- **const ParamResponsePair & get_source_pair (const Variables &target_vars);**
  
  convenience function for the continuation approach in `manage_failure()` for finding the nearest successful “source” evaluation to the failed “target”

- **void continuation (const Variables &target_vars, const ActiveSet &set, Response &response, const ParamResponsePair &source_pair, int failed_eval_id);**
  
  performs a 0th order continuation method to step from a successful “source” evaluation to the failed “target”. Invoked by `manage_failure()` for failAction == “continuation”.

- **void common_input_filtering (const Variables &vars);**
  
  common input filtering operations, e.g. mesh movement

- **void common_output_filtering (Response &response);**
  
  common output filtering operations, e.g. data filtering

### Private Attributes

- **int worldSize**
  
  size of `MPI_COMM_WORLD`

- **int worldRank**
  
  processor rank within `MPI_COMM_WORLD`

- **int iteratorCommSize**
  
  size of iteratorComm

- **int iteratorCommRank**
  
  processor rank within iteratorComm

- **bool ieMessagePass**
  
  flag for message passing at ie scheduling level

- **int numEvalServers**
  
  current number of evaluation servers

- **int numEvalServersSpec**
  
  user specification for number of evaluation servers

- **int procsPerEvalSpec**
user specification for processors per analysis servers

- bool eaMessagePass
  flag for message passing at ea scheduling level

- int numAnalysisServersSpec
  user spec for number of analysis servers

- int procsPerAnalysisSpec
  user specification for processors per analysis servers

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

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

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

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

- short evalScheduling
  user specification of evaluation scheduling algorithm: {DEFAULT,MASTER,PEER_DYNAMIC,PEER_STATIC}.
  - SCHEDULING. Used for manual overrides of auto-configure logic in ParallelLibrary::resolve_inputs().

- short analysisScheduling
  user specification of analysis scheduling algorithm: {DEFAULT,MASTER,PEER}.
  - SCHEDULING. Used for manual overrides of the auto-configure logic in ParallelLibrary::resolve_inputs().

- int asynchLocalEvalConcSpec
  user specification for asynchronous local evaluation concurrency

- int asynchLocalAnalysisConcSpec
  user specification for asynchronous local analysis concurrency

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

- bool asynchLocalEvalStatic
  whether the asynchronous local evaluations are to be performed with a static schedule (default false)

- BitArray localServerAssigned
  array with one bit per logical "server" indicating whether a job is currently running on the server (used for asynch local static schedules)

- String interfaceSynchronization
  interface synchronization specification: synchronous (default) or asynchronous

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

- bool evalCacheFlag
  used to manage a user request to deactivate the function evaluation cache (i.e., queries and insertions using the data.pairs cache).

- bool nearbyDuplicateDetect
  flag indicating optional usage of tolerance-based duplication detection (less efficient, but helpful when experiencing restart cache misses)
• Real **real**Tolerance
  tolerance value for tolerance-based duplication detection

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

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

• String **failAction**
  mitigation action for captured simulation failures: abort, retry, recover, or continuation

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

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

• IntResponseMap **historyDuplicateMap**
  used to bookkeep asynchronous evaluations which duplicate data_pairs evaluations. Map key is evalIdCntr, map value is corresponding response.

• std::map< int, std::pair< PRPQueueHIter, Response > > **beforeSynchDuplicateMap**
  used to bookkeep evalIdCntr, beforeSynchCorePRPQueue iterator, and response of asynchronous evaluations which duplicate queued beforeSynchCorePRPQueue evaluations

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

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

• PRPQueue **asyncLocalActivePRPQueue**
  used by nonblocking asynchronous local schedulers to bookkeep active local jobs

• std::map< int, IntIntPair > **msgPassRunningMap**
  used by nonblocking message passing schedulers to bookkeep which jobs are running remotely

• MPIPackBuffer ∗ **sendBuffers**
  array of pack buffers for evaluation jobs queued to a server

• MPIUnpackBuffer ∗ **recvBuffers**
  array of unpack buffers for evaluation jobs returned by a server

• MPI_Request ∗ **recvRequests**
  array of requests for nonblocking evaluation receives

### 13.3.1 Detailed Description

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

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

13.3.2 Member Function Documentation

```c
void init_serial( ) [inline], [protected], [virtual]
```

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

Reimplemented from Interface.

References ApplicationInterface::init_serial_analyses(), and ApplicationInterface::init_serial_evaluations().

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

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

The function evaluator for application interfaces. Called from derived_compute_response() and derived_->async_compute_response() in derived Model classes. If asynch_flag is not set, perform a blocking evaluation (using derived_map()). If asynch_flag is set, add the job to the beforeSynchCorePRPQueue queue for execution by one of the scheduler routines in synch() or synch_nowait(). Duplicate function evaluations are detected with duplication_detect().

Reimplemented from Interface.

References Response::active_set(), Interface::algebraic_mappings(), Interface::algebraicMappings, Interface::asv_mapping(), ApplicationInterface::asvControlFlag, ApplicationInterface::beforeSynchAlgPRPQueue, ApplicationInterface::beforeSynchCorePRPQueue, ApplicationInterface::broadcast_evaluation(), Response::copy(), Interface::coreMappings, Interface::currEvalId, Dakota::data_pairs, ApplicationInterface::defaultASV, ApplicationInterface::derived_map(), ApplicationInterface::duplication_detect(), ApplicationInterface::evalCacheFlag, Interface::evalIdCntr, Interface::fineGrainEvalCounters, Interface::fnGradCounter, Interface::fnHessCounter, Interface::fnLabels, Interface::fnValCounter, Response::function_labels(), Interface::init_algebraic_mappings(), Interface::interfaceId, ApplicationInterface::manage_failure(), Interface::multiProcEvalFlag, Interface::newEvalIdCntr, Interface::newFnGradCounter, Interface::newFnHessCounter, Interface::outputLevel, ApplicationInterface::parallelLib, ActiveSet::request_vector(), Interface::response_mapping(), ApplicationInterface::restartFileFlag, and ParallelLibrary::write_restart().

```c
const IntResponseMap & synch ( ) [protected], [virtual]
```

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

This function provides blocking synchronization for all cases of asynchronous evaluations, including the local asynchronous case (background system call, nonblocking fork, & multithreads), the message passing case, and the hybrid case. Called from derived_synchronize() in derived Model classes.

Reimplemented from Interface.

References Interface::algebraic_mappings(), Interface::algebraicMappings, Interface::asv_mapping(), ApplicationInterface::asynchLocalEvalStatic, ApplicationInterface::asynchronous_local_evaluations(), ApplicationInterface::beforeSynchAlgPRPQueue, ApplicationInterface::beforeSynchCorePRPQueue, ApplicationInterface::beforeSynchDuplicateMap, Interface::coreMappings, ApplicationInterface::evalScheduling, ApplicationInterface::historyDuplicateMap, Interface::ieDedMasterFlag, ApplicationInterface::ieMessagePass, Interface::interfaceId, Interface::interfaceType, ApplicationInterface::master_dynamic_schedule_evaluations(), Interface::multiProcEvalFlag, Interface::outputLevel, ApplicationInterface::peer_dynamic_schedule_evaluations(), ApplicationInterface::peer_static_schedule_evaluations(), Interface::rawResponseMap, and Interface::response_mapping().
const IntResponseMap & synch_nowait() [protected], [virtual]

executes a nonblocking schedule for asynchronous evaluations in the beforeSynchCorePRPQueue and returns a
partial set of completed jobs.

This function provides nonblocking synchronization for the local asynchronous case and selected nonblocking
message passing schedulers. Called from derived_sync_on() in derived Model classes.

Reimplemented from Interface.

References Dakota::abort_handler(), Interface::algebraic_mappings(), Interface::algebraicMappings, Interface-
::asv_mapping(), ApplicationInterface::asynchLocalEvalStatic, ApplicationInterface::asynchronous_local_evaluations-
_nowait(), ApplicationInterface::beforeSynchAlgPRPQueue, ApplicationInterface::beforeSynchCorePRPQueue,
ApplicationInterface::beforeSynchDuplicateMap, Interface::coreMappings, ParamResponsePair::eval_id(), Application-
Interface::evalScheduling, ApplicationInterface::historyDuplicateMap, Interface::ieDedMasterFlag, Application-
Interface::ieMessagePass, Interface::ifaceId, Interface::interfaceType, Dakota::lookup_by_eval_id(), Application-
Interface::master_dynamic_schedule_evaluations_nowait(), Interface::multiProcEvalFlag, Interface::outputLevel,
ApplicationInterface::peer_dynamic_schedule_evaluations_nowait(), ParamResponsePair::prp_response(), Interface-
::rawResponseMap, Interface::response_mapping(), and Response::update().

void serve_evaluations() [protected], [virtual]

run on evaluation servers to serve the iterator master

Invoked by the serve() function in derived Model classes. Passes control to serve_evaluations_synch(), serve-
evaluations_asynch(), serve_evaluations_synch_peer(), or serve_evaluations_asynch_peer() according to specified
concurrency, partition, and scheduler configuration.

Reimplemented from Interface.

References ApplicationInterface::asynchLocalEvalConcurrency, ApplicationInterface::evalServerId, Interface-
::ieDedMasterFlag, ApplicationInterface::serve_evaluations_synch(), ApplicationInterface::serve_evaluations_asynch-
_peer(), ApplicationInterface::serve_evaluations_synch(), and ApplicationInterface::serve_evaluations_synch_peer().

void stop_evaluation_servers() [protected], [virtual]

used by the iterator master to terminate evaluation servers

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

Reimplemented from Interface.

References ParallelLibrary::bcast_e(), ParallelLibrary::free(), ParallelConfiguration::ie_parallel_level(), Interface-
::ieDedMasterFlag, ParallelLibrary::isend_ie(), ApplicationInterface::iteratorCommSize, Interface::multiProcEval-
Flag, ApplicationInterface::numEvalServers, Interface::outputLevel, ParallelLibrary::parallel_configuration(), and
ApplicationInterface::parallelLib.

void init_communicators_checks(int max_eval_concurrency) [protected], [virtual]

perform construct-time error checks on the parallel configuration

Override DirectApplicInterface definition if plug-in to allow batch processing in Plugin{Serial,Parallel}Direct-
ApplicInterface.cpp

Reimplemented in DirectApplicInterface, SysCallApplicInterface, and ProcessHandleApplicInterface.

Referenced by ApplicationInterface::init_communicators().
void setCommunicatorsChecks(int maxEvalConcurrency) [protected], [virtual]

perform run-time error checks on the parallel configuration

Override DirectApplicInterface definition if plug-in to allow batch processing in Plugin{Serial,Parallel}DirectApplicInterface.cpp

Reimplemented in DirectApplicInterface, SerialDirectApplicInterface, ParallelDirectApplicInterface, SysCallApplicInterface, and ProcessHandleApplicInterface.

Referenced by ApplicationInterface::setCommunicators().

void masterDynamicScheduleAnalyses() [protected]

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

This code is called from derived classes to provide the master portion of a master-slave algorithm for the dynamic scheduling of analyses among slave servers. It is patterned after master_dynamic_schedule_evaluations().

It performs no analyses locally and matches either serve_analyses_synch() or serve_analyses_asynch() on the slave servers, depending on the value of asyncLocalAnalysisConcurrency. Dynamic scheduling assigns jobs in 2 passes. The 1st pass gives each server the same number of jobs (equal to asyncLocalAnalysisConcurrency). The 2nd pass assigns the remaining jobs to slave servers as previous jobs are completed. Single- and multilevel parallel use intra- and inter-communicators, respectively, for send/receive. Specific syntax is encapsulated within ParallelLibrary.

References ApplicationInterface::asyncLocalAnalysisConcurrency, ParallelLibrary::free(), ParallelLibrary::irecv_ea(), ParallelLibrary::isend_ea(), ApplicationInterface::numAnalysisDrivers, ApplicationInterface::numAnalysisServers, ApplicationInterface::parallelLib, ParallelLibrary::waitall(), and ParallelLibrary::waitsome().

Referenced by ProcessHandleApplicInterface::create_evaluation_process(), SysCallApplicInterface::create_evaluation_process(), and DirectApplicInterface::derived_map().

void serveAnalysesSynch() [protected]

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

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

References ApplicationInterface::analysisCommRank, ParallelLibrary::bcast_ea(), ParallelLibrary::isend_ea(), ApplicationInterface::multiProcAnalysisFlag, ApplicationInterface::parallelLib, ParallelLibrary::recv_ea(), ApplicationInterface::synchronous_local_analysis(), and ParallelLibrary::wait().

Referenced by ProcessHandleApplicInterface::create_evaluation_process(), SysCallApplicInterface::create_evaluation_process(), and DirectApplicInterface::derived_map().

bool duplicationDetect(const Variables &vars, Response &response, bool asynchFlag) [private]

checks data pairs and beforeSynchCorePRPQueue to see if the current evaluation request has already been performed or queued

Called from map() to check incoming evaluation request for duplication with content of data_pairs and beforeSynchCorePRPQueue. If duplication is detected, return true, else return false. Manage bookkeeping with 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 duplicationDetect(), even though a beforeSynchCorePRPQueue search would still be meaningful. Since the intent of this request is to streamline operations, both list searches are bypassed.
References Response::active_set(), ApplicationInterface::beforeSynchCorePRPQueue, ApplicationInterface::beforeSynchDuplicateMap, Response::copy(), Dakota::data_pairs, ParamResponsePair::eval_id(), Interface::evalIdCntr, ApplicationInterface::historyDuplicateMap, Interface::interfaceId, Dakota::lookup_by_val(), ApplicationInterface::nearbyDuplicateDetect, ApplicationInterface::nearbyTolerance, and Response::update().

Referenced by ApplicationInterface::map().

`void master_dynamic_schedule_evaluations( ) [private]`
blocking dynamic schedule of all evaluations in beforeSynchCorePRPQueue using message passing on a dedicated master partition; executes on iteratorComm master

This code is called from `synch()` to provide the master portion of a master-slave algorithm for the dynamic scheduling of evaluations among slave servers. It performs no evaluations locally and matches either `serve_evaluations_synch()` or `serve_evaluations_asynch()` on the slave servers, depending on the value of `asynchLocalEvalConcurrency`. Dynamic scheduling assigns jobs in 2 passes. The 1st pass gives each server the same number of jobs (equal to `asynchLocalEvalConcurrency`). The 2nd pass assigns the remaining jobs to slave servers as previous jobs are completed and returned. Single- and multilevel parallel use intra- and inter-communicators, respectively, for send/receive. Specific syntax is encapsulated within `ParallelLibrary`. peer

References ApplicationInterface::asynchLocalEvalConcurrency, ApplicationInterface::beforeSynchCorePRPQueue, Dakota::lookup_by_eval_id(), ApplicationInterface::numEvalServers, Interface::outputLevel, ApplicationInterface::parallelLib, ApplicationInterface::recvBuffer, ApplicationInterface::recvRequests, ApplicationInterface::send_evaluation(), ApplicationInterface::sendBuffers, ParallelLibrary::waitall(), and ParallelLibrary::waitsome().

Referenced by ApplicationInterface::synch().

`void peer_static_schedule_evaluations( ) [private]`
blocking static schedule of all evaluations in beforeSynchCorePRPQueue using message passing on a peer partition; executes on iteratorComm master

This code runs on the iteratorCommRank 0 processor (the iterator) and is called from `synch()` in order to manage a static schedule for cases where peer 1 must block when evaluating its local job allocation (e.g., single or multiprocessor direct interface evaluations). It performs `serve_evaluations_peerr()` for any other processors within the first evaluation partition and `serve_evaluations_{synch,asynch}()` for all other evaluation partitions (depending on `asynchLocalEvalConcurrency`). Dynamic scheduling assigns for its portion of the job allocation using either `asynchronous_local_evaluations()` or `synchronous_local_evaluations()`. Single-level and multilevel parallel use intra- and inter-communicators, respectively, for send/receive. Specific syntax is encapsulated within `ParallelLibrary`. The iteratorCommRank 0 processor assigns the static schedule since it is the only processor with access to beforeSynchCorePRPQueue (it runs the iterator and calls synchronize). The alternate design of each peer selecting its own jobs using the modulus operator would be applicable if execution of this function (and therefore the job list) were distributed.

References ApplicationInterface::asynchLocalEvalConcurrency, ApplicationInterface::asynchronous_local_evaluations(), ApplicationInterface::beforeSynchCorePRPQueue, ApplicationInterface::numEvalServers, Interface::outputLevel, ApplicationInterface::parallelLib, ApplicationInterface::recvEvaluation(), ApplicationInterface::recvBuffers, ApplicationInterface::recvRequests, ApplicationInterface::send_evaluation(), ApplicationInterface::sendBuffers, ApplicationInterface::synchronous_local_evaluations(), and ParallelLibrary::waitall().

Referenced by ApplicationInterface::synch().

`void peer_dynamic_schedule_evaluations( ) [private]`
blocking dynamic schedule of all evaluations in beforeSynchCorePRPQueue using message passing on a peer partition; executes on iteratorComm master
This code runs on the iteratorCommRank 0 processor (the iterator) and is called from synch() in order to manage a dynamic schedule, as enabled by nonblocking management of local asynchronous jobs. It matches serve_evaluations_{synch,asynch}() for other evaluation partitions, depending on asynchLocalEvalConcurrency; it does not match serve_evaluations_peer() since, for local asynchronous jobs, the first evaluation partition cannot be multiprocessor. It performs function evaluations locally for its portion of the job allocation using asynchronous_local_evaluations_nowait(). Single-level and multilevel parallel use intra- and inter-communicators, respectively, for send/receive. Specific syntax is encapsulated within ParallelLibrary.

References ApplicationInterface::assign_asynch_local_queue(), ApplicationInterface::asynchLocalEvalConcurrency, ApplicationInterface::beforeSynchCorePRPQueue, ApplicationInterface::msgPassRunningMap, ApplicationInterface::numEvalServers, Interface::outputLevel, ApplicationInterface::recvBuffers, ApplicationInterface::recvRequests, ApplicationInterface::send_evaluation(), ApplicationInterface::sendBuffers, ApplicationInterface::test_local_backfill(), and ApplicationInterface::test_runs_backfill().

Referenced by ApplicationInterface::synch().

void asynchronous_local_evaluations ( PRPQueue & local_prp_queue ) [private]

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

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

References ApplicationInterface::assign_asynch_local_queue(), ApplicationInterface::asynchLocalActivePRPQueue, ApplicationInterface::asynchLocalEvalConcurrency, ApplicationInterface::asynchLocalEvalStatic, ApplicationInterface::completionSet, ApplicationInterface::launch_asynch_local(), ApplicationInterface::localServerAssigned, Dakota::lookup_by_eval_id(), ApplicationInterface::numEvalServers, Interface::outputLevel, ApplicationInterface::process_asynch_local(), Interface::rawResponseMap, and ApplicationInterface::wait_local_evaluations().

Referenced by ApplicationInterface::peer_static_schedule_evaluations(), and ApplicationInterface::synch().

void synchronous_local_evaluations ( PRPQueue & local_prp_queue ) [private]

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

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

References ApplicationInterface::broadcast_evaluation(), Interface::currEvalId, ApplicationInterface::derived_map(), ApplicationInterface::manage_failure(), Interface::multiProcEvalFlag, and ApplicationInterface::process_synch_local().

Referenced by ApplicationInterface::peer_static_schedule_evaluations().

void master_dynamic_schedule_evaluations_nowait ( ) [private]

execute a nonblocking dynamic schedule in a master-slave partition

This code is called from synch_nowait() to provide the master portion of a nonblocking master-slave algorithm for the dynamic scheduling of evaluations among slave servers. It performs no evaluations locally and matches either serve_evaluations_synch() or serve_evaluations_asynch() on the slave servers, depending on the value of asynchLocalEvalConcurrency. Dynamic scheduling assigns jobs in 2 passes. The 1st pass gives each server the same number of jobs (equal to asynchLocalEvalConcurrency). The 2nd pass assigns the remaining jobs to
slave servers as previous jobs are completed. Single- and multilevel parallel use intra- and inter-communicators, respectively, for send/receive. Specific syntax is encapsulated within ParallelLibrary.

References Dakota::abort_handler(), ApplicationInterface::asynchLocalEvalConcurrency, ApplicationInterface::beforeSynchCorePRPQueue, ApplicationInterface::msgPassRunningMap, ApplicationInterface::numEvalServers, ApplicationInterface::recvBuffers, ApplicationInterface::recvRequests, ApplicationInterface::send_evaluation(), ApplicationInterface::sendBuffers, and ApplicationInterface::test_receives_backfill().

Referenced by ApplicationInterface::synch_nowait().

```cpp
void peer_dynamic_schedule_evaluations_nowait() [private]
```
execute a nonblocking static/dynamic schedule in a peer partition

This code runs on the iteratorCommRank 0 processor (the iterator) and is called from synch_nowait() in order to manage a nonblocking static schedule. It matches serve_evaluations_{synch,asynch}() for other evaluation partitions (depending on asynchLocalEvalConcurrency). It performs nonblocking local function evaluations for its portion of the static schedule using asynchronous_local_evaluations(). Single-level and multilevel parallel use intra- and inter-communicators, respectively, for send/receive. Specific syntax is encapsulated within ParallelLibrary. The iteratorCommRank 0 processor assigns the static schedule since it is the only processor with access to beforeSynchCorePRPQueue (it runs the iterator and calls synchronize). The alternate design of each peer selecting its own jobs using the modulus operator would be applicable if execution of this function (and therefore the job list) were distributed.

References Dakota::abort_handler(), ApplicationInterface::assign_asynch_local_queue(), ApplicationInterface::assign_asynch_local_queue_nowait(), ApplicationInterface::asynchLocalEvalConcurrency, ApplicationInterface::beforeSynchCorePRPQueue, Dakota::lookup_by_eval_id(), ApplicationInterface::msgPassRunningMap, ApplicationInterface::numEvalServers, ApplicationInterface::recvBuffers, ApplicationInterface::recvRequests, ApplicationInterface::send_evaluation(), ApplicationInterface::sendBuffers, ApplicationInterface::test_local_backfill(), and ApplicationInterface::test_receives_backfill().

Referenced by ApplicationInterface::synch_nowait().

```cpp
void asynchronous_local_evaluations_nowait (PRPQueue & local_prp_queue) [private]
```
launch new jobs in prp_queue asynchronously (if capacity is available), perform nonblocking query of all running jobs, and process any completed jobs (handles both local master- and local peer-scheduling cases)

This function provides nonblocking synchronization for the local asynch case (background system call, 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 test_local_evaluations() to capture completed jobs in nonblocking mode. It mirrors a nonblocking message passing scheduler as much as possible (test_local_evaluations() modeled after MPI_Testsome()). The result of this function is raw-ResponseMap, which uses eval_id as a key. It is assumed that the incoming local_prp_queue contains only active and new jobs - i.e., all completed jobs are cleared by synch_nowait().

Also supports asynchronous local evaluations with static scheduling. This scheduling policy specifically ensures that a completed asynchronous evaluation eval_id is replaced with an equivalent one, modulo asynchLocalEvalConcurrency. In the nowait case, this could render some servers idle if evaluations don’t come in eval_id order or some evaluations are cancelled by the caller in between calls. If this function is called with unlimited local eval concurrency, the static scheduling request is ignored.

References ApplicationInterface::assign_asynch_local_queue_nowait(), ApplicationInterface::asynchLocalActivePRPQueue, ApplicationInterface::asynchLocalEvalConcurrency, ApplicationInterface::asynchLocalEvalStatic, and ApplicationInterface::test_local_backfill().

Referenced by ApplicationInterface::synch_nowait().
void serve_evaluations_synch ( ) [private]

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

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

References ParallelLibrary::bcast_e(), Interface::currEvalId, ApplicationInterface::derived_map(), ApplicationInterface::evalCommRank, ParallelLibrary::send_ie(), ApplicationInterface::lenResponseMessage, ApplicationInterface::lenVarsActSetMessage, ApplicationInterface::manage_failure(), Interface::multiProcEvalFlag, ApplicationInterface::parallelLib, ParallelLibrary::recv_ie(), MPIPackBuffer::reset(), and ParallelLibrary::wait().

Referenced by ApplicationInterface::serve_evaluations().

void serve_evaluations_synch_peer ( ) [private]

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

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

References ParallelLibrary::bcast_e(), Interface::currEvalId, ApplicationInterface::derived_map(), ApplicationInterface::lenVarsActSetMessage, ApplicationInterface::manage_failure(), and ApplicationInterface::parallelLib.

Referenced by ApplicationInterface::serve_evaluations().

void serve_evaluations_asynch ( ) [private]

serve the evaluation message passing schedulers and manage multiple asynchronous evaluations.

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

References Dakota::abort_handler(), ApplicationInterface::asynchLocalActivePRPQueue, ApplicationInterface::asynchLocalEvalConcurrency, ParallelLibrary::bcast_e(), ApplicationInterface::completionSet, ApplicationInterface::derived_map_asynch(), ApplicationInterface::evalCommRank, Interface::interfaceId, ParallelLibrary::recv_ie(), ApplicationInterface::lenResponseMessage, ApplicationInterface::lenVarsActSetMessage, Dakota::lookup_eval_id(), Interface::multiProcEvalFlag, ApplicationInterface::parallelLib, ParallelLibrary::recv-ie(), MPIUnpackBuffer::reset(), ParallelLibrary::send-ie(), ParallelLibrary::test(), and ApplicationInterface::test_local_evaluations().

Referenced by ApplicationInterface::serve_evaluations().

void serve_evaluations_asynch_peer ( ) [private]

serve the evaluation message passing schedulers and perform multiple asynchronous evaluations as part of the 1st peer.

This code is invoked by serve_evaluations() to perform multiple asynchronous jobs on multiprocessor slave/peer servers. It matches the multiProcEvalFlag bcasts in ApplicationInterface::asynchronous_local_evaluations().

References Dakota::abort_handler(), ApplicationInterface::asynchLocalActivePRPQueue, ApplicationInterface::asynchLocalEvalConcurrency, ParallelLibrary::bcast_e(), ApplicationInterface::completionSet, ApplicationInterface::derived_map_asynch(), Interface::interfaceId, ApplicationInterface::lenVarsActSetMessage, Dakota::lookup_eval_id(), ApplicationInterface::parallelLib, MPIUnpackBuffer::reset(), and ApplicationInterface::test_local_evaluations().

Referenced by ApplicationInterface::serve_evaluations().
The documentation for this class was generated from the following files:

- ApplicationInterface.hpp
- ApplicationInterface.cpp

## 13.4 Approximation Class Reference

Base class for the approximation class hierarchy.

Inheritance diagram for Approximation:

```
Approximation
   \_ GaussProcApproximation
   \_ PecosApproximation
   \_ SurfpackApproximation
   \_ TANA3Approximation
   \_ TaylorApproximation
```

### Public Member Functions

- **Approximation ()**<br>
  default constructor

- **Approximation (ProblemDescDB &problem_db, const SharedApproxData &shared_data)**<br>
  standard constructor for envelope

- **Approximation (const SharedApproxData &shared_data)**<br>
  alternate constructor

- **Approximation (const Approximation &approx)**<br>
  copy constructor

- **virtual ~Approximation ()**<br>
  destructor

- **Approximation operator= (const Approximation &approx)**<br>
  assignment operator

- **virtual void build ()**<br>
  builds the approximation from scratch

- **virtual void rebuild ()**<br>
  rebuilds the approximation incrementally

- **virtual void pop (bool save_data)**<br>
  removes entries from end of SurrogateData::{vars,resp}Data (last points appended, or as specified in args)

- **virtual void restore ()**<br>
  restores state prior to previous append()

- **virtual void finalize ()**<br>
  finalize approximation by applying all remaining trial sets

- **virtual void store ()**<br>
  store current approximation for later combination

- **virtual void combine (short corr_type)**<br>
  combine current approximation with previously stored approximation

- **virtual Real value (const Variables &vars)**<br>
  retrieve the approximate function value for a given parameter vector
13.4. APPROXIMATION CLASS REFERENCE

- virtual const RealVector & gradient (const Variables &vars)
  retrieve the approximate function gradient for a given parameter vector
- virtual const RealSymMatrix & hessian (const Variables &vars)
  retrieve the approximate function Hessian for a given parameter vector
- virtual Real prediction_variance (const Variables &vars)
  retrieve the variance of the predicted value for a given parameter vector
- virtual bool diagnostics_available ()
  check if diagnostics are available for this approximation type
- virtual Real diagnostic (const String &metric_type)
  retrieve a single diagnostic metric for the diagnostic type specified
- virtual void primary_diagnostics (int fn_index)
  compute and print all requested diagnostics and cross-validation
- virtual void challenge_diagnostics (const RealMatrix &challenge_points, int fn_index)
  compute and print all requested diagnostics for user provided challenge pts
- virtual const RealVector & approximation_coefficients () const
  return the coefficient array computed by build/rebuild()
- virtual void approximation_coefficients (const RealVector &approx_coeffs)
  set the coefficient array from external sources, rather than computing with build/rebuild()
- virtual void coefficient_labels (std::vector<std::string> &coeff_labels) const
  print the coefficient array computed in build/rebuild()
- virtual void print_coefficients (std::ostream &s, bool normalized=false)
  print the coefficient array computed in build/rebuild()
- virtual int min_coefficients () const
  return the minimum number of samples (unknowns) required to build the derived class approximation type in numVars dimensions
- virtual int recommended_coefficients () const
  return the recommended number of samples (unknowns) required to build the derived class approximation type in numVars dimensions
- virtual int num_constraints () const
  return the number of constraints to be enforced via an anchor point
- virtual void clear_current ()
  clear current build data in preparation for next build
- int min_points (bool constraint_flag) const
  return the minimum number of points required to build the approximation type in numVars dimensions. Uses _coefficients() and num_constraints().
- int recommended_points (bool constraint_flag) const
  return the recommended number of samples to build the approximation type in numVars dimensions (default same as min_points)
- const Pecos::SurrogateData & approximation_data () const
  return approxData
- void add (const Pecos::SurrogateDataVars &svd, bool anchor_flag)
  append to SurrogateData::varsData or assign to SurrogateData::anchorVars
- void add (const Variables &vars, bool anchor_flag, bool deep_copy)
  extract the relevant vectors from Variables and invoke add(RealVector&, IntVector&, RealVector&)
• void add (const Real ∗c_vars, bool anchor_flag, bool deep_copy)
  create a RealVector view and invoke add(RealVector&, empty, empty)
• void add (const RealVector &c_vars, const IntVector &di_vars, const RealVector &dr_vars, bool anchor_flag, bool deep_copy)
  shared code among add(Variables&) and add(Real*): adds a new data point by either appending to SurrogateData::varsData or assigning to SurrogateData::anchorVars, as dictated by anchor_flag. Uses add_point() and add_anchor().
• void add (const Pecos::SurrogateDataResp &sdr, bool anchor_flag)
  append to SurrogateData::respData or assign to SurrogateData::anchorResp
• void add (const Response &response, int fn_index, bool anchor_flag, bool deep_copy)
  adds a new data point by either appending to SurrogateData::respData or assigning to SurrogateData::anchorResp, as dictated by anchor_flag. Uses add_point() and add_anchor().
• void pop_count (size_t count)
  appends to popCountStack (number of entries to pop from end of SurrogateData::{vars,resp}Data, based on size of last data set appended)
• void clear_all ()
  clear all build data (current and history) to restore original state
• void clear_anchor ()
  clear SurrogateData::anchor{Vars,Resp}
• void clear_data ()
  clear SurrogateData::{vars,resp}Data
• void clear_saved ()
  clear popCountStack and SurrogateData::saved{Vars,Resp}Trials
• void set_bounds (const RealVector &c_l_bnds, const RealVector &c_u_bnds, const IntVector &di_l_bnds, const IntVector &di_u_bnds, const RealVector &dr_l_bnds, const RealVector &dr_u_bnds)
  set approximation lower and upper bounds (currently only used by graphics)
• Approximation ∗ approx_rep () const
  returns approxRep for access to derived class member functions that are not mapped to the top Approximation level

Protected Member Functions

• Approximation (BaseConstructor, const ProblemDescDB &problem_db, const SharedApproxData &shared_data)
  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)
• Approximation (NoDBBaseConstructor, const SharedApproxData &shared_data)
  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

Protected Attributes

• RealVector approxGradient
  gradient of the approximation returned by gradient()
• RealSymMatrix approxHessian
  Hessian of the approximation returned by hessian()
• Pecos::SurrogateData approxData
contains the variables/response data for constructing a single approximation model (one response function)

- SharedApproxData * sharedDataRep
  contains the approximation data that is shared among the response set

Private Member Functions

- Approximation * get_approx (ProblemDescDB &problem_db, const SharedApproxData &shared_data)
  Used only by the standard envelope constructor to initialize approxRep to the appropriate derived type.

- Approximation * get_approx (const SharedApproxData &shared_data)
  Used only by the alternate envelope constructor to initialize approxRep to the appropriate derived type.

Private Attributes

- SizetArray popCountStack
  a stack managing the number of points previously added by calls to append() that can be removed by calls to pop()

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

- int referenceCount
  number of objects sharing approxRep

13.4.1 Detailed Description

Base class for the approximation class hierarchy.

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

13.4.2 Constructor & Destructor Documentation

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.

Approximation ( ProblemDescDB & problem_db, const SharedApproxData & shared_data )

standard constructor for envelope

Envelope constructor only needs to extract enough data to properly execute get_approx, since Approximation(-BaseConstructor, problem_db) builds the actual base class data for the derived approximations.

References Dakota::abort_handler(), Approximation::approxRep, and Approximation::get_approx().
Approximation (const SharedApproxData & shared_data)

alternate constructor
This is the alternate envelope constructor for instantiations on the fly. Since it does not have access to problem_db, it utilizes the NoDBBaseConstructor constructor chain.
References Dakota::abort_handler(), Approximation::approxRep, and Approximation::get_approx().

Approximation (const Approximation & approx)
copy constructor
Copy constructor manages sharing of approxRep and incrementing of referenceCount.
References Approximation::approxRep, and Approximation::referenceCount.

~Approximation() [virtual]
destructor
Destructor decrements referenceCount and only deletes approxRep when referenceCount reaches zero.
References Approximation::approxRep, and Approximation::referenceCount.

Approximation (BaseConstructor, const ProblemDescDB & problem_db, const SharedApproxData & shared_data) [protected]
constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)
This constructor is the one which must build the base class data for all derived classes. get_approx() instantiates a derived class letter and the derived constructor selects this base class constructor in its initialization list (to avoid recursion in the base class constructor calling get_approx() again). Since the letter IS the representation, its rep pointer is set to NULL (an uninitialized pointer causes problems in ~Approximation).

Approximation (NoDBBaseConstructor, const SharedApproxData & shared_data) [protected]
constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)
This constructor is the one which must build the base class data for all derived classes. get_approx() instantiates a derived class letter and the derived constructor selects this base class constructor in its initialization list (to avoid recursion in the base class constructor calling get_approx() again). Since the letter IS the representation, its rep pointer is set to NULL (an uninitialized pointer causes problems in ~Approximation).

13.4.3 Member Function Documentation

Approximation operator=(const Approximation & approx)
assignment operator
References Approximation::approxRep, and Approximation::referenceCount.
13.4. APPROXIMATION CLASS REFERENCE

**void build( ) [virtual]**

builds the approximation from scratch

This is the common base class portion of the virtual fn and is insufficient on its own; derived implementations should explicitly invoke (or reimplement) this base class contribution.

Reimplemented in PecosApproximation, GaussProcApproximation, SurfpackApproximation, TaylorApproximation, and TANA3Approximation.

References Dakota::abort_handler(), Approximation::approxData, Approximation::approxRep, Approximation::build(), Approximation::min_points(), SharedApproxData::numVars, and Approximation::sharedDataRep.

Referenced by TANA3Approximation::build(), TaylorApproximation::build(), Approximation::build(), SurfpackApproximation::build(), GaussProcApproximation::build(), PecosApproximation::build(), and Approximation::rebuild().

**void rebuild( ) [virtual]**

rebuilds the approximation incrementally

This is the common base class portion of the virtual fn and is insufficient on its own; derived implementations should explicitly invoke (or reimplement) this base class contribution.

Reimplemented in PecosApproximation.

References Approximation::approxRep, Approximation::build(), and Approximation::rebuild().

Referenced by Approximation::rebuild().

**void pop( bool save_data ) [virtual]**

removes entries from end of SurrogateData::{vars,resp}Data (last points appended, or as specified in args)

This is the common base class portion of the virtual fn and is insufficient on its own; derived implementations should explicitly invoke (or reimplement) this base class contribution.

Reimplemented in PecosApproximation.

References Dakota::abort_handler(), Approximation::approxData, Approximation::approxRep, Approximation::pop(), and Approximation::popCountStack.

Referenced by Approximation::pop(), and PecosApproximation::pop().

**void restore( ) [virtual]**

restores state prior to previous append()

This is the common base class portion of the virtual fn and is insufficient on its own; derived implementations should explicitly invoke (or reimplement) this base class contribution.

Reimplemented in PecosApproximation.

References Approximation::approxData, Approximation::approxRep, Approximation::popCountStack, SharedApproxData::restoration_index(), Approximation::restore(), and Approximation::sharedDataRep.

Referenced by Approximation::restore(), and PecosApproximation::restore().

**void finalize( ) [virtual]**

finalizes approximation by applying all remaining trial sets

This is the common base class portion of the virtual fn and is insufficient on its own; derived implementations should explicitly invoke (or reimplement) this base class contribution.

Reimplemented in PecosApproximation.

References Approximation::approxData, Approximation::approxRep, Approximation::clear_saved(), SharedApproxData::finalization_index(), Approximation::finalize(), and Approximation::sharedDataRep.

Referenced by Approximation::finalize(), and PecosApproximation::finalize().
void clear_current( ) [inline], [virtual]
clear current build data in preparation for next build
  Redefined by TANA3Approximation to clear current data but preserve history.
  Reimplemented in TANA3Approximation.
  References Approximation::approxRep, Approximation::clear_all(), and Approximation::clear_current().
  Referenced by Approximation::clear_current().

void clear_all( ) [inline]
clear all build data (current and history) to restore original state
  Clears out any history (e.g., TANA3Approximation use for a different response function in NonDReliability).
  References Approximation::approxData, Approximation::approxRep, and Approximation::clear_all().
  Referenced by Approximation::clear_all(), and Approximation::clear_current().

Approximation * get_approx ( ProblemDescDB &problem_db, const SharedApproxData &shared_data ) [private]
Used only by the standard envelope constructor to initialize approxRep to the appropriate derived type.
  Used only by the envelope constructor to initialize approxRep to the appropriate derived type.
  References SharedApproxData::approxType, SharedApproxData::data_rep(), and Dakota::strends().
  Referenced by Approximation::Approximation().

Approximation * get_approx ( const SharedApproxData &shared_data ) [private]
Used only by the alternate envelope constructor to initialize approxRep to the appropriate derived type.
  Used only by the envelope constructor to initialize approxRep to the appropriate derived type.
  References SharedApproxData::approxType, SharedApproxData::data_rep(), and Dakota::strends().
  The documentation for this class was generated from the following files:
  • DakotaApproximation.hpp
  • DakotaApproximation.cpp

13.5 ApproximationInterface Class Reference
Derived class within the interface class hierarchy for supporting approximations to simulation-based results.
  Inheritance diagram for ApproximationInterface:

  Interface
  |                |
  | ApproximationInterface |

Public Member Functions
  • ApproximationInterface (ProblemDescDB &problem_db, const Variables &am_vars, bool am_cache, const String &am_interface_id, size_t num_fns)
    primary constructor
ApproximationInterface (const String &approx_type, const UShortArray &approx_order, const Variables &am_vars, bool am_cache, const String &am_interface_id, size_t num_fns, short data_order, short output_level)

Alternate constructor for instantiations on the fly

~ApproximationInterface ()
Destructor

Protected Member Functions

- void map (const Variables &vars, const ActiveSet &set, Response &response, bool async_flag=false)
  Function evaluator: provides an approximate mapping from the variables to the responses using function-surfaces

- int minimum_points (bool constraint_flag) const
  Returns the minimum number of samples required to build the function-surfaces

- int recommended_points (bool constraint_flag) const
  Returns the recommended number of samples recommended to build the function-surfaces

- void approximation_function_indices (const IntSet &approx_fn_indices)
  Sets the (currently active) approximation function index set

- void update_approximation (const Variables &vars, const IntResponsePair &response_pr)

- void update_approximation (const RealMatrix &samples, const IntResponseMap &resp_map)

- void update_approximation (const VariablesArray &vars_array, const IntResponseMap &resp_map)

- void append_approximation (const Variables &vars, const IntResponsePair &response_pr)

- void append_approximation (const RealMatrix &samples, const IntResponseMap &resp_map)

- void append_approximation (const VariablesArray &vars_array, const IntResponseMap &resp_map)

- void build_approximation (const RealVector &cl_bnds, const RealVector &cu_bnds, const IntVector &dl_bnds, const IntVector &du_bnds, const RealVector &dr_l_bnds, const RealVector &dr_u_bnds)

- void rebuild_approximation (const BoolDeque &rebuild_deque)

- void pop_approximation (bool save_surr_data)

- void restore_approximation ()

- bool restore_available ()
  Queries the approximation for the ability to restore a previous increment

- void finalize_approximation ()
  Finalizes the approximation by applying all trial increments

- void store_approximation ()
  Move the current approximation into storage for later increments

- void combine_approximation (short corr_type)
  Combine the current approximation with one previously stored

- void clear_current ()
  Clears current data from an approximation interface

- void clear_all ()
  Clears all data from an approximation interface

- void clear_saved ()
  Clears saved data (from pop invocations) from an approximation interface

- SharedApproxData & shared_approximation ()
  Retrieve the SharedApproxData within an ApproximationInterface
• std::vector< Approximation > & approximations ()
  retrieve the Approximations within an ApproximationInterface
• const Pecos::SurrogateData & approximation_data (size_t index)
  retrieve the approximation data from a particular Approximation within an ApproximationInterface
• const RealVectorArray & approximation_coefficients ()
  retrieve the approximation coefficients from each Approximation within an ApproximationInterface
• void approximation_coefficients (const RealVectorArray &approx_coeffs)
  set the approximation coefficients within each Approximation within an ApproximationInterface
• const RealVector & approximation_variances (const Variables &vars)
  retrieve the approximation variances from each Approximation within an ApproximationInterface
• const IntResponseMap & synch ()
  recovers data from a series of asynchronous evaluations (blocking)
• const IntResponseMap & synch_nowait ()
  recovers data from a series of asynchronous evaluations (nonblocking)

Private Member Functions

• void mixed_add (const Variables &vars, const Response &response, bool anchor)
  add variables/response data to functionSurfaces using a mixture of shallow and deep copies
• void mixed_add (const Real ∗c_vars, const Response &response, bool anchor)
  add variables/response data to functionSurfaces using a mixture of shallow and deep copies
• void shallow_add (const Variables &vars, const Response &response, bool anchor)
  add variables/response data to functionSurfaces using a shallow copy
• void sample_to_variables (const Real ∗sample_c_vars, size_t num_cv, Variables &vars)
  populate continuous variables within vars from sample_c_vars
• void update_pop_counts (const IntResponseMap &resp_map)
  append to the popCountStack within each of the functionSurfaces based on the active set definitions within resp_map
• void read_challenge_points ()
  Load approximation test points from user challenge points file.

Private Attributes

• IntSet approxFnIndices
  for incomplete approximation sets, this array specifies the response function subset that is approximated
• SharedApproxData sharedData
  data that is shared among all functionSurfaces
• std::vector< Approximation > functionSurfaces
  list of approximations, one per response function
• RealVectorArray functionSurfaceCoeffs
  array of approximation coefficient vectors, one vector per response function
• RealVector functionSurfaceVariances
  vector of approximation variances, one value per response function
• String challengeFile
  data file for user-supplied challenge data (per interface, since may contain multiple responses)
13.5. APPROXIMATIONINTERFACE CLASS REFERENCE

- bool challengeAnnotated
  whether the points file is annotated
- RealMatrix challengePoints
  container for the challenge points data
- Variables actualModelVars
  copy of the actualModel variables object used to simplify conversion among differing variable views
- bool actualModelCache
  indicates usage of an evaluation cache by the actualModel
- String actualModelInterfaceId
  the interface id from the actualModel used for ordered PRPCache lookups
- IntResponseMap beforeSynchResponseMap
  bookkeeping map to catalogue responses generated in map() for use in synch() and synch_nowait(). This supports pseudo-asynchronous operations (approximate responses are always computed synchronously, but asynchronous virtual functions are supported through bookkeeping).

Additional Inherited Members

13.5.1 Detailed Description

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

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

13.5.2 Member Function Documentation

void update_approximation ( const Variables & vars, const IntResponsePair & response_pr )
[protected], [virtual]
This function populates/replaces each Approximation::anchorPoint with the incoming variables/response data point.
  Reimplemented from Interface.
  References ApproximationInterface::actualModelCache, ApproximationInterface::actualModelInterfaceId, Dakota::data_pairs, Dakota::lookup_by_ids(), ApproximationInterface::mixed_add(), and ApproximationInterface::shallow_add().

void update_approximation ( const RealMatrix & samples, const IntResponseMap & resp_map )
[protected], [virtual]
This function populates/replaces each Approximation::currentPoints with the incoming variables/response arrays.
  Reimplemented from Interface.
  References Dakota::abort_handler(), ApproximationInterface::actualModelCache, ApproximationInterface::actualModelInterfaceId, ApproximationInterface::actualModelVars, ApproximationInterface::approxFnIndices, Dakota::data_pairs, ApproximationInterface::functionSurfaces, Dakota::lookup_by_ids(), ApproximationInterface::mixed_add(), ApproximationInterface::sample_to_variables(), and ApproximationInterface::shallow_add().
void update_approximation ( const VariablesArray & vars_array, const IntResponseMap & resp_map )
[protected], [virtual]
This function populates/replaces each Approximation::currentPoints with the incoming variables/response arrays.
  Reimplemented from Interface.
  References Dakota::abort_handler(), ApproximationInterface::actualModelCache, ApproximationInterface::actualModelInterfaceId, ApproximationInterface::approxFnIndices, Dakota::data_pairs, ApproximationInterface::functionSurfaces, Dakota::lookup_by_ids(), ApproximationInterface::mixed_add(), and ApproximationInterface::shallow_add().

void append_approximation ( const Variables & vars, const IntResponsePair & response_pr )
[protected], [virtual]
This function appends to each Approximation::currentPoints with one incoming variables/response data point.
  Reimplemented from Interface.
  References ApproximationInterface::actualModelCache, ApproximationInterface::actualModelInterfaceId, ApproximationInterface::approxFnIndices, Dakota::data_pairs, ApproximationInterface::functionSurfaces, Dakota::lookup_by_ids(), ApproximationInterface::mixed_add(), and ApproximationInterface::shallow_add().

void append_approximation ( const RealMatrix & samples, const IntResponseMap & resp_map )
[protected], [virtual]
This function appends to each Approximation::currentPoints with multiple incoming variables/response data points.
  Reimplemented from Interface.
  References Dakota::abort_handler(), ApproximationInterface::actualModelCache, ApproximationInterface::actualModelInterfaceId, ApproximationInterface::actualModelVars, Dakota::data_pairs, Dakota::lookup_by_ids(), ApproximationInterface::mixed_add(), ApproximationInterface::sample_to_variables(), ApproximationInterface::shallow_add(), and ApproximationInterface::update_pop_counts().

void append_approximation ( const VariablesArray & vars_array, const IntResponseMap & resp_map )
[protected], [virtual]
This function appends to each Approximation::currentPoints with multiple incoming variables/response data points.
  Reimplemented from Interface.
  References Dakota::abort_handler(), ApproximationInterface::actualModelCache, ApproximationInterface::actualModelInterfaceId, Dakota::data_pairs, Dakota::lookup_by_ids(), ApproximationInterface::mixed_add(), ApproximationInterface::shallow_add(), and ApproximationInterface::update_pop_counts().

void build_approximation ( const RealVector & e_l_bnds, const RealVector & e_u_bnds, const IntVector & d_l_bnds, const IntVector & d_u_bnds, const RealVector & dr_l_bnds, const RealVector & dr_u_bnds )
[protected], [virtual]
This function finds the coefficients for each Approximation based on the data passed through update_approximation() calls. The bounds are used only for graphics visualization.
  Reimplemented from Interface.
  References ApproximationInterface::approxFnIndices, SharedApproxData::build(), ApproximationInterface::challengeFile, ApproximationInterface::challengePoints, ApproximationInterface::functionSurfaces, ApproximationInterface::read_challenge_points(), SharedApproxData::set_bounds(), and ApproximationInterface::sharedData.
void rebuild_approximation ( const BoolDeque & rebuild_deque ) [protected], [virtual]
This function updates the coefficients for each Approximation based on data increments provided by \{update, append\}_approximation().
  Reimplemented from Interface.
  References ApproximationInterface::approxFnIndices, ApproximationInterface::functionSurfaces, SharedApproxData::rebuild(), and ApproximationInterface::sharedData.

void pop_approximation ( bool save_surr_data ) [inline], [protected], [virtual]
This function removes data provided by a previous append_approximation() call, possibly different numbers for each function, or as specified in pop_count, which is assumed to be the same for all functions.
  Reimplemented from Interface.
  References ApproximationInterface::approxFnIndices, ApproximationInterface::functionSurfaces, SharedApproxData::pop(), and ApproximationInterface::sharedData.

void restore_approximation ( ) [inline], [protected], [virtual]
This function updates the coefficients for each Approximation based on data increments provided by \{update, append\}_approximation().
  Reimplemented from Interface.
  References ApproximationInterface::approxFnIndices, ApproximationInterface::functionSurfaces, SharedApproxData::post_restore(), SharedApproxData::pre_restore(), and ApproximationInterface::sharedData.

### 13.5.3 Member Data Documentation

```
std::vector<Approximation> functionSurfaces [private]
```

list of approximations, one per response function

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

Referenced by ApproximationInterface::append_approximation(), ApproximationInterface::approximation_coefficients(), ApproximationInterface::approximation_data(), ApproximationInterface::approximation_variances(), ApproximationInterface::build_approximation(), ApproximationInterface::approximations(), ApproximationInterface::build_approximation(), ApproximationInterface::clear_all(), ApproximationInterface::clear_current(), ApproximationInterface::clear_saved(), ApproximationInterface::combine_approximation(), ApproximationInterface::finalize_approximation(), ApproximationInterface::map(), ApproximationInterface::minimum_points(), ApproximationInterface::mixed_add(), ApproximationInterface::pop_approximation(), ApproximationInterface::read_challenge_points(), ApproximationInterface::rebuild_approximation(), ApproximationInterface::recommended_points(), ApproximationInterface::restore_approximation(), ApproximationInterface::shallow_add(), ApproximationInterface::store_approximation(), ApproximationInterface::update_approximation(), and ApproximationInterface::update_pop_counts().

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

- ApproximationInterface.hpp
- ApproximationInterface.cpp

### 13.6 APPSEvalMgr Class Reference

Evaluation manager class for APPSPACK.

Inherits Executor.
Public Member Functions

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

- **∼APPSEvalMgr ()**
  
  *destructor*

- **bool isReadyForWork () const**
  
  tells APPS whether or not there is a processor available to perform a function evaluation

- **bool submit (const int apps_tag, const HOPSPACK::Vector &apps_xtrial, const HOPSPACK::EvalRequestType apps_request)**
  
  performs a function evaluation at APPS-provided x_in

- **int recv (int &apps_tag, HOPSPACK::Vector &apps_f, HOPSPACK::Vector &apps_cEqs, HOPSPACK::Vector &apps_cIneqs, string &apps_msg)**
  
  returns a function value to APPS

- **std::string getEvaluatorType (void) const**
  
  return the type of the Dakota linked evaluator

- **void printDebugInfo (void) const**
  
  empty implementation of debug info needed to complete the interface

- **void printTimingInfo (void) const**
  
  empty implementation of timing info needed to complete the interface

- **void set_asynch_flag (const bool dakotaAsynchFlag)**
  
  publishes whether or not to do asynchronous evaluations

- **void set_blocking_synch (const bool blockingSynchFlag)**
  
  publishes whether or not APPS is operating synchronously

- **void set_total_workers (const int numDakotaWorkers)**
  
  publishes the number of processors available for function evaluations

- **void set_constraint_map (std::vector< int > constraintMapIndices, std::vector< double > constraintMapMultipliers, std::vector< double > constraintMapOffsets)**
  
  publishes constraint transformation

Private Attributes

- **Model & iteratedModel**
  
  reference to the APPSOptimizer's model passed in the constructor

- **bool modelAsynchFlag**
  
  flag for asynchronous function evaluations

- **bool blockingSynch**
  
  flag for APPS synchronous behavior

- **int numWorkersUsed**
  
  number of processors actively performing function evaluations

- **int numWorkersTotal**
  
  total number of processors available for performing function evaluations

- **std::vector< int > constrMapIndices**
  
  map from Dakota constraint number to APPS constraint number
13.6. APPSEVALMGR CLASS REFERENCE

- std::vector<double> constrMapMultipliers
  multipliers for constraint transformations
- std::vector<double> constrMapOffsets
  offsets for constraint transformations
- RealVector xTrial
  trial iterate
- std::map<int, int> tagList
  map of DAKOTA eval id to APPS eval id (for asynchronous evaluations)
- std::map<int, RealVector> functionList
  map of APPS eval id to responses (for synchronous evaluations)
- IntResponseMap dakotaResponseMap
  map of DAKOTA responses returned by synchronize_nowait()

13.6.1 Detailed Description

Evaluation manager class for APPSPACK.

The APPSEvalMgr class is derived from APPSPACK’s Executor class. It implements the methods of that class in such a way that allows DAKOTA to manage the computation of responses instead of APPS. Iterate and response values are passed between Dakota and APPSPACK via this interface.

13.6.2 Constructor & Destructor Documentation

APPSEvalMgr ( Model & model )

constructor

Evaluation manager class for APPSPACK.

The APPSEvalMgr class is derived from APPSPACK’s Executor class. It implements the methods of that class in such a way that allows DAKOTA to manage the computation of responses instead of APPS. Iterate and response values are passed between Dakota and APPSPACK via this interface.

13.6.3 Member Function Documentation

bool isReadyForWork ( ) const

tells APPS whether or not there is a processor available to perform a function evaluation.

Check to see if all processors available for function evaluations are being used. If not, tell APPS that one is available.

References APPSEvalMgr::numWorkersTotal, and APPSEvalMgr::numWorkersUsed.

bool submit ( const int apps_tag, const HOPSPACK::Vector & apps_xtrial, const HOPSPACK::EvalRequestType apps_request )

performs a function evaluation at APPS-provided x_in.

Convert APPSPACK vector of variables to DAKOTA vector of variables and perform function evaluation asynchronously or not as specified in the DAKOTA input deck. If evaluation is asynchronous, map the dakota id to the APPS tag. If evaluation is synchronous, map the responses to the APPS tag.

References Model::asynch_compute_response(), Model::compute_response(), Model::continuous_variables(), Model::current_response(), Model::evaluation_id(), Response::function_values(), APPSEvalMgr::functionList, APPSEvalMgr::iteratedModel, APPSEvalMgr::modelAsynchFlag, APPSEvalMgr::numWorkersTotal, APPSEvalMgr::numWorkersUsed, APPSEvalMgr::tagList, and APPSEvalMgr::xTrial.
int recv ( int & apps_tag, HOPSPACK::Vector & apps_f, HOPSPACK::Vector & apps_cEqs, HOPSPACK::Vector & apps_cIneqs, string & apps_msg )

returns a function value to APPS

Retrieve a set of 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.

References APPSEvalMgr::blockingSynch, APPSEvalMgr::constrMapIndices, APPSEvalMgr::constrMapMultipliers, APPSEvalMgr::constrMapOffsets, APPSEvalMgr::dakotaResponseMap, APPSEvalMgr::functionList, APPSEvalMgr::iteratedModel, APPSEvalMgr::modelAsynchFlag, Model::num_nonlinear_eq_constraints(), APPSEvalMgr::numWorkersUsed, Model::primary_response_fn_sense(), Model::synchronize(), Model::synchronize_nowait(), and APPSEvalMgr::tagList.

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

- APPSEvalMgr.hpp
- APPSEvalMgr.cpp

### 13.7 APPSOptimizer Class Reference

Wrapper class for HOPSPACK.

Inheritance diagram for APPSOptimizer:

```
  APPSOptimizer
     |        |
     v        v
Minimizer  Optimizer
     |        |
     v        v
  Iterator
```

#### Public Member Functions

- **APPSOptimizer (ProblemDescDB &problem_db, Model &model)**
  constructor
- **APPSOptimizer (Model &model)**
  alternate constructor for on-the-fly instantiation without ProblemDescDB
- **~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
13.7. APPSOPTIMIZER CLASS REFERENCE

Protected Attributes

- HOPSPACK::ParameterList params
  Pointer to APPS parameter list.
- HOPSPACK::ParameterList * problemParams
  Pointer to APPS problem parameter sublist.
- HOPSPACK::ParameterList * linearParams
  Pointer to APPS linear constraint parameter sublist.
- HOPSPACK::ParameterList * mediatorParams
  Pointer to APPS mediator parameter sublist.
- HOPSPACK::ParameterList * citizenParams
  Pointer to APPS citizen/algorithm parameter sublist.
- APPSEvalMgr * evalMgr
  Pointer to the APPS evaluation manager object.
- std::vector<int> constraintMapIndices
  map from Dakota constraint number to APPS constraint number
- std::vector<double> constraintMapMultipliers
  multipliers for constraint transformations
- std::vector<double> constraintMapOffsets
  offsets for constraint transformations

Additional Inherited Members

13.7.1 Detailed Description

Wrapper class for HOPSPACK.

The APPSOptimizer class provides a wrapper for HOPSPACK, a Sandia-developed C++ library for generalized pattern search. HOPSPACK defaults to a coordinate pattern search but also allows for augmented search patterns. It can solve problems with bounds, linear constraints, and general nonlinear constraints. APPSOptimizer uses an APPSEvalMgr object to manage the function evaluations.

The user input mappings are as follows: output max function evaluations, constraint tol initial delta, contraction factor, threshold delta, solution target, synchronization, merit function, constraint penalty, and smoothing factor are mapped into HOPS’s "Display", "Maximum Evaluations", "Active Tolerance"/"Nonlinear Active Tolerance", "Initial Step", "Contraction Factor", "Step Tolerance", "Objective Target", "Synchronous Evaluations", "Penalty Function", "Penalty Parameter", and "Penalty Smoothing Value" data attributes. Refer to the HOPS web site (https://software.sandia.gov/trac/hopspack) for additional information on HOPS objects and controls.

13.7.2 Member Function Documentation

void find_optimum ( ) [virtual]

Performs the iterations to determine the optimal solution.

find_optimum redefines the Optimizer virtual function to perform the optimization using HOPS. It first sets up the problem data, then executes minimize() on the HOPS optimizer, and finally catalogues the results.

Implements Optimizer.
void set_apps_parameters ( ) [protected]

sets options for specific methods based on user specifications

Set all of the HOPS algorithmic parameters as specified in the DAKOTA input deck. This is called at construction time.

References APPSOptimizer::citizenParams, Minimizer::constraintTol, APPSOptimizer::evalMgr, ProblemDescDB::get_real(), ProblemDescDB::get_string(), ProblemDescDB::is_null(), APPSOptimizer::linearParams, Iterator::maxEvalConcurrency, Iterator::maxFunctionEvals, APPSOptimizer::mediatorParams, Minimizer::numContinuousVars, Minimizer::numNonlinearConstraints, Iterator::outputLevel, APPSOptimizer::params, Iterator::probDescDB, APPSOptimizer::problemParams, and APPSEvalMgr::set_blocking_synch().

Referenced by APPSOptimizer::APPSOptimizer().

void initialize_variables_and_constraints ( ) [protected]

initializes problem variables and constraints

Set the variables and constraints as specified in the DAKOTA input deck. This is done at run time.

References Minimizer::bigRealBoundSize, APPSOptimizer::constraintMapIndices, APPSOptimizer::constraintMapMultipliers, APPSOptimizer::constraintMapOffsets, Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), Model::continuous_variables(), APPSOptimizer::evalMgr, Iterator::iteratedModel, Model::linear_eq_constraint_coeffs(), Model::linear_eq_constraint_targets(), Model::linear_ineq_constraint_coeffs(), Model::linear_ineq_constraint_lower_bounds(), Model::linear_ineq_constraint_upper_bounds(), APPSOptimizer::linearParams, Model::nonlinear_eq_constraint_targets(), Model::nonlinear_ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Minimizer::numContinuousVars, Minimizer::numLinearEqConstraints, Minimizer::numLinearIneqConstraints, Minimizer::numNonlinearEqConstraints, Minimizer::numNonlinearIneqConstraints, APPSOptimizer::problemParams, and APPSEvalMgr::set_constraint_map().

Referenced by APPSOptimizer::find_optimum().

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

- APPSOptimizer.hpp
- APPSOptimizer.cpp

13.8 BaseConstructor Struct Reference

Dummy struct for overloading letter-envelope constructors.

Public Member Functions

- **BaseConstructor** (int=0)

  C++ structs can have constructors.
13.8.1 Detailed Description

Dummy struct for overloading letter-envelope constructors.

BaseConstructor is used to overload the constructor for the base class portion of letter objects. It avoids infinite recursion (Coplien p.139) in the letter-envelope idiom by preventing the letter from instantiating another envelope. Putting this struct here avoids circular dependencies.

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

- dakota_global_defs.hpp

13.9 callback_data Struct Reference

Public Attributes

- double rosen_cdv_upper_bd
  
  upper bound value to pass through parser to callback function

13.9.1 Detailed Description

Data structure to pass application-specific values through Dakota back to the callback function, for example to convey late updates to bounds, initial points, etc., to Dakota.

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

- library_node.cpp

13.10 COLINApplication Class Reference

Inherits Application< colin::MO_MINLP2::problem >.

Public Member Functions

- COLINApplication ()
  
  Default constructor. Required by COLIN’s ApplicationHandle creation.

- COLINApplication (Model &model)
  
  Constructor with Model (not presently used).

- ~COLINApplication ()
  
  Destructor.

- void set_problem (Model &model)
  
  Helper function called after default construction to extract problem information from the Model and set it for COLIN.

- void set_blocking_synch (const bool blockingSynchFlag)
  
  Publishes whether or not COLIN is operating synchronously

- virtual utilib::Any spawn_evaluation_impl (const utilib::Any &domain, const colin::AppRequest::request_map_t &requests, utilib::seed_t &seed)
  
  Schedule one or more requests at specified domain point, returning a DAKOTA-specific evaluation tracking ID.

- virtual bool evaluation_available ()
  
  Check to see if there are any function values ready to be collected.
• **virtual void perform** evaluation **impl** (const utilib::Any &domain, const colin::AppRequest::request_map_t &requests, utilib::seed_t &seed, colin::AppResponse::response_map_t &colin_responses)

  Perform a function evaluation at t given point.

• **virtual utilib::Any collect** evaluation **impl** (colin::AppResponse::response_map_t &responses, utilib::seed_t &seed)

  Collect a completed evaluation from DAKOTA.

• **virtual void colin** request to dakota **request** (const utilib::Any &domain, const colin::AppRequest::request_map_t &requests, utilib::seed_t &seed)

  Helper function to convert evaluation request data from COLIN structures to DAKOTA structures.

• **virtual void dakota** response to colin **response** (const Response &dakota_response, colin::AppResponse::response_map_t &colin_responses)

  Helper function to convert evaluation response data from DAKOTA structures to COLIN structures.

• **virtual bool map** domain (const utilib::Any &src, utilib::Any &native, bool forward=true) const

  Map the domain point into data type desired by this application context.

### Protected Attributes

• **Model iteratedModel**

  Shallow copy of the model on which COLIN will iterate.

• **bool blockingSynch**

  Flag for COLIN synchronous behavior (Pattern Search only).

• **ActiveSet activeSet**

  Local copy of model’s active set for convenience.

• **std::vector<int> requestedEvals**

  Evaluations queued for asynch evaluation.

• **IntResponseMap dakota responses**

  eval_id to response mapping to cache completed jobs.

#### 13.10.1 Detailed Description

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

#### 13.10.2 Member Function Documentation

**void set_problem ( Model & model )**

Helper function called after default construction to extract problem information from the Model and set it for COLIN.

Set variable bounds and linear and nonlinear constraints. This avoids using probDescDB, so it is called by both the standard and the on-the-fly COLINOptimizer constructors.

References Response::active set(), COLINApplication::activeSet, Model::continuous lower bounds(), Model::continuous upper bounds(), Model::current response(), Model::cv(), Model::discrete int lower bounds(), Model::discrete int sets(), Model::discrete int upper bounds(), Model::discrete set int values(), Model::discrete set real values(), Model::div(), Model::drv(), COLINApplication::iteratedModel, Model::linear eq constraint coeffs(),
Model::linear_eq_constraint_targets(), Model::linear_ineq_constraint_coeffs(), Model::linear_ineq_constraint_lower_bounds(), Model::linear_ineq_constraint_upper_bounds(), Model::nonlinear_eq_constraint_targets(), Model::nonlinear_ineq_constraint_coeffs(), Model::nonlinear_ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Model::num_functions(), Model::num_linear_eq_constraints(), Model::num_linear_ineq_constraints(), Model::num_nonlinear_eq_constraints(), Model::num_nonlinear_ineq_constraints(), and Model::primary_response_fn_sense.

Referenced by COLINApplication::COLINApplication().

utilib::Any spawn_evaluation_impl ( const utilib::Any & domain, const colin::AppRequest::request_map_t & requests, utilib::seed_t & seed ) [virtual]

Schedule one or more requests at specified domain point, returning a DAKOTA-specific evaluation tracking ID.

Schedule one or more requests at specified domain point, returning a DAKOTA-specific evaluation tracking ID. This is only called by COLIN’s concurrent evaluator, which is only instantiated when the Model supports async evals. The domain point is guaranteed to be compatible with data type specified by map_domain(...)

References Model::asynch_compute_response(), COLINApplication::colin_request_to_dakota_request(), Model::evaluation_id(), and COLINApplication::iteratedModel.

bool evaluation_available ( ) [virtual]

Check to see if there are any function values ready to be collected.

Check to see if any asynchronous evaluations have finished. This is only called by COLIN’s concurrent evaluator, which is only instantiated when the Model supports async evals.

References COLINApplication::blockingSynch, COLINApplication::dakota_responses, COLINApplication::iteratedModel, Model::synchronize(), and Model::synchronize_nowait().

void perform_evaluation_impl ( const utilib::Any & domain, const colin::AppRequest::request_map_t & requests, utilib::seed_t & seed, colin::AppResponse::response_map_t & colin_responses ) [virtual]

Perform a function evaluation at t given point.

Perform an evaluation at a specified domain point. Wait for and return the response. This is only called by COLIN’s serial evaluator, which is only instantiated when the Model does not support async evals. The domain point is guaranteed to be compatible with data type specified by map_domain(...)

References COLINApplication::colin_request_to_dakota_request(), Model::compute_response(), Model::current_response(), COLINApplication::dakota_response_to_colin_response(), and COLINApplication::iteratedModel.

utilib::Any collect_evaluation_impl ( colin::AppResponse::response_map_t & colin_responses, utilib::seed_t & seed ) [virtual]

Collect a completed evaluation from DAKOTA.

Collect the next completed evaluation from DAKOTA. Always returns the evalid of the response returned.

References COLINApplication::dakota_response_to_colin_response(), and COLINApplication::dakota_responses.

void colin_request_to_dakota_request ( const utilib::Any & domain, const colin::AppRequest::request_map_t & requests, utilib::seed_t & seed ) [virtual]

Helper function to convert evaluation request data from COLIN structures to DAKOTA structures.

Map COLIN info requests to DAKOTA objectives and constraints.

References Model::continuous_variables(), Model::discrete_int_sets(), Model::discrete_int_variable(), Model::discrete_real_variable(), Model::discrete_set_int_values(), Model::discrete_set_real_values(), Model::div(), Model::drv(), COLINApplication::iteratedModel, Model::num_functions(), and Dakota::set_index_to_value().
Referenced by COLINApplication::perform_evaluation_impl(), and COLINApplication::spawn_evaluation_impl().

```cpp
void dakota_response_to_colin_response ( const Response & dakota_response, colin::AppResponse::response_map_t & colin_responses ) [virtual]
```

Gelper function to convert evaluation response data from DAKOTA structures to COLIN structures.
- Map DAKOTA objective and constraint values to COLIN response.
- References Response::active_set_request_vector(), and Response::function_value().
- Referenced by COLINApplication::collect_evaluation_impl(), and COLINApplication::perform_evaluation_impl().

```cpp
bool map_domain ( const utilib::Any & src, utilib::Any & native, bool forward = true ) const [virtual]
```

Map the domain point into data type desired by this application context.
- Map the domain point into data type desired by this application context (utilib::MixedIntVars). This data type can be exposed from the Any &domain presented to spawn and collect.
- The documentation for this class was generated from the following files:
  - COLINApplication.hpp
  - COLINApplication.cpp

### 13.11 COLINOptimizer Class Reference

Wrapper class for optimizers defined using COLIN.

Inheritance diagram for COLINOptimizer:

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
13.11. COLINOPTIMIZER CLASS REFERENCE

**Destructor**
- void `reset()`
  
  Clears internal optimizer state

- void `find_optimum()`
  
  Iterates the COLIN solver to determine the optimal solution

- bool `returns_multiple_points()` const
  
  Some COLIN methods can return multiple points

**Protected Member Functions**

- void `solver_setup(unsigned short method_name)`
  
  Convenience function for setting up the particular COLIN solver and appropriate Application

- void `set_rng(int seed)`
  
  Sets up the random number generator for stochastic methods

- void `set_solver_parameters()`
  
  Sets construct-time options for specific methods based on user specifications, including calling method-specific set functions

- void `post_run(std::ostream &)`
  
  Get the final set of points from the solver. Look up responses and sort, first according to constraint violation, then according to function value.

- `std::pair<bool, bool> colin_cache_lookup(const colin::AppResponse &colinResponse, Response &tmpResponseHolder)`
  
  Retrieve response from Colin AppResponse, return pair indicating success for <objective, constraints>

- double `constraint_violation(const Response &tmpResponseHolder)`
  
  Compute constraint violation, based on nonlinear constraints in iteratedModel and provided Response data.

**Protected Attributes**

- short `solverType`
  
  COLIN solver sub-type as enumerated in COLINOptimizer.cpp.

- `colin::SolverHandle colinSolver`
  
  Handle to the COLIN solver

- `std::pair<colin::ApplicationHandle, COLINApplication *> colinProblem`
  
  Handle and pointer to the COLINApplication object

- `colin::EvaluationManager_Base * colinEvalMgr`
  
  Pointer to the COLIN evaluation manager object

- `utilib::RNG * rng`
  
  Random number generator pointer

- bool `blockingSynch`
  
  The synchronization setting: true if blocking, false if nonblocking

- Real `constraint_penalty`
  
  Buffer to hold problem constraint penalty parameter.

- bool `constant_penalty`
  
  Buffer to hold problem constant_penalty parameter.
Additional Inherited Members

13.11.1 Detailed Description

Wrapper class for optimizers defined using COLIN.

The COLINOptimizer class wraps COLIN, a Sandia-developed C++ optimization interface library. A variety of COLIN optimizers are defined in COLIN and its associated libraries, including SCOLIB which contains the optimization components from the old COLINY (formerly SGOPT) library. COLIN contains optimizers such as genetic algorithms, pattern search methods, and other nongradient-based techniques. COLINOptimizer uses a COLINApplication object to perform the function evaluations.

The user input mappings are as follows: max_iterations, max_function_evaluations, convergence_tolerance, and solution_accuracy are mapped into COLIN's max_iterations, max_function_evaluations, this_trial, function_value_tolerance, sufficient_objective_value properties. An outputLevel is mapped to COLIN's output_level property and a setting of debug activates output of method initialization and sets the COLIN debug attribute to 10000 for the DEBUG output level. Refer to [Hart, W.E., 2006] for additional information on COLIN objects and controls.

13.11.2 Constructor & Destructor Documentation

COLINOptimizer ( ProblemDescDB & problem_db, Model & model )

standard constructor
Standard constructor.
References ProblemDescDB::get_int(), ProblemDescDB::get_ushort(), Iterator::probDescDB, COLINOptimizer::set_rng(), COLINOptimizer::set_solver_parameters(), and COLINOptimizer::solver_setup().

COLINOptimizer ( const String & method_name, Model & model, int seed, int max_iter, int max_eval )
alternate constructor for on-the-fly instantiations
Alternate constructor for on-the-fly instantiations.
References Iterator::maxFunctionEvals, Iterator::maxIterations, Iterator::method_string_to_enum(), COLINOptimizer::set_rng(), COLINOptimizer::set_solver_parameters(), and COLINOptimizer::solver_setup().

COLINOptimizer ( const String & method_name, Model & model )
alternate constructor for Iterator instantiations by name
Alternate constructor for Iterator instantiations by name.
References Iterator::method_string_to_enum(), COLINOptimizer::set_solver_parameters(), and COLINOptimizer::solver_setup().

13.11.3 Member Function Documentation

void find_optimum ( ) [virtual]
iterates the COLIN solver to determine the optimal solution
find_optimum redefines the Optimizer virtual function to perform the optimization using COLIN. It first sets up the problem data, then executes optimize() on the COLIN solver and finally catalogues the results.
Implements Optimizer.
References Dakota::NPOS, Dakota::abort_handler(), Model::asynch_flag(), COLINOptimizer::blockingSynch, COLINOptimizer::colinEvalMgr, COLINOptimizer::colinProblem, COLINOptimizer::colinSolver, COLINOptimizer::constant_penalty, COLINOptimizer::constraint_penalty, Model::continuous_variables(), Model::discrete_int_sets(),
Model::discrete_int_variables(), Model::discrete_real_variables(), Model::discrete_set_int_values(), Model::discrete_set_real_values(), Model::evaluation_capacity(), Iterator::iteratedModel, Minimizer::numDiscreteIntVars, Minimizer::numDiscreteRealVars, Iterator::outputLevel, Dakota::set_value_to_index(), and COLINOptimizer::solverType.

bool returns_multiple_points() const [virtual]

some COLIN methods can return multiple points
Designate which solvers can return multiple final points.
Reimplemented from Iterator.
References COLINOptimizer::solverType.

void solver_setup(unsigned short method_name) [protected]

convenience function for setting up the particular COLIN solver and appropriate Application
This convenience function is called by the constructors in order to instantiate the solver.
References COLINOptimizer::colinProblem, COLINOptimizer::colinSolver, COLINOptimizer::constant_penalty, COLINOptimizer::constraint_penalty, ProblemDescDB::get_string(), Iterator::method_enum_to_string(), Iterator::probDescDB, and COLINOptimizer::solverType.
Referenced by COLINOptimizer::COLINOptimizer().

void set_rng(int seed) [protected]

sets up the random number generator for stochastic methods
Instantiate random number generator (RNG).
References COLINOptimizer::colinSolver, and COLINOptimizer::rng.
Referenced by COLINOptimizer::COLINOptimizer().

void set_solver_parameters() [protected]

sets construct-time options for specific methods based on user specifications, including calling method-specific set functions
Sets solver properties based on user specifications. Called at construction time.
References Model::asynch_flag(), COLINOptimizer::blockingSynch, COLINOptimizer::colinSolver, COLINOptimizer::constant_penalty, COLINOptimizer::constraint_penalty, Iterator::convergenceTol, ProblemDescDB::get_bool(), ProblemDescDB::get_int(), ProblemDescDB::get_real(), ProblemDescDB::get_sai(), ProblemDescDB::get_string(), ProblemDescDB::is_null(), Iterator::iteratedModel, Iterator::maxEvalConcurrency, Iterator::maxFunctionEvals, Iterator::maxIterations, Minimizer::numContinuousVars, Iterator::outputLevel, Iterator::probDescDB, and COLINOptimizer::solverType.
Referenced by COLINOptimizer::COLINOptimizer().

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

Get the final set of points from the solver Look up responses and sort, first according to constraint violation, then according to function value.
Supplement Optimizer::post_run to first retrieve points from the Colin cache (or possibly the Dakota DB) and rank them. When complete, this function will populate bestVariablesArray and bestResponsesArray with iterator-space data, that is, in the context of the solver, leaving any further untransformation to Optimizer.
Reimplemented from Iterator.
References Iterator::bestResponseArray, Iterator::bestVariablesArray, COLINOptimizer::colin_cache_lookup(), COLINOptimizer::colinProblem, COLINOptimizer::colinSolver, COLINOptimizer::constraintViolation(), Variables::continuous_variables(), Variables::copy(), Response::copy(), Model::current_response(), Model::current_variables(),
Model::discrete_int_sets(), Variables::discrete_int_variable(), Variables::discrete_real_variable(), Model::discrete_set_int_values(), Model::discrete_set_real_values(), Response::function_values(), Iterator::iteratedModel, Optimizer::localObjectiveRecast, Minimizer::numDiscreteIntVars, Minimizer::numDiscreteRealVars, Iterator::numFinalSolutions, Optimizer::numObjectiveFns, Minimizer::objective(), Optimizer::post_run(), Model::primary_response_fn_weights(), Minimizer::resize_bestRespArray(), Minimizer::resize_best_vars_array(), Dakota::set_index_to_value(), and Model::subordinate_model().

std::pair<bool, bool> colin_cache_lookup ( const colin::AppResponse & colinResponse, Response & tmpResponseHolder ) [protected]
Retrieve response from Colin AppResponse, return pair indicating success for <objective, constraints>
Encapsulated Colin Cache response extraction, which will ultimately become the default lookup. Might want to return separate vectors of function values and constraints for use in the sort, but not for now (least change). Return true if not needed or successful lookup.
References Response::function_value(), Minimizer::numNonlinearConstraints, and Optimizer::numObjectiveFns.
Referenced by COLINOptimizer::post_run().

double constraint_violation ( const Response & tmpResponseHolder ) [protected]
Compute constraint violation, based on nonlinear constraints in iteratedModel and provided Response data.
BMA TODO: incorporate constraint tolerance, possibly via elevating SurrBasedMinimizer::constraint_violation(). Always use iteratedModel to get the constraints; they are in the right space.
References Response::function_values(), Iterator::iteratedModel, Model::nonlinear_eq_constraint_targets(), Model::nonlinear_ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Model::num_nonlinear_eq_constraints(), Model::num_nonlinear_ineq_constraints(), and Minimizer::numIterPrimaryFns.
Referenced by COLINOptimizer::post_run().
The documentation for this class was generated from the following files:
- COLINOptimizer.hpp
- COLINOptimizer.cpp

13.12 CollabHybridMetaIterator Class Reference
Meta-iterator for hybrid iteration using multiple collaborating optimization and nonlinear least squares methods.
Inheritance diagram for CollabHybridMetaIterator:

```
Iterator

MetaIterator

CollabHybridMetaIterator
```

Public Member Functions
- CollabHybridMetaIterator (ProblemDescDB &problem_db)
standard constructor

- \texttt{CollabHybridMetaIterator} (\texttt{ProblemDescDB} &\texttt{problem\_db}, \texttt{Model} &\texttt{model})

alternate constructor

- \texttt{\~CollabHybridMetaIterator}()

destructor

Protected Member Functions

- void \texttt{core\_run} ()
  
  Performs the collaborative hybrid iteration.

- const \texttt{Variables} & \texttt{variables\_results} () const
  
  return the final solution from the collaborative iteration (variables)

- const \texttt{Response} & \texttt{response\_results} () const
  
  return the final solution from the collaborative iteration (response)

Private Attributes

- String \texttt{hybridCollabType}
  
  abo or hops

- StringArray \texttt{methodList}
  
  the list of method name identifiers

- bool \texttt{lightwtCtor}
  
  indicates use of lightweight Iterator ctors

- IteratorArray \texttt{selectedIterators}
  
  the set of iterators, one for each entry in \texttt{methodList}

- ModelArray \texttt{selectedModels}
  
  the set of models, one for each iterator

- Variables \texttt{bestVariables}
  
  best variables found in collaborative iteration

- Response \texttt{bestResponse}
  
  best response found in collaborative iteration

Additional Inherited Members

13.12.1 Detailed Description

Meta-iterator for hybrid iteration using multiple collaborating optimization and nonlinear least squares methods.

This meta-iterator has two approaches to hybrid iteration: (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:

- \texttt{CollabHybridMetaIterator.hpp}
- \texttt{CollabHybridMetaIterator.cpp}
13.13 CommandLineHandler Class Reference

Utility class for managing command line inputs to DAKOTA.

Inheritance diagram for CommandLineHandler:

```
CommandLineHandler
   GoLongOpt
```

Public Member Functions

- **CommandLineHandler ()**
  
  Default constructor, requires `check_usage()` call for parsing

- **CommandLineHandler (int argc, char **argv, int world_rank)**
  
  Constructor with parsing

- **~CommandLineHandler ()**
  
  Destructor

- **void check_usage (int argc, char **argv)**
  
  Verifies that DAKOTA is called with the correct command usage. Prints a descriptive message and exits the program if incorrect.

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

- **void usage (std::ostream &outfile=Cout) const**
  
  Print usage information to outfile, conditionally on rank.

Private Member Functions

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

- **void output_helper (const std::string &message, std::ostream &os) const**
  
  Output only on Dakota worldRank 0 if possible

Private Attributes

- **int worldRank**
  
  Rank of this process within Dakota’s allocation; manages conditional output.

Additional Inherited Members

13.13.1 Detailed Description

Utility class for managing command line inputs to DAKOTA.

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

void output_helper ( const std::*string & message, std::*ostream & os ) const [private]

output only on Dakota worldRank 0 if possible
   When there is a valid ParallelLibrary, output only on rank 0
   References CommandLineHandler::worldRank.
   Referenced by CommandLineHandler::check_usage().
   The documentation for this class was generated from the following files:
   • CommandLineHandler.hpp
   • CommandLineHandler.cpp

13.14 CommandShell Class Reference

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

Public Member Functions

- CommandShell (const std::*string &work_dir)
  constructor
- ~CommandShell ()
  destructor
- CommandShell & operator<< (const char ∗cmd)
  appends cmd to sysCommand
- CommandShell & operator<< (const std::*string &cmd)
  convenient operator: appends string to the commandString to be executed
- CommandShell & operator<< (CommandShell & ∗f)(CommandShell &))
  allows passing of the flush function to the shell using <<
- CommandShell & flush ()
  “flushes” the shell; i.e. executes the sysCommand
- void asynch_flag (const bool flag)
  set the asynchFlag
- bool asynch_flag () const
  get the asynchFlag
- void suppress_output_flag (const bool flag)
  set the suppressOutputFlag
- bool suppress_output_flag () const
  get the suppressOutputFlag

Private Attributes

- const std::*string & workDir
  To convey working directory when useWorkdir is true:
- std::*string sysCommand
  The command string that is constructed through one or more << insertions and then executed by flush.
• bool asynchFlag
  flags nonblocking operation (background system calls)
• bool suppressOutputFlag
  flags suppression of shell output (no command echo)

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

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

13.14.2 Member Function Documentation

CommandShell & operator<< ( const char * cmd ) [inline]
appends cmd to sysCommand
  convenient operator: appends string to the commandString to be executed
  References CommandShell::sysCommand.

CommandShell & operator<< ( CommandShell &*(CommandShell &)*f ) [inline]
allows passing of the flush function to the shell using <<
  convenience operator: allows passing of the flush func to the shell via <<

CommandShell & flush ()
"flushes" the shell; i.e. executes the sysCommand
  Executes the sysCommand by passing it to system(). Appends an "&" if asynchFlag is set (background system
call) and echos the sysCommand to Cout if suppressOutputFlag is not set.
  References Dakota::abort_handler(), CommandShell::asynchFlag, CommandShell::suppressOutputFlag, Command-
Shell::sysCommand, and CommandShell::workDir.
  Referenced by Dakota::flush().
  The documentation for this class was generated from the following files:

  • CommandShell.hpp
  • CommandShell.cpp

13.15 ConcurrentMetaIterator Class Reference

Meta-iterator for multi-start iteration or pareto set optimization.

Inheritance diagram for ConcurrentMetaIterator:

```
  Iterator
  ▼
     MetaIterator
     ▼
     ConcurrentMetaIterator
```
13.15. **CONCURRENTMETAINTERATOR CLASS REFERENCE**

**Public Member Functions**

- **ConcurrentMetaIterator** (ProblemDescDB &problem_db)
  
  Standard constructor

- **ConcurrentMetaIterator** (ProblemDescDB &problem_db, Model &model)
  
  Alternate constructor

- ∼**ConcurrentMetaIterator**()
  
  Destructor

**Protected Member Functions**

- **void core_run()**
  
  Performs the concurrent iteration by executing selectedIterator on iteratedModel multiple times in parallel for different settings within the iterator or model.

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

- **void initialize_iterator(int job_index)**
  
  Used by IteratorScheduler to set the starting data for a run

- **void pack_parameters_buffer(MPIPackBuffer &send_buffer, int job_index)**
  
  Used by IteratorScheduler to pack starting data for an iterator run

- **void unpack_parameters_buffer(MPIUnpackBuffer &recv_buffer)**
  
  Used by IteratorScheduler to unpack starting data for an iterator run

- **void pack_results_buffer(MPIPackBuffer &send_buffer, int job_index)**
  
  Used by IteratorScheduler to pack results data from an iterator run

- **void unpack_results_buffer(MPIUnpackBuffer &recv_buffer, int job_index)**
  
  Used by IteratorScheduler to unpack results data from an iterator run

- **void update_local_results(int job_index)**
  
  Used by IteratorScheduler to update local results arrays

- **const Model & algorithm_space_model() const**
  
  Return the result of any recasting or surrogate model recursion layered on top of iteratedModel by the derived Iterator ctor chain

**Private Member Functions**

- **void initialize(int param_set_len)**
  
  Shared constructor code

- **void initialize_iterator(const RealVector &param_set)**
  
  Called by unpack_parameters_buffer(MPIUnpackBuffer) and initialize_iterator(int) to update iteratedModel and selectedIterator

- **void initialize_model(int &param_set_len)**
  
  Initialize the iterated Model prior to Iterator instantiation and define param_set_len
Private Attributes

- **Iterator selectedIterator**
  the iterator selected for concurrent iteration
- **bool lightwtCtor**
  indicates use of lightweight Iterator ctors
- **RealVector initialPt**
  the initial continuous variables for restoring the starting point in the Pareto set minimization
- **RealVectorArray parameterSets**
  an array of parameter set vectors (either multistart variable sets or pareto multi-objective/least squares weighting sets) to be performed.
- **PRPArray prpResults**
  1-d array of ParamResponsePair results corresponding to numIteratorJobs

Additional Inherited Members

13.15.1 Detailed Description

Meta-iterator for multi-start iteration or pareto set optimization.

This meta-iterator 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.

13.15.2 Member Function Documentation

```cpp
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.

References Variables::continuous_variables(), Variables::discrete_int_variables(), Variables::discrete_real_variables(), ParamResponsePair::eval_id(), Iterator::methodName, ConcurrentMetaIterator::parameterSets, ParamResponsePair::prp_parameters(), ParamResponsePair::prp_response(), ConcurrentMetaIterator::prpResults, and Response-::write_tabular().

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

- ConcurrentMetaIterator.hpp
- ConcurrentMetaIterator.cpp

13.16 CONMINOptimizer Class Reference

Wrapper class for the CONMIN optimization library.

Inheritance diagram for CONMINOptimizer:
Public Member Functions

- `CONMINOptimizer (ProblemDescDB &problem_db, Model &model)`
  standard constructor
- `CONMINOptimizer (const String &method_string, Model &model)`
  alternate constructor; construct without `ProblemDescDB`
- `~CONMINOptimizer ()`
  destructor
- `void find_optimum ()`
  Used within the optimizer branch for computing the optimal solution. Redefines the `run` virtual function for the optimizer branch.

Protected Member Functions

- `void initialize_run ()`
  performs run-time set up

Private Member Functions

- `void initialize ()`
  `Shared constructor code.`
- `void allocate_workspace ()`
  `Allocates workspace for the optimizer`.
- `void deallocate_workspace ()`
  `Releases workspace memory`.
- `void allocate_constraints ()`
  `Allocates constraint mappings`.

Private Attributes

- `int conminInfo`
  `INFO from CONMIN manual`.
- `int printControl`
  `IPRINT from CONMIN manual (controls output verbosity)`
- `Real objFnValue`
value of the objective function passed to CONMIN

- RealVector `constraintValues` array of nonlinear constraint values passed to CONMIN
- int `numConminNlnConstr` total number of nonlinear constraints seen by CONMIN
- int `numConminLinConstr` total number of linear constraints seen by CONMIN
- int `numConminConstr` total number of linear and nonlinear constraints seen by CONMIN
- SizetArray `constraintMappingIndices` a container of indices for referencing the corresponding Response constraints used in computing the CONMIN constraints.
- RealArray `constraintMappingMultipliers` a container of multipliers for mapping the Response constraints to the CONMIN constraints.
- RealArray `constraintMappingOffsets` a container of offsets for mapping the Response constraints to the CONMIN constraints.
- int `N1` Size variable for CONMIN arrays. See CONMIN manual.
- int `N2` Size variable for CONMIN arrays. See CONMIN manual.
- int `N3` Size variable for CONMIN arrays. See CONMIN manual.
- int `N4` Size variable for CONMIN arrays. See CONMIN manual.
- int `N5` Size variable for CONMIN arrays. See CONMIN manual.
- int `NFDG` Finite difference flag.
- int `IPRINT` Flag to control amount of output data.
- int `ITMAX` Flag to specify the maximum number of iterations.
- double `FDCH` Relative finite difference step size.
- double `FDCHM` Absolute finite difference step size.
- double `CT` Constraint thickness parameter.
- double `CTMIN` Minimum absolute value of CT used during optimization.
- double `CTL` Constraint thickness parameter for linear and side constraints.
- double `CTLMIN`
Minimum value of CTL used during optimization.

• double DELFUN
  Relative convergence criterion threshold.

• double DABFUN
  Absolute convergence criterion threshold.

• double * conminDesVars
  Array of design variables used by CONMIN (length N1 = numdv+2)

• double * conminLowerBnds
  Array of lower bounds used by CONMIN (length N1 = numdv+2)

• double * conminUpperBnds
  Array of upper bounds used by CONMIN (length N1 = numdv+2)

• double * S
  Internal CONMIN array.

• double * G1
  Internal CONMIN array.

• double * G2
  Internal CONMIN array.

• double * B
  Internal CONMIN array.

• double * C
  Internal CONMIN array.

• int * MS1
  Internal CONMIN array.

• double * SCAL
  Internal CONMIN array.

• double * DF
  Internal CONMIN array.

• double * A
  Internal CONMIN array.

• int * ISC
  Internal CONMIN array.

• int * IC
  Internal CONMIN array.

Additional Inherited Members

13.16.1 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 NPSOL-Optimizer 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.

13.16.2 Member Data Documentation

int conminInfo [private]
INFO from CONMIN manual.
Information requested by CONMIN: 1 = evaluate objective and constraints, 2 = evaluate gradients of objective and constraints.
Referenced by CONMINOptimizer::find_optimum(), and CONMINOptimizer::initialize().

int printControl [private]
IPRINT from CONMIN manual (controls output verbosity)
Values range from 0 (nothing) to 4 (most output). 0 = nothing, 1 = initial and final function information, 2 = all of #1 plus function value and design vars at each iteration, 3 = all of #2 plus constraint values and direction vectors, 4 = all of #3 plus gradients of the objective function and constraints, 5 = all of #4 plus proposed design vector, plus objective and constraint functions from the 1-D search
Referenced by CONMINOptimizer::initialize().

RealVector constraintValues [private]
array of nonlinear constraint values passed to CONMIN
This array must be of nonzero length and must contain only one-sided inequality constraints which are <= 0 (which requires a transformation from 2-sided inequalities and equalities).
Referenced by CONMINOptimizer::allocate_workspace(), and CONMINOptimizer::find_optimum().

SizetArray constraintMappingIndices [private]
a container of indices for referencing the corresponding Response constraints used in computing the CONMIN constraints.
The length of the container corresponds to the number of CONMIN constraints, and each entry in the container points to the corresponding DAKOTA constraint.
Referenced by CONMINOptimizer::allocate_constraints(), and CONMINOptimizer::find_optimum().

RealArray constraintMappingMultipliers [private]
a container of multipliers for mapping the Response constraints to the CONMIN constraints.
The length of the container corresponds to the number of CONMIN constraints, and each entry in the container stores a multiplier for the DAKOTA constraint identified with constraintMappingIndices. These multipliers are currently +1 or -1.
Referenced by CONMINOptimizer::allocate_constraints(), and CONMINOptimizer::find_optimum().
13.16. CONMINOPTIMIZER CLASS REFERENCE

**RealArray constraintMappingOffsets [private]**

A container of offsets for mapping the Response constraints to the CONMIN constraints.

The length of the container corresponds to the number of CONMIN constraints, and each entry in the container stores an offset for the DAKOTA constraint identified with constraintMappingIndices. These offsets involve inequality bounds or equality targets, since CONMIN assumes constraint allowables = 0.

Referenced by CONMINOptimizer::allocate_constraints(), and CONMINOptimizer::find_optimum().

**int N1 [private]**

Size variable for CONMIN arrays. See CONMIN manual.

\[N_1 = \text{number of variables} + 2\]

Referenced by CONMINOptimizer::allocate_workspace(), CONMINOptimizer::find_optimum(), and CONMINOptimizer::initialize_run().

**int N2 [private]**

Size variable for CONMIN arrays. See CONMIN manual.

\[N_2 = \text{number of constraints} + 2 \times (\text{number of variables})\]

Referenced by CONMINOptimizer::allocate_workspace(), and CONMINOptimizer::find_optimum().

**int N3 [private]**

Size variable for CONMIN arrays. See CONMIN manual.

\[N_3 = \text{Maximum possible number of active constraints}\]

Referenced by CONMINOptimizer::allocate_workspace(), and CONMINOptimizer::find_optimum().

**int N4 [private]**

Size variable for CONMIN arrays. See CONMIN manual.

\[N_4 = \text{Maximum}(N_3, \text{number of variables})\]

Referenced by CONMINOptimizer::allocate_workspace(), and CONMINOptimizer::find_optimum().

**int N5 [private]**

Size variable for CONMIN arrays. See CONMIN manual.

\[N_5 = 2 \times (N_4)\]

Referenced by CONMINOptimizer::allocate_workspace(), and CONMINOptimizer::find_optimum().

**double CT [private]**

Constraint thickness parameter.

The value of CT decreases in magnitude during optimization.

Referenced by CONMINOptimizer::find_optimum(), and CONMINOptimizer::initialize().

**double* S [private]**

Internal CONMIN array.

Move direction in N-dimensional space.

Referenced by CONMINOptimizer::allocate_workspace(), CONMINOptimizer::deallocate_workspace(), and CONMINOptimizer::find_optimum().
**double** G1  [**private**]

Internal CONMIN array.
Temporary storage of constraint values.
Referenced by CONMINOptimizer::allocate_workspace(), CONMINOptimizer::deallocate_workspace(), and CONMINOptimizer::find_optimum().

**double** G2  [**private**]

Internal CONMIN array.
Temporary storage of constraint values.
Referenced by CONMINOptimizer::allocate_workspace(), CONMINOptimizer::deallocate_workspace(), and CONMINOptimizer::find_optimum().

**double** B  [**private**]

Internal CONMIN array.
Temporary storage for computations involving array S.
Referenced by CONMINOptimizer::allocate_workspace(), CONMINOptimizer::deallocate_workspace(), and CONMINOptimizer::find_optimum().

**double** C  [**private**]

Internal CONMIN array.
Temporary storage for use with arrays B and S.
Referenced by CONMINOptimizer::allocate_workspace(), CONMINOptimizer::deallocate_workspace(), and CONMINOptimizer::find_optimum().

**int** MS1  [**private**]

Internal CONMIN array.
Temporary storage for use with arrays B and S.
Referenced by CONMINOptimizer::allocate_workspace(), CONMINOptimizer::deallocate_workspace(), and CONMINOptimizer::find_optimum().

**double** SCAL  [**private**]

Internal CONMIN array.
Vector of scaling parameters for design parameter values.
Referenced by CONMINOptimizer::allocate_workspace(), CONMINOptimizer::deallocate_workspace(), and CONMINOptimizer::find_optimum().

**double** DF  [**private**]

Internal CONMIN array.
Temporary storage for analytic gradient data.
Referenced by CONMINOptimizer::allocate_workspace(), CONMINOptimizer::deallocate_workspace(), and CONMINOptimizer::find_optimum().
**13.17. CONSTRAINTS CLASS REFERENCE**

`double* A [private]`

Internal CONMIN array.
Temporary 2-D array for storage of constraint gradients.
Referenced by CONMINOptimizer::allocate_workspace(), CONMINOptimizer::deallocate_workspace(), and CONMINOptimizer::find_optimum().

`int* ISC [private]`

Internal CONMIN array.
Array of flags to identify linear constraints. (not used in this implementation of CONMIN)
Referenced by CONMINOptimizer::allocate_workspace(), CONMINOptimizer::deallocate_workspace(), CONMINOptimizer::find_optimum(), and CONMINOptimizer::initialize_run().

`int* IC [private]`

Internal CONMIN array.
Array of flags to identify active and violated constraints
Referenced by CONMINOptimizer::allocate_workspace(), CONMINOptimizer::deallocate_workspace(), CONMINOptimizer::find_optimum(), and CONMINOptimizer::initialize_run().

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

- CONMINOptimizer.hpp
- CONMINOptimizer.cpp

### 13.17 Constraints Class Reference

Base class for the variable constraints class hierarchy.

Inheritance diagram for Constraints:

```
  Constraints
  |      |
  |      |
  v      v
MixedVarConstraints  RelaxedVarConstraints
```

**Public Member Functions**

- **Constraints ()**
  *default constructor*

- **Constraints (const ProblemDescDB &prob_db, const SharedVariablesData &svd)**
  *standard constructor*

- **Constraints (const SharedVariablesData &svd)**
  *alternate constructor for instantiations on the fly*

- **Constraints (const Constraints &con)**
  *copy constructor*

- virtual **~Constraints ()**
  *destructor*
• **Constraints operator= (const Constraints &con)**
  assignment operator
• **virtual void write (std::ostream &s) const**
  write a variable constraints object to an std::ostream
• **virtual void read (std::istream &s)**
  read a variable constraints object from an std::istream
• **virtual void reshape (const SizetArray &vc_totals)**
  reshape the lower/upper bound arrays within the Constraints hierarchy
• **const RealVector & continuous_lower_bounds () const**
  return the active continuous variable lower bounds
• **void continuous_lower_bounds (const RealVector &cl_bnds)**
  set the active continuous variable lower bounds
• **void continuous_lower_bound (Real cl_bnd, size_t i)**
  set an active continuous variable lower bound
• **const RealVector & continuous_upper_bounds () const**
  return the active continuous variable upper bounds
• **void continuous_upper_bounds (const RealVector &cu_bnds)**
  set the active continuous variable upper bounds
• **void continuous_upper_bound (Real cu_bnd, size_t i)**
  set an active continuous variable upper bound
• **const IntVector & discrete_int_lower_bounds () const**
  return the active discrete variable lower bounds
• **void discrete_int_lower_bounds (const IntVector &dil_bnds)**
  set the active discrete variable lower bounds
• **void discrete_int_lower_bound (int dil_bnd, size_t i)**
  set an active discrete variable lower bound
• **const IntVector & discrete_int_upper_bounds () const**
  return the active discrete variable upper bounds
• **void discrete_int_upper_bounds (const IntVector &diu_bnds)**
  set the active discrete variable upper bounds
• **void discrete_int_upper_bound (int diu_bnd, size_t i)**
  set an active discrete variable upper bound
• **const RealVector & discrete_real_lower_bounds () const**
  return the active discrete variable lower bounds
• **void discrete_real_lower_bounds (const RealVector &drl_bnds)**
  set the active discrete variable lower bounds
• **void discrete_real_lower_bound (Real drl_bnd, size_t i)**
  set an active discrete variable lower bound
• **const RealVector & discrete_real_upper_bounds () const**
  return the active discrete variable upper bounds
• **void discrete_real_upper_bounds (const RealVector &dru_bnds)**
  set the active discrete variable upper bounds
• **void discrete_real_upper_bound (Real dru_bnd, size_t i)**
set an active discrete variable upper bound

- const RealVector & inactive_continuous_lower_bounds () const
  return the inactive continuous lower bounds
- void inactive_continuous_lower_bounds (const RealVector &icl_bnds)
  set the inactive continuous lower bounds
- const RealVector & inactive_continuous_upper_bounds () const
  return the inactive continuous upper bounds
- void inactive_continuous_upper_bounds (const RealVector &icu_bnds)
  set the inactive continuous upper bounds
- const IntVector & inactive_discrete_int_lower_bounds () const
  return the inactive discrete lower bounds
- void inactive_discrete_int_lower_bounds (const IntVector &idil_bnds)
  set the inactive discrete lower bounds
- const IntVector & inactive_discrete_int_upper_bounds () const
  return the inactive discrete upper bounds
- void inactive_discrete_int_upper_bounds (const IntVector &idiu_bnds)
  set the inactive discrete upper bounds
- const RealVector & inactive_discrete_real_lower_bounds () const
  return the inactive discrete lower bounds
- void inactive_discrete_real_lower_bounds (const RealVector &idrl_bnds)
  set the inactive discrete lower bounds
- const RealVector & inactive_discrete_real_upper_bounds () const
  return the inactive discrete upper bounds
- void inactive_discrete_real_upper_bounds (const RealVector &idru_bnds)
  set the inactive discrete upper bounds
- const RealVector & all_continuous_lower_bounds () const
  returns a single array with all continuous lower bounds
- void all_continuous_lower_bounds (const RealVector &acl_bnds)
  sets all continuous lower bounds using a single array
- void all_continuous_lower_bound (Real acl_bnd, size_t i)
  set a lower bound within the all continuous lower bounds array
- const RealVector & all_continuous_upper_bounds () const
  returns a single array with all continuous upper bounds
- void all_continuous_upper_bounds (const RealVector &acu_bnds)
  sets all continuous upper bounds using a single array
- void all_continuous_upper_bound (Real acu_bnd, size_t i)
  set an upper bound within the all continuous upper bounds array
- const IntVector & all_discrete_int_lower_bounds () const
  returns a single array with all discrete lower bounds
- void all_discrete_int_lower_bounds (const IntVector &adil_bnds)
  sets all discrete lower bounds using a single array
- void all_discrete_int_lower_bound (int adil_bnd, size_t i)
  set a lower bound within the all discrete lower bounds array
• const IntVector & \texttt{all\_discrete\_int\_upper\_bounds} () const
  returns a single array with all discrete upper bounds

• void \texttt{all\_discrete\_int\_upper\_bounds} (const IntVector &adiu_bnds)
  sets all discrete upper bounds using a single array

• void \texttt{all\_discrete\_int\_upper\_bound} (int adiu_bnd, size_t i)
  set an upper bound within the all discrete upper bounds array

• const RealVector & \texttt{all\_discrete\_real\_lower\_bounds} () const
  returns a single array with all discrete lower bounds

• void \texttt{all\_discrete\_real\_lower\_bounds} (const RealVector &adrl_bnds)
  sets all discrete lower bounds using a single array

• void \texttt{all\_discrete\_real\_lower\_bound} (Real adrl_bnd, size_t i)
  set a lower bound within the all discrete lower bounds array

• const RealVector & \texttt{all\_discrete\_real\_upper\_bounds} () const
  returns a single array with all discrete upper bounds

• void \texttt{all\_discrete\_real\_upper\_bounds} (const RealVector &adru_bnds)
  sets all discrete upper bounds using a single array

• void \texttt{all\_discrete\_real\_upper\_bound} (Real adru_bnd, size_t i)
  set an upper bound within the all discrete upper bounds array

• size_t \texttt{num\_linear\_ineq\_constraints} () const
  return the number of linear inequality constraints

• size_t \texttt{num\_linear\_eq\_constraints} () const
  return the number of linear equality constraints

• const RealMatrix & \texttt{linear\_ineq\_constraint\_coeffs} () const
  return the linear inequality constraint coefficients

• void \texttt{linear\_ineq\_constraint\_coeffs} (const RealMatrix &lin_ineq_coeffs)
  set the linear inequality constraint coefficients

• const RealVector & \texttt{linear\_ineq\_constraint\_lower\_bounds} () const
  return the linear inequality constraint lower bounds

• void \texttt{linear\_ineq\_constraint\_lower\_bounds} (const RealVector &lin_ineq_l_bnds)
  set the linear inequality constraint lower bounds

• const RealVector & \texttt{linear\_ineq\_constraint\_upper\_bounds} () const
  return the linear inequality constraint upper bounds

• void \texttt{linear\_ineq\_constraint\_upper\_bounds} (const RealVector &lin_ineq_u_bnds)
  set the linear inequality constraint upper bounds

• const RealMatrix & \texttt{linear\_eq\_constraint\_coeffs} () const
  return the linear equality constraint coefficients

• void \texttt{linear\_eq\_constraint\_coeffs} (const RealMatrix &lin_eq_coeffs)
  set the linear equality constraint coefficients

• const RealVector & \texttt{linear\_eq\_constraint\_targets} () const
  return the linear equality constraint targets

• void \texttt{linear\_eq\_constraint\_targets} (const RealVector &lin_eq_targets)
  set the linear equality constraint targets

• size_t \texttt{num\_nonlinear\_ineq\_constraints} () const
13.17. CONSTRAINTS CLASS REFERENCE

- return the number of nonlinear inequality constraints
  
  ```cpp
  size_t num_nonlinear_ineq_constraints () const
  ```

- return the number of nonlinear equality constraints
  
  ```cpp
  const RealVector & nonlinear_eq_constraint_lower_bounds () const
  ```

- return the nonlinear inequality constraint lower bounds
  
  ```cpp
  void nonlinear_ineq_constraint_lower_bounds (const RealVector & nln_ineq_l_bnds)
  ```

- set the nonlinear inequality constraint lower bounds
  
  ```cpp
  const RealVector & nonlinear_ineq_constraint_upper_bounds () const
  ```

- return the nonlinear inequality constraint upper bounds
  
  ```cpp
  void nonlinear_ineq_constraint_upper_bounds (const RealVector & nln_ineq_u_bnds)
  ```

- set the nonlinear inequality constraint upper bounds
  
  ```cpp
  const RealVector & nonlinear_eq_constraint_targets () const
  ```

- return the nonlinear equality constraint targets
  
  ```cpp
  void nonlinear_eq_constraint_targets (const RealVector & nln_eq_targets)
  ```

- set the nonlinear equality constraint targets
  
  ```cpp
  Constraints copy () const
  ```

  * for use when a deep copy is needed (the representation is not shared)
  
  ```cpp
  void reshape (size_t num_nln_ineq_cons, size_t num_nln_eq_cons, size_t num_lin_ineq_cons, size_t num_lin_eq_cons, const SizetArray & vc_totals)
  ```

  * reshape the linear/nonlinear/bound constraint arrays within the Constraints hierarchy

- void reshape (size_t num_nln_ineq_cons, size_t num_nln_eq_cons, size_t num_lin_ineq_cons, size_t num_lin_eq_cons)

  * reshape the linear/nonlinear constraint arrays within the Constraints hierarchy

- void inactive_view (short view2)

  * sets the inactive view based on higher level (nested) context

- bool is_null () const

  * function to check constraintsRep (does this envelope contain a letter)

### Protected Member Functions

- Constraints (BaseConstructor, const ProblemDescDB & problem_db, const SharedVariablesData & svd)

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

- Constraints (BaseConstructor, const SharedVariablesData & svd)

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

- virtual void build_active_views ()

  * construct active views of all variables bounds arrays

- virtual void build_inactive_views ()

  * construct inactive views of all variables bounds arrays

- void build_views ()

  * construct active/inactive views of all variables arrays

- void manage_linear_constraints (const ProblemDescDB & problem_db)

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

- `SharedVariablesData sharedVarsData`
  configuration data shared from a `Variables` instance
- `RealVector allContinuousLowerBnds`
  a continuous lower bounds array combining continuous design, uncertain, and continuous state variable types (all view).
- `RealVector allContinuousUpperBnds`
  a continuous upper bounds array combining continuous design, uncertain, and continuous state variable types (all view).
- `IntVector allDiscreteIntLowerBnds`
  a discrete lower bounds array combining discrete design and discrete state variable types (all view).
- `IntVector allDiscreteIntUpperBnds`
  a discrete upper bounds array combining discrete design and discrete state variable types (all view).
- `RealVector allDiscreteRealLowerBnds`
  a discrete lower bounds array combining discrete design and discrete state variable types (all view).
- `RealVector allDiscreteRealUpperBnds`
  a discrete upper bounds array combining discrete design and discrete state variable types (all view).
- `size_t numNonlinearIneqCons`
  number of nonlinear inequality constraints
- `size_t numNonlinearEqCons`
  number of nonlinear equality constraints
- `RealVector nonlinearIneqConLowerBnds`
  nonlinear inequality constraint lower bounds
- `RealVector nonlinearIneqConUpperBnds`
  nonlinear inequality constraint upper bounds
- `RealVector nonlinearEqConTargets`
  nonlinear equality constraint targets
- `size_t numLinearIneqCons`
  number of linear inequality constraints
- `size_t numLinearEqCons`
  number of linear equality constraints
- `RealMatrix linearIneqConCoeffs`
  linear inequality constraint coefficients
- `RealMatrix linearEqConCoeffs`
  linear equality constraint coefficients
- `RealVector linearIneqConLowerBnds`
  linear inequality constraint lower bounds
- `RealVector linearIneqConUpperBnds`
  linear inequality constraint upper bounds
- `RealVector linearEqConTargets`
  linear equality constraint targets
- `RealVector continuousLowerBnds`
  the active continuous lower bounds array view
13.17. CONSTRAINTS CLASS REFERENCE

- RealVector continuousUpperBnds
  the active continuous upper bounds array view
- IntVector discreteIntLowerBnds
  the active discrete lower bounds array view
- IntVector discreteIntUpperBnds
  the active discrete upper bounds array view
- RealVector discreteRealLowerBnds
  the active discrete lower bounds array view
- RealVector discreteRealUpperBnds
  the active discrete upper bounds array view
- RealVector inactiveContinuousLowerBnds
  the inactive continuous lower bounds array view
- RealVector inactiveContinuousUpperBnds
  the inactive continuous upper bounds array view
- IntVector inactiveDiscreteIntLowerBnds
  the inactive discrete lower bounds array view
- IntVector inactiveDiscreteIntUpperBnds
  the inactive discrete upper bounds array view
- RealVector inactiveDiscreteRealLowerBnds
  the inactive discrete lower bounds array view
- RealVector inactiveDiscreteRealUpperBnds
  the inactive discrete upper bounds array view

Private Member Functions

- Constraints * get_constraints (const ProblemDescDB &problem_db, const SharedVariablesData &svd)
  Used only by the constructor to initialize constraintsRep to the appropriate derived type.
- Constraints * get_constraints (const SharedVariablesData &svd) const
  Used by copy() to initialize constraintsRep to the appropriate derived type.

Private Attributes

- Constraints * constraintsRep
  pointer to the letter (initialized only for the envelope)
- int referenceCount
  number of objects sharing constraintsRep

13.17.1 Detailed Description

Base class for the variable constraints class hierarchy.

The Constraints class is the base class for the class hierarchy managing bound, linear, and nonlinear constraints. Using the variable lower and upper bounds arrays from the input specification, different derived classes define different views of this data. The linear and nonlinear constraint data is consistent in all views and is managed at the base class level. For memory efficiency and enhanced polymorphism, the variable constraints hierarchy employs the "letter/envelope idiom" (see Coplien "Advanced C++", p. 133), for which the base class (Constraints) serves as the envelope and one of the derived classes (selected in Constraints::get_constraints()) serves as the letter.
13.17.2 Constructor & Destructor Documentation

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.

Constraints ( const ProblemDescDB & problem_db, const SharedVariablesData & svd )

standard constructor

The envelope constructor only needs to extract enough data to properly execute get_constraints, since the constructor overloaded with BaseConstructor builds the actual base class data inherited by the derived classes.

References Dakota::abort_handler(), Constraints::constraintsRep, and Constraints::get_constraints().

Constraints ( const SharedVariablesData & svd )

alternate constructor for instantiations on the fly

Envelope constructor for instantiations on the fly. This constructor executes get_constraints(view), which invokes the default derived/base constructors, followed by a reshape() based on vars_comps.

References Dakota::abort_handler(), Constraints::constraintsRep, and Constraints::get_constraints().

Constraints ( const Constraints & con )

copy constructor

Copy constructor manages sharing of constraintsRep and incrementing of referenceCount.

References Constraints::constraintsRep, and Constraints::referenceCount.

~Constraints ( ) [virtual]

destructor

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

References Constraints::constraintsRep, and Constraints::referenceCount.

Constraints ( BaseConstructor, const ProblemDescDB & problem_db, const SharedVariablesData & svd ) [protected]

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

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

Constraints ( BaseConstructor, const SharedVariablesData & svd ) [protected]

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

This constructor is the one which must build the base class data for all derived classes. get_constraints() instantiates a derived class letter and the derived constructor selects this base class constructor in its initialization list (to
13.17. CONSTRAINTS CLASS REFERENCE

avoid recursion in the base class constructor calling get_constraints() again). Since the letter IS the representation, its rep pointer is set to NULL (an uninitialized pointer causes problems in ~Constraints).

13.17.3 Member Function Documentation

Constraints operator= ( const Constraints & con )

assignment operator


References Constraints::constraintsRep, and Constraints::referenceCount.

void reshape ( const SizetArray & vc_totals ) [virtual]

reshape the lower/upper bound arrays within the Constraints hierarchy

Resizes the derived bounds arrays.

Reimplemented in RelaxedVarConstraints, and MixedVarConstraints.

References Constraints::constraintsRep, and Constraints::reshape().

Referenced by DataFitSurrModel::DataFitSurrModel(), RecastModel::RecastModel(), MixedVarConstraints::reshape(), RelaxedVarConstraints::reshape(), and Constraints::reshape().

Constraints copy () const

for use when a deep copy is needed (the representation is not shared)

Deep copies are used for history mechanisms that catalogue permanent copies (should not change as the representation within userDefinedConstraints changes).

References Constraints::allContinuousLowerBnds, Constraints::allContinuousUpperBnds, Constraints::allDiscrete-IntLowerBnds, Constraints::allDiscreteIntUpperBnds, Constraints::allDiscreteRealLowerBnds, Constraints::allDiscreteRealUpperBnds, Constraints::build_views(), Constraints::constraintsRep, Constraints::get_constraints(), Constraints::linearEqConCoeffs, Constraints::linearEqConTargets, Constraints::linearEqConLowerBnds, Constraints::linearEqConUpperBnds, Constraints::nonlinearEqConTargets, Constraints::nonlinearEqConLowerBnds, Constraints::nonlinearEqConUpperBnds, Constraints::numLinearEqCons, Constraints::numNonlinearEqCons, and Constraints::sharedVarsData.

Referenced by SurrogateModel::force_rebuild(), and RecastModel::RecastModel().

void reshape ( size_t num_nln_ineq_cons, size_t num_nln_eq_cons, size_t num_lin_ineq_cons, size_t num_lin_eq_cons )

reshape the linear/nonlinear constraint arrays within the Constraints hierarchy

Resizes the linear and nonlinear constraint arrays at the base class. Does NOT currently resize the derived bounds arrays.

References Constraints::constraintsRep, Constraints::linearEqConTargets, Constraints::linearEqConLowerBnds, Constraints::linearEqConUpperBnds, Constraints::nonlinearEqConTargets, Constraints::nonlinearEqConLowerBnds, Constraints::nonlinearEqConUpperBnds, Constraints::numLinearEqCons, Constraints::numNonlinearEqCons, and Constraints::reshape().
CHAPTER 13. CLASS DOCUMENTATION

void build_views ( ) [inline], [protected]

construct active/inactive views of all variables arrays
    = EMPTY)
    = EMPTY)

References Constraints::build_active_views(), Constraints::build_inactive_views(), Constraints::sharedVarsData, and SharedVariablesData::view().

Referenced by Constraints::copy(), MixedVarConstraints::MixedVarConstraints(), RelaxedVarConstraints::RelaxedVarConstraints(), MixedVarConstraints::reshape(), and RelaxedVarConstraints::reshape().

void manage_linear_constraints ( const ProblemDescDB & problem_db ) [protected]

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

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

References Dakota::abort_handler(), Constraints::continuousLowerBnds, Dakota::copy_data(), Constraints::discreteIntLowerBnds, Constraints::discreteRealLowerBnds, ProblemDescDB::get_rv(), Constraints::linearEqConCoeffs, Constraints::linearEqConTargets, Constraints::linearIneqConCoeffs, Constraints::linearIneqConLowerBnds, Constraints::linearIneqConUpperBnds, Constraints::numLinearEqCons, and Constraints::numLinearIneqCons.

Referenced by MixedVarConstraints::MixedVarConstraints(), and RelaxedVarConstraints::RelaxedVarConstraints().

Constraints * get_constraints ( const ProblemDescDB & problem_db, const SharedVariablesData & svd ) [private]

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

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

References SharedVariablesData::view().

Referenced by Constraints::Constraints(), and Constraints::copy().

Constraints * get_constraints ( const SharedVariablesData & svd ) const [private]

Used by copy() to initialize constraintsRep to the appropriate derived type.

Initializes constraintsRep to the appropriate derived type, as given by the variables view. The default derived class constructors are invoked.

References SharedVariablesData::view().

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

- DakotaConstraints.hpp
- DakotaConstraints.cpp

13.18 DataEnvironment Class Reference

Handle class for environment specification data.
Public Member Functions

- **DataEnvironment ()**
  constructor
- **DataEnvironment (const DataEnvironment &)**
  copy constructor
- **~DataEnvironment ()**
  destructor
- **DataEnvironment & operator= (const DataEnvironment &)**
  assignment operator
- **void write (std::ostream &s) const**
  write a DataEnvironment object to an std::ostream
- **void read (MPIUnpackBuffer &s)**
  read a DataEnvironment object from a packed MPI buffer
- **void write (MPIPackBuffer &s) const**
  write a DataEnvironment object to a packed MPI buffer
- **DataEnvironmentRep * data_rep ()**
  return dataEnvRep

Private Attributes

- **DataEnvironmentRep * dataEnvRep**
  pointer to the body (handle-body idiom)

Friends

- class ProblemDescDB
- class NIDRProblemDescDB

13.18.1 Detailed Description

Handle class for environment specification data.

The DataEnvironment class is used to provide a memory management handle for the data in DataEnvironmentRep. It is populated by IDRProblemDescDB::environment_kwhandler() and is queried by the ProblemDescDB::get_<datatype>() functions. A single DataEnvironment object is maintained in ProblemDescDB::environment-Spec.

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

- DataEnvironment.hpp
- DataEnvironment.cpp

13.19 DataEnvironmentRep Class Reference

Body class for environment specification data.
Public Attributes

- bool graphicsFlag
  flags use of graphics by the environment (from the graphics specification in EnvIndControl)
- bool tabularDataFlag
  flags tabular data collection by the environment (from the tabular data specification in EnvIndControl)
- String tabularDataFile
  the filename used for tabular data collection by the environment (from the tabular file specification in EnvIndControl)
- int outputPrecision
  output precision for tabular and screen output
- bool resultsOutputFlag
  flags use of results output to default file
- String resultsOutputFile
  named file for results output
- String topMethodPointer
  method identifier for the environment (from the top method pointer specification)

Private Member Functions

- DataEnvironmentRep ()
  constructor
- ~DataEnvironmentRep ()
  destructor
- void write (std::ostream &s) const
  write a DataEnvironmentRep object to an std::ostream
- void read (MPIUnpackBuffer &s)
  read a DataEnvironmentRep object from a packed MPI buffer
- void write (MPIPackBuffer &s) const
  write a DataEnvironmentRep object to a packed MPI buffer

Private Attributes

- int referenceCount
  number of handle objects sharing this dataEnvironmentRep

Friends

- class DataEnvironment
  the handle class can access attributes of the body class directly
13.19.1 Detailed Description

Body class for environment specification data. The DataEnvironmentRep class is used to contain the data from the environment keyword specification. Default values are managed in the DataEnvironmentRep constructor. Data is public to avoid maintaining set/get functions, but is still encapsulated within ProblemDescDB since ProblemDescDB::environmentSpec is private.

The documentation for this class was generated from the following files:
- DataEnvironment.hpp
- DataEnvironment.cpp

13.20 DataFitSurrModel Class Reference

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

Inheritance diagram for DataFitSurrModel:

<table>
<thead>
<tr>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>SurrogateModel</td>
</tr>
<tr>
<td>DataFitSurrModel</td>
</tr>
</tbody>
</table>

Public Member Functions

- **DataFitSurrModel (ProblemDescDB &problem_db)**
  
  *constructor*

- **DataFitSurrModel (Iterator &dace_iterator, Model &actual_model, const String &approx_type, const UShortArray &approx_order, short corr_type, short corr_order, short data_order, short output_level, const String &point_reuse, const String &export_points_file=String(), bool export_annotated=true, const String &import_points_file=String(), bool import_annotated=true)**

  *alternate constructor for instantiations on the fly*

- **~DataFitSurrModel ()**

  *destructor*

- **void total_points (int points)**

  *set pointsTotal and pointsManagement mode*

Protected Member Functions

- **void derived_compute_response (const ActiveSet &set)**

  *portion of compute_response() specific to DataFitSurrModel*

- **void derived_asynch_compute_response (const ActiveSet &set)**

  *portion of asynch Compute_response() specific to DataFitSurrModel*

- **const IntResponseMap & derived_sync ( )**

  *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 & derived_interface ()
  return approxInterface
• void primary_response_fn_weights (const RealVector &wts, bool recurse_flag=true)
  set the relative weightings for multiple objective functions or least squares terms and optionally recurses into actualModel
• void surrogate_response_mode (short mode)
  set response Mode and pass any bypass request on to actualModel for any lower-level surrogates.
• void surrogate_function_indices (const IntSet &surr_fn_indices)
  (re)set the surrogate index set in SurrogateModel::surrogateFnIndices and ApproximationInterface::approxFnIndices
• void build_approximation ()
  Builds the local/multipoint/global approximation using daceIterator/actualModel to generate new data points.
• bool build_approximation (const Variables &vars, const IntResponsePair &response_pr)
  Builds the local/multipoint/global approximation using daceIterator/actualModel to generate new data points that augment the vars/response anchor point.
• void update_approximation (bool rebuild_flag)
  replaces the approximation data with daceIterator results and rebuilds the approximation if requested
• void update_approximation (const Variables &vars, const IntResponsePair &response_pr, bool rebuild_flag)
  replaces the anchor point, and rebuilds the approximation if requested
• void update_approximation (const VariablesArray &vars_array, const IntResponseMap &resp_map, bool rebuild_flag)
  replaces the current points array and rebuilds the approximation if requested
• void append_approximation (bool rebuild_flag)
  appends daceIterator results to a global approximation and rebuilds it if requested
• void append_approximation (const Variables &vars, const IntResponsePair &response_pr, bool rebuild_flag)
  appends a point to a global approximation and rebuilds it if requested
• void append_approximation (const VariablesArray &vars_array, const IntResponseMap &resp_map, bool rebuild_flag)
  appends an array of points to a global approximation and rebuilds it if requested
• void pop_approximation (bool save_surr_data, bool rebuild_flag=false)
  remove approximation data added on previous append_approximation() call or a specified number of points
• void restore_approximation ()
13.20. DATAFITSURRMODEL CLASS REFERENCE

restore a previous approximation data state

- bool restore_available ()
  query for whether a trial increment is restorable

- void finalize_approximation ()
  finalize data fit by applying all previous trial increments

- void store_approximation ()
  store the current data fit approximation for later combination

- void combine_approximation (short corr_type)
  combine the current data fit approximation with one previously stored

- SharedApproxData & shared_approximation ()
  retrieve the SharedApproxData from approxInterface

- std::vector< Approximation > & approximations ()
  retrieve the set of Approximations from approxInterface

- const RealVectorArray & approximation_coefficients ()
  return the approximation coefficients from each Approximation (request forwarded to approxInterface)

- void approximation_coefficients (const RealVectorArray &approx_coeffs)
  set the approximation coefficients within each Approximation (request forwarded to approxInterface)

- const RealVector & approximation_variances (const Variables &vars)
  return the approximation variance from each Approximation (request forwarded to approxInterface)

- const Pecos::SurrogateData & approximation_data (size_t index)
  return the approximation data from a particular Approximation (request forwarded to approxInterface)

- void component_parallel_mode (short mode)
  update component parallel mode for supporting parallelism in actualModel

- void derived_init_communicators (int max_eval_concurrency, bool recurse_flag=true)
  set up actualModel for parallel operations

- void derived_init_serial ()
  set up actualModel for serial operations.

- void derived_set_communicators (int max_eval_concurrency, bool recurse_flag=true)
  set active parallel configuration within actualModel

- void derived_free_communicators (int max_eval_concurrency, bool recurse_flag=true)
  deallocate communicator partitions for the DataFitSurrModel (request forwarded to actualModel)

- void serve (int max_eval_concurrency)
  Service actualModel job requests received from the master. Completes when a termination message is received from stop_servers().

- void stop_servers ()
  Executed by the master to terminate actualModel server operations when DataFitSurrModel iteration is complete.

- void inactive_view (short view, bool recurse_flag=true)
  update the Model's inactive view based on higher level (nested) context and optionally recurse into actualModel

- const String & interface_id () const
  return the approxInterface identifier

- int evaluation_id () const
  return the current evaluation id for the DataFitSurrModel

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

- void fine_grained_evaluation_counters()
  request fine-grained evaluation reporting within approxInterface and actualModel

- void print_evaluation_summary(std::ostream &s, bool minimal_header=false, bool relative_count=true) const
  print the evaluation summary for the DataFitSurrModel (request forwarded to approxInterface and actualModel)

- virtual void eval_tag_prefix(const String &eval_id_str)
  set the hierarchical eval ID tag prefix

Private Member Functions

- void import_points(bool annotated)
  optionally read surrogate data points from provided file

- void initialize_export()
  initialize file stream for exporting surrogate evaluations

- void manage_data_recastings()
  initialize manageRecasting and recastFlags for data import/export

- void export_point(int eval_id, const Variables &vars, const Response &resp)
  initialize file stream for exporting surrogate evaluations

- void derived_synchronize_approx(const IntResponseMap &approxRespMap, IntResponseMap &approxRespMapRekey)
  Common code for processing of approximate response maps shared by derived_synchronize() and derived_synchronize_nowait()

- void update_global()
  Updates fit arrays for global approximations.

- void update_local_multipoint()
  Updates fit arrays for local or multipoint approximations.

- void build_global()
  Builds a global approximation using daceIterator.

- void build_local_multipoint()
  Builds a local or multipoint approximation using actualModel.

- void update_actual_model()
  update actualModel with data from current variables/labels/bounds/targets

- void update_from_actual_model()
  update current variables/labels/bounds/targets with data from actualModel

- bool inside(const RealVector &c_vars, const IntVector &di_vars, const RealVector &dr_vars)
  test if c_vars and d_vars are within [c_min, c_max] and [d_min, d_max]
13.20. DATAFITSURRMODEL CLASS REFERENCE

Private Attributes

- int surrModelEvalCntr
  number of calls to `derived_compute_response()` or `derived_asynch_compute_response()`
- int pointsTotal
  total points the user specified to construct the surrogate
- short pointsManagement
  configuration for points management in `build_global()`
- String pointReuse
  type of point reuse for approximation builds: `all`, `region` (default if points file), or `none` (default if no points file)
- bool manageRecasting
  flag indicating need to manage data recastings when importing build data or exporting approximate evaluations
- BoolDeque recastFlags
  a key indicating which models within a model recursion involve recasting
- String importPointsFile
  file name from `import_points_file` specification
- String exportPointsFile
  file name from `export_points_file` specification
- bool exportAnnotated
  annotation setting for file export of variables and approximate responses
- std::ofstream exportFileStream
  file name for `export_points_file` specification
- VariablesList reuseFileVars
  array of variables sets read from the `import_points_file`
- ResponseList reuseFileResponses
  array of response sets read from the `import_points_file`
- Interface approxInterface
  manages the building and subsequent evaluation of the approximations (required for both global and local)
- Model actualModel
  the truth model which provides evaluations for building the surrogate (optional for global, required for local)
- Iterator daceIterator
  selects parameter sets on which to evaluate actualModel in order to generate the necessary data for building global approximations (optional for global since restart data may also be used)
- String evalTagPrefix
  cached evalTag Prefix from parents to use at compute_response time

Additional Inherited Members

13.20.1 Detailed Description

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

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

~DataFitSurrModel() [inline]

destructor

Virtual destructor handles referenceCount at Strategy level.
References DataFitSurrModel::exportFileStream, and DataFitSurrModel::exportPointsFile.

13.20.3 Member Function Documentation

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 correc-
tion (if active) to the results.
Reimplemented from Model.
References DiscrepancyCorrection::active(), Response::active_set(), DataFitSurrModel::actualModel, Discrepancy-
Correction::apply(), SurrogateModel::approxBuilds, DataFitSurrModel::approxInterface, SurrogateModel::asv_,
mapping(), DataFitSurrModel::build_approximation(), DataFitSurrModel::component_parallel_mode(), Discrepancy-
Correction::compute(), Model::compute_response(), Response::copy(), Model::current_response(), Model::current-
Response, Model::currentVariables, SurrogateModel::deltaCorr, Model::eval_tag_prefix(), DataFitSurrModel::eval-
TagPrefix, DataFitSurrModel::export_point(), SurrogateModel::force_rebuild(), Model::hierarchicalTagging, Interface-
::map(), Model::outputLevel, ActiveSet::request_vector(), SurrogateModel::response_mapping(), SurrogateModel-
::responseMode, DataFitSurrModel::surrModelEvalCntr, Response::update(), and DataFitSurrModel::update_actual-
_model().

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 re-
sults for return in derived_synchronize() below).
Reimplemented from Model.
References DataFitSurrModel::actualModel, SurrogateModel::approxBuilds, DataFitSurrModel::approxInterface,
SurrogateModel::asv_mapping(), Model::asynch_compute_response(), DataFitSurrModel::build_approximation(),
Variables::copy(), Model::currentResponse, Model::currentVariables, Model::eval_tag_prefix(), DataFitSurrModel-
::evalTagPrefix, Interface::evaluation_id(), Model::evaluation_id(), DataFitSurrModel::exportPointsFile, Surrogate-
Model::force_rebuild(), Model::hierarchicalTagging, Interface::map(), SurrogateModel::rawVarsMap, ActiveSet-
::request_vector(), SurrogateModel::responseMode, SurrogateModel::surrIdMap, DataFitSurrModel::surrModel-
EvalCntr, SurrogateModel::truthIdMap, and DataFitSurrModel::update_actual_model().

const IntResponseMap & derived_synchronize() [protected], [virtual]

portion of synchronize() specific to DataFitSurrModel

Blocking retrieval of asynchronous evaluations from actualModel, approxInterface, or both (mixed case). For
the approxInterface portion, apply correction (if active) to each response in the array. derived_synchronize() is
designed for the general case where derived_asynch_compute_response() may be inconsistent in its use of actual
evaluations, approximate evaluations, or both.
Reimplemented from Model.
References Dakota::abort_handler(), DataFitSurrModel::actualModel, DataFitSurrModel::approxInterface, SurrogateModel::cachedApproxRespMap, DataFitSurrModel::component_parallel_mode(), DiscrepancyCorrection::compute(), SurrogateModel::deltaCorr, DataFitSurrModel::derived_synchronize_approx(), Model::outputLevel, SurrogateModel::response_mapping(), SurrogateModel::responseMode, SurrogateModel::surrIdMap, SurrogateModel::surrResponseMap, Interface::synch(), Model::synchronize(), and SurrogateModel::truthIdMap.

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

portion of synchronize_nowait() specific to DataFitSurrModel

Nonblocking retrieval of asynchronous evaluations from actualModel, approxInterface, or both (mixed case).

For the approxInterface portion, apply correction (if active) to each response in the map. derived_synchronize_nowait() is designed for the general case where derived_asynch_compute_response() may be inconsistent in its use of actual evals, approx evals, or both.

Reimplemented from Model.

References Dakota::abort_handler(), DataFitSurrModel::actualModel, DataFitSurrModel::approxInterface, SurrogateModel::cachedApproxRespMap, DataFitSurrModel::component_parallel_mode(), DiscrepancyCorrection::compute(), SurrogateModel::deltaCorr, DataFitSurrModel::derived_synchronize_approx(), Model::outputLevel, SurrogateModel::response_mapping(), SurrogateModel::responseMode, Model::synchronize(), and SurrogateModel::truthIdMap.

void build_approximation() [protected], [virtual]

Builds the local/multipoint/global approximation using daceIterator/actualModel to generate new data points.

This function constructs a new approximation, discarding any previous data. It constructs any required data for SurrogateData::{vars,resp}Data and does not define an anchor point for SurrogateData::anchor{Vars,Resp}.

Reimplemented from Model.

References DataFitSurrModel::actualModel, SurrogateModel::approxBuilds, DataFitSurrModel::approxInterface, Interface::build_approximation(), DataFitSurrModel::build_global(), DataFitSurrModel::build_local_multipoint(), Interface::clear_current(), Constraints::continuous_lower_bounds(), Model::continuous_lower_bounds(), Constraints::continuous_upper_bounds(), Model::continuous_upper_bounds(), Constraints::discrete_int_lower_bounds(), Model::discrete_int_lower_bounds(), Constraints::discrete_int_upper_bounds(), Model::discrete_int_upper_bounds(), Constraints::discrete_real_lower_bounds(), Model::discrete_real_lower_bounds(), Constraints::discrete_real_upper_bounds(), Model::discrete_real_upper_bounds(), Model::is_null(), Dakota::strbegins(), Model::surrogateType, DataFitSurrModel::update_actual_model(), DataFitSurrModel::update_global(), DataFitSurrModel::update_local_multipoint(), and Model::userDefinedConstraints.

Referenced by DataFitSurrModel::derived_asynch_compute_response(), and DataFitSurrModel::derived_compute_response().

bool build_approximation ( const Variables & vars, const IntResponsePair & response_pr ) [protected], [virtual]

Builds the local/multipoint/global approximation using daceIterator/actualModel to generate new data points that augment the vars/response anchor point.

This function constructs a new approximation, discarding any previous data. It uses the passed data to populate SurrogateData::{vars,resp}Data and constructs any required data points for SurrogateData::{vars,resp}Data.

Reimplemented from Model.

References DataFitSurrModel::actualModel, SurrogateModel::approxBuilds, DataFitSurrModel::approxInterface, Interface::build_approximation(), DataFitSurrModel::build_global(), Interface::clear_current(), Constraints::continuous_lower_bounds(), Model::continuous_lower_bounds(), Constraints::continuous_upper_bounds(), Model::continuous_upper_bounds(), Constraints::discrete_int_lower_bounds(), Model::discrete_int_lower_bounds(), Constraints::discrete_int_upper_bounds(), Model::discrete_int_upper_bounds(), Constraints::discrete_real_lower_bounds(), Model::discrete_real_lower_bounds(), Constraints::discrete_real_upper_bounds(), Model::discrete_real_upper_bounds(), Constraints::discrete_real_bounds(), Model::discrete_bounds(), Constraints::discrete_bounds(), Model::is_null(), Dakota::strbegins(), Model::surrogateType, DataFitSurrModel::update_actual_model(), DataFitSurrModel::update_global(), DataFitSurrModel::update_local_multipoint(), and Model::userDefinedConstraints.
void update_approximation ( bool rebuild_flag ) [protected], [virtual]

replaces the approximation data with daceIterator results and rebuilds the approximation if requested.

This function populates/replaces SurrogateData::anchor{Vars,Resp} and rebuilds the approximation, if requested. It does not clear other data (i.e., SurrogateData::{vars,resp}Data) and does not update the actualModel with revised bounds, labels, etc. Thus, it updates data from a previous call to build_approximation(), and is not intended to be used in isolation.

Reimplemented from Model.

References SurrogateModel::approxBuilds, DataFitSurrModel::approxInterface, Model::numFns, Interface::rebuild_approximation(), Model::surrogateType, and Interface::update_approximation().

void update_approximation ( const Variables & vars, const IntResponsePair & response_pr, bool rebuild_flag ) [protected], [virtual]

replaces the anchor point, and rebuilds the approximation if requested.

This function populates/replaces SurrogateData::anchor{Vars,Resp} and rebuilds the approximation, if requested. It does not clear other data (i.e., SurrogateData::{vars,resp}Data) and does not update the actualModel with revised bounds, labels, etc. Thus, it updates data from a previous call to build_approximation(), and is not intended to be used in isolation.

Reimplemented from Model.

References SurrogateModel::approxBuilds, DataFitSurrModel::approxInterface, Model::numFns, Interface::rebuild_approximation(), Model::surrogateType, and Interface::update_approximation().

void update_approximation ( const VariablesArray & vars_array, const IntResponseMap & resp_map, bool rebuild_flag ) [protected], [virtual]

replaces the current points array and rebuilds the approximation if requested.

This function populates/replaces SurrogateData::{vars,resp}Data and rebuilds the approximation, if requested. It does not clear other data (i.e., SurrogateData::anchor{Vars,Resp}) and does not update the actualModel with revised bounds, labels, etc. Thus, it updates data from a previous call to build_approximation(), and is not intended to be used in isolation.

Reimplemented from Model.

References SurrogateModel::approxBuilds, DataFitSurrModel::approxInterface, Model::numFns, Interface::rebuild_approximation(), Model::surrogateType, and Interface::update_approximation().

void append_approximation ( bool rebuild_flag ) [protected], [virtual]

appends daceIterator results to a global approximation and rebuilds it if requested.

This function appends one point to SurrogateData::{vars,resp}Data and rebuilds the approximation, if requested. It does not modify other data (i.e., SurrogateData::anchor{Vars,Resp}) and does not update the actualModel with revised bounds, labels, etc. Thus, it appends data from a previous call to build_approximation(), and is not intended to be used in isolation.

Reimplemented from Model.
void append_approximation ( const Variables & vars, const IntResponsePair & response_pr, bool rebuild_flag ) [protected], [virtual]

appends a point to a global approximation and rebuilds it if requested

This function appends one point to SurrogateData::{vars,resp}Data and rebuilds the approximation, if requested. It does not modify other data (i.e., SurrogateData::anchor{Vars,Resp}) and does not update the actual Model with revised bounds, labels, etc. Thus, it appends to data from a previous call to build_approximation(), and is not intended to be used in isolation.

Reimplemented from Model.

References Iterator::all_responses(), Iterator::all_samples(), Iterator::all_variables(), Interface::append_approximation(), SurrogateModel::approxBuilds, DataFitSurrModel::approxInterface, Iterator::compact_mode(), DataFitSurrModel::daceIterator, Model::numFns, Interface::rebuild_approximation(), and Model::surrogateType.

void append_approximation ( const VariablesArray & vars_array, const IntResponseMap & resp_map, bool rebuild_flag ) [protected], [virtual]

appends an array of points to a global approximation and rebuilds it if requested

This function appends multiple points to SurrogateData::{vars,resp}Data and rebuilds the approximation, if requested. It does not modify other data (i.e., SurrogateData::anchor{Vars,Resp}) and does not update the actual Model with revised bounds, labels, etc. Thus, it appends to data from a previous call to build_approximation(), and is not intended to be used in isolation.

Reimplemented from Model.

References Iterator::all_responses(), SurrogateModel::approxBuilds, DataFitSurrModel::approxInterface, Model::numFns, Interface::rebuild_approximation(), and Model::surrogateType.

void derived_init_communicators ( int max_eval_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_eval_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.

References DataFitSurrModel::actualModel, DataFitSurrModel::approxInterface, DataFitSurrModel::daceIterator, Model::derivative_concurrency(), Iterator::init_communicators(), Iterator::is_null(), Model::is_null(), Iterator::iterated_model(), Iterator::maximum_evaluation_concurrency(), and Interface::minimum_points().

int evaluation_id ( ) const [inline], [protected], [virtual]

return the current evaluation id for the DataFitSurrModel

return the DataFitSurrModel evaluation count. Due to possibly intermittent use of surrogate bypass, this is not the same as either the approxInterface or actualModel model evaluation counts. It also does not distinguish duplicate evals.

Reimplemented from Model.

References DataFitSurrModel::surrModelEvalCntr.
void import_points ( bool annotated ) [private]

optionally read surrogate data points from provided file

  Constructor helper to read the points file once, if provided, and then reuse its data as appropriate within

build_global()

  References SharedVariablesData::cv(), and ActiveSet::derivative_vector().
  Referenced by DataFitSurrModel::DataFitSurrModel().

void initialize_export ( ) [private]

initialize file stream for exporting surrogate evaluations

  Constructor helper to export approximation-based evaluations to a file.
  Referenced by DataFitSurrModel::DataFitSurrModel().

void manage_data_recastings ( ) [private]

initialize manageRecasting and recastFlags for data import/export

  Constructor helper to manage model recastings for data import/export.
  Referenced by DataFitSurrModel::DataFitSurrModel().

void export_point ( int eval_id, const Variables & vars, const Response & resp ) [private]

initialize file stream for exporting surrogate evaluations

  Constructor helper to export approximation-based evaluations to a file.

  References Dakota::_NPOS, RecastModel::inverse_transform_response(), Model::model_rep(), and Recast-
  Model::transform_variables().

  Referenced by DataFitSurrModel::derived_compute_response().

void build_global ( ) [private]

Builds a global approximation using daceIterator.

  Determine points to use in building the approximation and then evaluate them on actualModel using dace-
  Iterator. Any changes to the bounds should be performed by setting them at a higher level (e.g., SurrBasedOpt-
  Strategy).

  References Dakota::abort_handler(), Iterator::active_set(), DataFitSurrModel::actualModel, Iterator::all_responses(),
  Iterator::all_samples(), Iterator::all_variables(), Interface::append_approximation(), Interface::approximation_data(),
  DataFitSurrModel::approxInterface, SurrogateModel::asv_mapping(), Iterator::compact_mode(), DataFitSurrModel-
  ::component_parallel_mode(), Variables::continuous_variables(), Model::currentVariables, Variables::cv(), Model-
  ::cv(), DataFitSurrModel::daceIterator, Dakota::data_pairs, Variables::discrete_int_variables(), Variables::discrete-
  real_variables(), Variables::div(), Model::div(), Variables::drv(), Model::drv(), Iterator::eval_tag_prefix(), Data-
  FitSurrModel::evalTagPrefix, Model::hierarchicalTagging, DataFitSurrModel::importPointsFile, DataFitSurrModel-
  ::inside(), Model::interface_id(), RecastModel::inverse_transform_variables(), Iterator::is_null(), Model::is_null(),
  DataFitSurrModel::manageRecasting, Interface::minimum_points(), Model::model_rep(), Model::modelList, Iterator-
  ::num_samples(), Model::outputLevel, DataFitSurrModel::pointReuse, DataFitSurrModel::pointsManagement, Data-
  FitSurrModel::pointsTotal, DataFitSurrModel::recastFlags, Interface::recommended_points(), ActiveSet::request-
  _vector(), DataFitSurrModel::reuseFileResponses, DataFitSurrModel::reuseFileVars, Iterator::run(), Iterator::sampling-
  _reset(), DataFitSurrModel::surrModelEvalCntr, and SurrogateModel::surrogateFnIndices.

  Referenced by DataFitSurrModel::build_approximation().
void build_local_multipoint() [private]

Builds a local or multipoint approximation using actualModel.

Evaluate the value, gradient, and possibly Hessian needed for a local or multipoint approximation using actualModel.

References Response::active_set(), DataFitSurrModel::actualModel, DataFitSurrModel::approxInterface, SurrogateModel::asv_mapping(), DataFitSurrModel::component_parallel_mode(), Model::compute_response(), Model::continuous_variable_ids(), Model::current_response(), Model::current_variables(), ActiveSet::derivative_vector(), Model::evaluation_id(), Model::hessian_type(), Model::numFns, ActiveSet::request_vector(), Dakota::strbegins(), Model::surrogateType, and Interface::update_approximation().

Referenced by DataFitSurrModel::build_approximation().

void update_actual_model() [private]

update actualModel with data from current variables/labels/bounds/targets

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

References Dakota::abort_handler().

Referenced by DataFitSurrModel::build_approximation(), DataFitSurrModel::derived_asynch_compute_response(), and DataFitSurrModel::derived_compute_response().

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.

References Dakota::abort_handler().

Referenced by DataFitSurrModel::DataFitSurrModel(), and DataFitSurrModel::update_from_subordinate_model().

13.20.4 Member Data Documentation

Model actualModel [private]

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

actualModel is unrestricted in type; arbitrary nestings are possible.

Referenced by DataFitSurrModel::build_approximation(), DataFitSurrModel::build_global(), DataFitSurrModel::build_local_multipoint(), DataFitSurrModel::DataFitSurrModel(), DataFitSurrModel::derived_asynch_compute_response(), DataFitSurrModel::derived_compute_response(), DataFitSurrModel::derived_free_communicators(), DataFitSurrModel::derived_init_communicators(), DataFitSurrModel::derived_init_serial(), DataFitSurrModel::derived_set_communicators(), DataFitSurrModel::derived_subordinate_models(), DataFitSurrModel::derived_synchronize(), DataFitSurrModel::derived_synchronize_nowait(), DataFitSurrModel::fine_grained_evaluation_counts(), DataFitSurrModel::inactive_view(), DataFitSurrModel::inside(), DataFitSurrModel::primary_response_fn_weights(), DataFitSurrModel::print_evaluation_summary(), DataFitSurrModel::serve(), DataFitSurrModel::stop_servers(), DataFitSurrModel::surrogate_response_mode(), DataFitSurrModel::truth_model(), DataFitSurrModel::update_from_subordinate_model(), DataFitSurrModel::update_global(), and DataFitSurrModel::update_local_multipoint().

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

- DataFitSurrModel.hpp
- DataFitSurrModel.cpp
13.21 DataInterface Class Reference

Handle class for interface specification data.

Public Member Functions

- **DataInterface** ()
  \textit{constructor}
- **DataInterface** (const DataInterface &)
  \textit{copy constructor}
- **\texttt{\~ DataInterface}** ()
  \textit{destructor}
- **DataInterface & operator=** (const DataInterface &)
  \textit{assignment operator}
- void **write** (std::ostream &s) const
  \textit{write a DataInterface object to an std::ostream}
- void **read** (MPIUnpackBuffer &s)
  \textit{read a DataInterface object from a packed MPI buffer}
- void **write** (MPIPackBuffer &s) const
  \textit{write a DataInterface object to a packed MPI buffer}
- DataInterfaceRep * **data_rep** ()
  return \texttt{dataIfaceRep}

Static Public Member Functions

- static bool **id_compare** (const DataInterface &di, const std::string &id)
  \textit{compares the idInterface attribute of DataInterface objects}

Private Attributes

- DataInterfaceRep * **dataIfaceRep**
  \textit{pointer to the body (handle-body idiom)}

Friends

- class ProblemDescDB
- class NIDRProblemDescDB

13.21.1 Detailed Description

Handle class for interface specification data.

The **DataInterface** class is used to provide a memory management handle for the data in DataInterface-Rep. It is populated by IDRProblemDescDB::interface_kwhandler() and is queried by the ProblemDescDB::get_<-<datatype>() functions. A list of DataInterface objects is maintained in ProblemDescDB::dataInterfaceList, one for each interface specification in an input file.

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

- DataInterface.hpp
- DataInterface.cpp
13.22 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
- DataMethodRep * data_rep ()
  return dataMethodRep

Static Public Member Functions

- static bool id_compare (const DataMethod &dm, const std::string &id)
  compares the idMethod attribute of DataMethod objects

Private Attributes

- DataMethodRep * dataMethodRep
  pointer to the body (handle-body idiom)

Friends

- class ProblemDescDB
- class NIDRProblemDescDB

13.22.1 Detailed Description

Handle class for method specification data.

The DataMethod class is used to provide a memory management handle for the data in DataMethodRep. It is populated by IDRProblemDescDB::method_kwhandler() and is queried by the ProblemDescDB::get_<datatype>() functions. A list of DataMethod objects is maintained in ProblemDescDB::dataMethodList, one for each method specification in an input file.

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

- DataMethod.hpp
- DataMethod.cpp
13.23 DataMethodRep Class Reference

Body class for method specification data.

Public Attributes

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

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

- short methodOutput
  method verbosity control: \{SILENT,QUIET,NORMAL,VERBOSE,DEBUG\}.OUTPUT (from the output specification in MethodIndControl)

- int maxIterations
  maximum number of iterations allowed for the method (from the max_iterations specification in MethodIndControl)

- int maxFunctionEvaluations
  maximum number of function evaluations allowed for the method (from the max_function_evaluations specification in MethodIndControl)

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

- bool methodUseDerivsFlag
  flag for usage of derivative data to enhance the computation of surrogate models (PCE/SC expansions, GP models for EGO/EGRA/EGIE) based on the use_derivatives specification

- Real convergenceTolerance
  iteration convergence tolerance for the method (from the convergence_tolerance specification in MethodIndControl)

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

- bool methodScaling
  flag indicating scaling status (from the scaling specification in MethodIndControl)

- size_t numFinalSolutions
  number of final solutions returned from the iterator

- RealVector linearIneqConstraintCoeffs
  coefficient matrix for the linear inequality constraints (from the linear_inequality_constraint_matrix specification in MethodIndControl)

- RealVector linearIneqLowerBnds
  lower bounds for the linear inequality constraints (from the linear_inequality_lower_bounds specification in MethodIndControl)

- RealVector linearIneqUpperBnds
  upper bounds for the linear inequality constraints (from the linear_inequality_upper_bounds specification in MethodIndControl)

- StringArray linearIneqScaleTypes
• RealVector linearIneqScales
  scaling factors for the linear inequality constraints (from the linear.inequality.scales specification in MethodIndControl)

• RealVector linearEqConstraintCoeffs
  coefficient matrix for the linear equality constraints (from the linear.equality.constraint_matrix specification in MethodIndControl)

• RealVector linearEqTargets
  targets for the linear equality constraints (from the linear.equality.targets specification in MethodIndControl)

• StringArray linearEqScaleTypes
  scaling types for the linear equality constraints (from the linear.equality.scales specification in MethodIndControl)

• RealVector linearEqScales
  scaling factors for the linear equality constraints (from the linear.equality.scales specification in MethodIndControl)

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

• unsigned short subMethod
  enum value for a sub-method type

• String subMethodName
  string identifier for a sub-method name within a multi-option method specification (e.g., from meta-iterators)

• String subModelPointer
  string pointer for a sub-model specification used by a meta-iterator

• String subMethodPointer
  string pointer for a sub-method specification used by a meta-iterator

• int iteratorServers
  number of servers for concurrent iterator parallelism (from the iterator_servers specification)

• int procsPerIterator
  number of processors for each concurrent iterator partition (from the processors_per_iterator specification)

• short iteratorScheduling
  type of scheduling ({DEFAULT,MASTER,PEER}_SCHEDULING) used in concurrent iterator parallelism (from the iterator_scheduling specification)

• StringArray hybridMethodNames
  array of methods for the sequential and collaborative hybrid meta-iterators (from the method_name_list specification)

• StringArray hybridModelPointers
  array of models for the sequential and collaborative hybrid meta-iterators (from the model_pointer_list specification)

• StringArray hybridMethodPointers
  array of methods for the sequential and collaborative hybrid meta-iterators (from the method_pointer_list specification)

• String hybridGlobalMethodName
• String **hybridGlobalModelPointer**
  
  *global model pointer for embedded hybrids (from the global_model_pointer specification)*

• String **hybridGlobalMethodPointer**
  
  *global method pointer for embedded hybrids (from the global_method_pointer specification)*

• String **hybridLocalMethodName**
  
  *local method name for embedded hybrids (from the local_method_name specification)*

• String **hybridLocalModelPointer**
  
  *local model pointer for embedded hybrids (from the local_model_pointer specification)*

• String **hybridLocalMethodPointer**
  
  *local method pointer for embedded hybrids (from the local_method_pointer specification)*

• Real **hybridLSProb**
  
  *local search probability for embedded hybrids (from the local_search_probability specification)*

• int **concurrentRandomJobs**
  
  *number of random jobs to perform in the pareto_set and multi_start meta-iterators (from the random_starts and random_weight_sets specifications)*

• RealVector **concurrentParameterSets**
  
  *user-specified (i.e., nonrandom) parameter sets to evaluate in the pareto_set and multi_start meta-iterators (from the starting_points and weight_sets specifications)*

• int **surrBasedLocalSoftConvLimit**
  
  *number of consecutive iterations with change less than convergenceTolerance required to trigger convergence within the surrogate-based local method (from the soft_convergence_limit specification in MethodSBL)*

• bool **surrBasedLocalLayerBypass**
  
  *flag to indicate user-specification of a bypass of any/all layerings in evaluating truth response values in SBL.*

• Real **surrBasedLocalTRInitSize**
  
  *initial trust region size in the surrogate-based local method (from the initial_size specification in MethodSBL)*

• Real **surrBasedLocalTRMinSize**
  
  *minimum trust region size in the surrogate-based local method (from the minimum_size specification in MethodSBL), if the trust region size falls below this threshold the SBL iterations are terminated (note: if kriging is used with SBL, the min trust region size is set to 1.0e-3 in attempt to avoid ill-conditioned matrices that arise in kriging over small trust regions)*

• Real **surrBasedLocalTRContractTrigger**
  
  *trust region minimum improvement level (ratio of actual to predicted decrease in objective fcn) in the surrogate-based local method (from the contract_threshold specification in MethodSBL), the trust region shrinks or is rejected if the ratio is below this value (“eta_1” in the Conn-Gould-Toint trust region book)*

• Real **surrBasedLocalTRExpandTrigger**
  
  *trust region sufficient improvement level (ratio of actual to predicted decrease in objective fcn) in the surrogate-based local method (from the expand_threshold specification in MethodSBL), the trust region expands if the ratio is above this value (“eta_2” in the Conn-Gould-Toint trust region book)*

• Real **surrBasedLocalTRContract**
  
  *trust region contraction factor in the surrogate-based local method (from the contraction_factor specification in MethodSBL)*

• Real **surrBasedLocalTRExpand**
trust region expansion factor in the surrogate-based local method (from the expansion_factor specification in MethodSBL)

- short surrBasedLocalSubProbObj
  
  SBL approximate subproblem objective: ORIGINAL_PRIMARY, SINGLE_OBJECTIVE, LAGRANGIAN_OBJECTIVE, or AUGMENTED_LAGRANGIAN_OBJECTIVE.

- short surrBasedLocalSubProbCon
  
  SBL approximate subproblem constraints: NO_CONSTRAINTS, LINEARIZED_CONSTRAINTS, or ORIGINAL_CONSTRAINTS.

- short surrBasedLocalMeritFn
  
  SBL merit function type: BASIC_PENALTY, ADAPTIVE_PENALTY, BASIC_LAGRANGIAN, or AUGMENTED_LAGRANGIAN.

- short surrBasedLocalAcceptLogic
  
  SBL iterate acceptance logic: TR_RATIO or FILTER.

- short surrBasedLocalConstrRelax
  
  SBL constraint relaxation method: NO_RELAX or HOMOTOPY.

- bool surrBasedGlobalReplacePts
  
  user-specified method for adding points to the set upon which the next surrogate is based in the surrogate-based global method.

- String dlDetails
  
  string of options for a dynamically linked solver

- void * dlLib
  
  handle to dynamically loaded library

- int verifyLevel
  
  the verify_level specification in MethodNPSOLDC

- Real functionPrecision
  
  the function_precision specification in MethodNPSOLDC and the EPSILON specification in NOMAD

- Real lineSearchTolerance
  
  the linesearch_tolerance specification in MethodNPSOLDC

- Real absConvTol
  
  absolute function convergence tolerance

- Real xConvTol
  
  x-convergence tolerance

- Real singConvTol
  
  singular convergence tolerance

- Real singRadius
  
  radius for singular convergence test

- Real falseConvTol
  
  false-convergence tolerance

- Real initTRRadius
  
  initial trust radius

- int covarianceType
  
  kind of covariance required

- bool regressDiag
  
  whether to print the regression diagnostic vector
The following are string and real-valued parameters:

- **String `searchMethod`**
  - The search method specification for Newton and nonlinear interior-point methods in MethodOPTPPDC
- **Real `gradientTolerance`**
  - The gradient tolerance specification in MethodOPTPPDC
- **Real `maxStep`**
  - The max step specification in MethodOPTPPDC
- **String `meritFn`**
  - The merit function specification for nonlinear interior-point methods in MethodOPTPPDC
- **Real `steplength_to_boundary`**
  - The steplength_to_boundary specification for nonlinear interior-point methods in MethodOPTPPDC
- **String `boxDivision`**
  - The division setting (major dimension or all dimensions) for the DIRECT method in MethodSCOLIBDIR
- **Real `maxBoxSize`**
  - The max_boxsize_limit for the DIRECT method in MethodSCOLIBDIR
- **Real `minBoxSize`**
  - The min_boxsize_limit for the DIRECT method in MethodSCOLIBDIR and MethodNCSUDC
- **String `globalBalanceParam`**
  - The global_balance_parameter for the DIRECT method in MethodSCOLIBDIR
- **Real `localBalanceParam`**
  - The local_balance_parameter for the DIRECT method in MethodSCOLIBDIR
- **Real `constraintPenalty`**
  - The initial constraint_penalty for COLINY methods in MethodAPPS, MethodSCOLIBDIR, MethodSCOLIBPS, MethodSCOLIBSW and MethodSCOLIBEA
- **Bool `constantPenalty`**
  - The constant_penalty flag for COLINY methods in MethodSCOLIBPS and MethodSCOLIBSW
- **Real `smoothFactor`**
  - The initial smoothFactor value for nonlinearly constrained APPS in MethodAPPSDC
- **Real `constrPenalty`**
  - The constrPenalty choice for nonlinearly constrained APPS in MethodAPPSDC
- **Real `initStepLength`**
  - The initStepLength choice for nonlinearly constrained APPS in MethodAPPSDC
- **Real `contractStepLength`**
  - The contractStepLength choice for nonlinearly constrained APPS in MethodAPPSDC
- **Real `threshStepLength`**
  - The threshStepLength choice for nonlinearly constrained APPS in MethodAPPSDC
- **Real `constraintPenalty`**
  - The initial constraint_penalty for COLINY methods in MethodSCOLIBPS, MethodSCOLIBSW and MethodSCOLIBEA
- **Bool `mutationAdaptive`**
  - The mutationAdaptive choice for COLINY methods in MethodSCOLIBSW and MethodSCOLIBEA

These parameters control various aspects of optimization algorithms, including search methods, tolerances, and constraints.
the non-adaptive specification for the coliny_ea method in MethodSCOLIBEA

* bool showMiscOptions
  
  the show_misc_options specification in MethodSCOLIBDC

* StringArray miscOptions
  
  the misc_options specification in MethodSCOLIBDC

* Real solnTarget
  
  the solution_target specification in MethodSCOLIBDC

* Real crossoverRate
  
  the crossover_rate specification for EA methods in MethodSCOLIBEA

* Real mutationRate
  
  the mutation_rate specification for EA methods in MethodSCOLIBEA

* Real mutationScale
  
  the mutation_scale specification for EA methods in MethodSCOLIBEA

* Real mutationMinScale
  
  the min_scale specification for mutation in EA methods in MethodSCOLIBEA

* Real initDelta
  
  the initial_delta specification for APPS/COBYLA/PS/SW methods in MethodAPPS, MethodSCOLIBCOB, MethodSCOLIBPS, and MethodSCOLIBSW

* Real threshDelta
  
  the threshold_delta specification for APPS/COBYLA/PS/SW methods in MethodAPPS, MethodSCOLIBCOB, MethodSCOLIBPS, and MethodSCOLIBSW

* Real contractFactor
  
  the contraction_factor specification for APPS/PS/SW methods in MethodAPPS, MethodSCOLIBPS, and MethodSCOLIBSW

* int newSolnsGenerated
  
  the new_solutions_generated specification for GA/EPSA methods in MethodSCOLIBEA

* int numberRetained
  
  the integer assignment to random, chc, or elitist in the replacement_type specification for GA/EPSA methods in MethodSCOLIBEA

* bool expansionFlag
  
  the no_expansion specification for APPS/PS/SW methods in MethodAPPS, MethodSCOLIBPS, and MethodSCOLIBSW

* int expandAfterSuccess
  
  the expand_after_success specification for PS/SW methods in MethodSCOLIBPS and MethodSCOLIBSW

* int contractAfterFail
  
  the contract_after_failure specification for the SW method in MethodSCOLIBSW

* int mutationRange
  
  the mutation_range specification for the pga_int method in MethodSCOLIBEA

* int totalPatternSize
  
  the total_pattern_size specification for PS methods in MethodSCOLIBPS

* bool randomizeOrderFlag
  
  the stochastic specification for the PS method in MethodSCOLIBPS

* String selectionPressure
  
  the fitness_type specification for EA methods in MethodSCOLIBEA
- String **replacementType**
  *the replacement type specification for EA methods in MethodSCOLIBEA*
- String **crossoverType**
  *the crossover type specification for EA methods in MethodSCOLIBEA*
- String **mutationType**
  *the mutation type specification for EA methods in MethodSCOLIBEA*
- String **exploratoryMoves**
  *the exploratory moves specification for the PS method in MethodSCOLIBPS*
- String **patternBasis**
  *the pattern basis specification for APPS/PS methods in MethodAPPS and MethodSCOLIBPS*
- String **betaSolverName**
  *beta solvers don’t need documentation*
- String **evalSynchronize**
  *the synchronization setting for parallel pattern search methods in MethodSCOLIBPS and MethodAPPS*
- size_t **numCrossPoints**
  *The number of crossover points or multi-point schemes.*
- size_t **numParents**
  *The number of parents to use in a crossover operation.*
- size_t **numOffspring**
  *The number of children to produce in a crossover operation.*
- String **fitnessType**
  *the fitness assessment operator to use.*
- String **convergenceType**
  *The means by which this JEGA should converge.*
- Real **percentChange**
  *The minimum percent change before convergence for a fitness tracker converger.*
- size_t **numGenerations**
  *The number of generations over which a fitness tracker converger should track.*
- Real **fitnessLimit**
  *The cutoff value for survival in fitness limiting selectors (e.g., below_limit selector).*
- Real **shrinkagePercent**
  *The minimum percentage of the requested number of selections that must take place on each call to the selector (0, 1).*
- String **nichingType**
  *The niching type.*
- RealVector **nicheVector**
  *The discretization percentage along each objective.*
- size_t **numDesigns**
  *The maximum number of designs to keep when using the max_designs nicher.*
- String **postProcessorType**
  *The post processor type.*
- RealVector **distanceVector**
  *The discretization percentage along each objective.*
• String initializationType
   The means by which the JEGA should initialize the population.
• String flatFile
   The filename to use for initialization.
• String logFile
   The filename to use for logging.
• int populationSize
   the population_size specification for GA methods in MethodSCOLIBEA
• bool printPopFlag
   The print_each_pop_flag to set the printing of the population at each generation.
• Real volBoxSize
   the volume_boxsize_limit for the DIRECT method in MethodNCSUDC
• int numSymbols
   the symbols specification for DACE methods
• bool mainEffectsFlag
   the main_effects specification for sampling methods in MethodDDACE
• bool latinizeFlag
   the latinize specification for FSU QMC and CVT methods in MethodFSUDACE
• bool volQualityFlag
   the quality_metrics specification for sampling methods (FSU QMC and CVT methods in 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
• String historyFile
   the HISTORY_FILE specification for NOMAD
• String displayFormat
   the DISPLAY_STATS specification for NOMAD
• Real vns
   the VNS specification for NOMAD
• bool showAllEval
   the DISPLAY_ALL_EVAL specification for NOMAD
• int numSamples
   the samples specification for NonD & DACE methods
• bool fixedSeedFlag
flag for fixing the value of the seed among different NonD/DACE sample sets. This results in the use of the same sampling stencil/pattern throughout an execution with repeated sampling.

- bool fixedSequenceFlag
  flag for fixing the sequence for Halton or Hammersley QMC sample sets. This results in the use of the same sampling stencil/pattern throughout an execution with repeated sampling.

- int previousSamples
  the number of previous samples when augmenting a LHS sample

- bool vbdFlag
  the var_based_decomp specification for a variety of sampling methods

- Real vbdDropTolerance
  the var_based_decomp tolerance for omitting index output

- unsigned short vbdOrder
  a sub-specification of vbdFlag: interaction order limit for calculation/output of component VBD indices

- short covarianceControl
  restrict the calculation of a full response covariance matrix for high dimensional outputs: {DEFAULT,DIAGONAL, FULL}.COVARIANCE

- String rngName
  the basic random-number generator for NonD

- short refinementType
  refinement type for stochastic expansions from dimension refinement keyword group

- short refinementControl
  refinement control for stochastic expansions from dimension refinement keyword group

- short nestingOverride
  override for default point nesting policy: NO_NESTING OVERRIDE, NESTED, or NON_NESTED

- short growthOverride
  override for default point growth restriction policy: NO_GROWTH OVERRIDE, RESTRICTED, or UNRESTRICTED

- short expansionType
  enumeration for u-space type that defines u-space variable targets for probability space transformations: EXTENDED_U (default), ASKEY_U, STD_NORMAL_U, or STD_UNIFORM_U

- bool piecewiseBasis
  boolean indicating presence of piecewise keyword

- short expansionBasisType
  enumeration for type of basis in sparse grid interpolation (Pecos::{NODAL,HIERARCHICAL} _INTERPOLANT) or regression (Pecos::{TENSOR_PRODUCT,TOTAL_ORDER,ADAPTED} _BASIS).

- UShortArray expansionOrder
  the expansion order specification in MethodNonDPCE

- SizetArray expansionSamples
  the expansion samples specification in MethodNonDPCE

- String expansionSampleType
  allows for incremental PCE construction using the incremental_lhs specification in MethodNonDPCE

- UShortArray quadratureOrder
  the quadrature order specification in MethodNonDPCE and MethodNonDSC

- UShortArray sparseGridLevel
the `sparse.grid_level` specification in `MethodNonDPCE`, `MethodNonDSC`, and other stochastic expansion-enabled methods

- RealVector `anisoDimPref`
  the dimension_preference specification for tensor and sparse grids and expansion orders in `MethodNonDPCE` and `MethodNonDSC`

- unsigned short `cubIntOrder`
  the cubature_integrand specification in `MethodNonDPCE`

- SizetArray `collocationPoints`
  the collocation_points specification in `MethodNonDPCE`

- Real `collocationRatio`
  the collocation_ratio specification in `MethodNonDPCE`

- Real `collocRatioTermsOrder`
  order applied to the number of expansion terms when applying or computing the collocation ratio within regression PCE; based on the ratio_order specification in `MethodNonDPCE`

- short `regressionType`
  type of regression: LS, OMP, BP, BPDN, LARS, or LASSO

- short `lsRegressionType`
  type of least squares regression: SVD or EQ CON QR

- RealVector `regressionNoiseTol`
  noise tolerance(s) for OMP, BPDN, LARS, and LASSO

- Real `regressionL2Penalty`
  L2 regression penalty for a variant of LASSO known as the elastic net method (default of 0 gives standard LASSO)

- bool `crossValidation`
  flag indicating the use of cross-validation across expansion orders (given a prescribed maximum order) and, for some methods, noise tolerances

- bool `normalizedCoeffs`
  flag indicating the output of PCE coefficients corresponding to normalized basis polynomials

- String `pointReuse`
  allows PCE construction to reuse points from previous sample sets or data import using the reuse_points specification in `MethodNonDPCE`

- bool `tensorGridFlag`
  flag for usage of a sub-sampled set of tensor-product grid points within regression PCE; based on the tensor_grid specification in `MethodNonDPCE`

- UShortArray `tensorGridOrder`
  order of tensor-product grid points that are sub-sampled within orthogonal least interpolation PCE; based on the tensor_grid specification in `MethodNonDPCE`

- String `expansionImportFile`
  the import_expansion_file specification in `MethodNonDPCE`

- String `expansionExportFile`
  the export_expansion_file specification in `MethodNonDPCE`

- unsigned short `sampleType`
  the sample_type specification in `MethodNonDMC`, `MethodNonDPCE`, and `MethodNonDSC`

- unsigned short `reliabilitySearchType`
the type of limit state search in MethodNonDLocalRel \(x_{\text{taylor\ mean}}, x_{\text{taylor\ mpp}}, x_{\text{two\ point}}, u_{\text{taylor\ mean}}, u_{\text{taylor\ mpp}}, u_{\text{two\ point}}, \text{or}\ no_{\text{approx}}\) or MethodNonDGlobalRel \(x_{\text{gaussian\ process}}\) or \(u_{\text{gaussian\ process}}\)

- String reliabilityIntegration
  - the \textit{first\ order} or \textit{second\ order} integration selection in MethodNonDLocalRel
- unsigned short integrationRefine
  - the import, \textit{adapt\ import}, or \textit{mm\ adapt\ import} integration refinement selection in MethodNonDLocalRel, MethodNonDPCE, and MethodNonDSC
- int refineSamples
  - the refinement samples selection in MethodNonDLocalRel, MethodNonDPCE, and MethodNonDSC
- short distributionType
  - the distribution cumulative or complementary specification in MethodNonD
- short responseLevelTarget
  - the compute probabilities, reliabilities, or gen\_reliabilities specification in MethodNonD
- short responseLevelTargetReduce
  - the system series or parallel specification in MethodNonD
- RealVectorArray responseLevels
  - the response\_levels specification in MethodNonD
- RealVectorArray probabilityLevels
  - the probability\_levels specification in MethodNonD
- RealVectorArray reliabilityLevels
  - the reliability\_levels specification in MethodNonD
- RealVectorArray genReliabilityLevels
  - the gen\_reliability\_levels specification in MethodNonD
- int emulatorSamples
  - the number of samples to construct a GP emulator for Bayesian calibration methods (MethodNonDBayesCalib)
- short emulatorType
  - the emulator specification in MethodNonDBayesCalib
- String mcmcType
  - the mcmc type specification in MethodNonDBayesCalib
- String rejectionType
  - the rejection type specification in MethodNonDBayesCalib
- String metropolisType
  - the metropolis type specification in MethodNonDBayesCalib
- RealVector proposalCovScale
  - the proposal\_covariance scale factor in MethodNonDBayesCalib
- Real likelihoodScale
  - the likelihood scale factor in MethodNonDBayesCalib
- String fitnessMetricType
  - the fitness metric type specification in MethodNonDAdaptive
- String batchSelectionType
  - the batch selection type specification in MethodNonDAdaptive
- int batchSize
The size of the batch (e.g. number of supplemental points added) to be added to the build points for an emulator at each iteration.

- `bool calibrateSigmaFlag`
  flag to indicate if the sigma terms should be calibrated in `MethodNonDBayesCalib`

- `int numChains`
  number of concurrent chains

- `int numCR`
  number of CR-factors

- `int crossoverChainPairs`
  number of crossover chain pairs

- `Real grThreshold`
  threshold for the Gelman-Rubin statistic

- `int jumpStep`
  how often to perform a long jump in generations

- `RealVector finalPoint`
  the final_point specification in `MethodPSVPS`

- `RealVector stepVector`
  the step_vector specification in `MethodPSVPS` and `MethodPSCPS`

- `int numSteps`
  the num_steps specification in `MethodPSVPS`

- `IntVector stepsPerVariable`
  the deltas_per_variable specification in `MethodPSCPS`

- `RealVector listOfPoints`
  the list_of_points specification in `MethodPSLPS`

- `String pstudyFilename`
  the import_points_file spec for a file-based parameter study

- `bool pstudyFileAnnotated`
  whether the parameter study points file is annotated

- `UShortArray varPartitions`
  the partitions specification for `PStudy` method in `MethodPSMPS`

- `Real refinementRate`
  rate of mesh refinement in Richardson extrapolation

- `String approxImportFile`
  the file name for point import in surrogate-based methods

- `bool approxImportAnnotated`
  whether the point import file is annotated (default true)

- `String approxExportFile`
  the file name for point export in surrogate-based methods

- `bool approxExportAnnotated`
  whether the point export file is annotated (default true)
Private Member Functions

- `DataMethodRep ()`
  
  constructor
- `~DataMethodRep ()`
  
  destructor
- `void write (std::ostream &s) const`
  
  write a DataInterfaceRep object to an std::ostream
- `void read (MPIUnpackBuffer &s)`
  
  read a DataInterfaceRep object from a packed MPI buffer
- `void write (MPIPackBuffer &s) const`
  
  write a DataInterfaceRep object to a packed MPI buffer

Private Attributes

- `int referenceCount`
  
  number of handle objects sharing this dataMethodRep

Friends

- `class DataMethod`
  
  the handle class can access attributes of the body class directly

13.23.1 Detailed Description

Body class for method specification data.

The `DataMethodRep` class is used to contain the data from a method keyword specification. Default values are managed in the `DataMethodRep` constructor. Data is public to avoid maintaining set/get functions, but is still encapsulated within `ProblemDescDB` since `ProblemDescDB::dataMethodList` is private.

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

- `DataMethod.hpp`
- `DataMethod.cpp`

13.24 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 &)`
13.25. DATAMODELREP CLASS REFERENCE

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

• DataModelRep * data_rep ()
  return dataModelRep

Static Public Member Functions

• static bool id_compare (const DataModel &dm, const std::string &id)
  compares the idModel attribute of DataModel objects

Private Attributes

• DataModelRep * dataModelRep
  pointer to the body (handle-body idiom)

Friends

• class ProblemDescDB
• class NIDRProblemDescDB

13.24.1 Detailed Description

Handle class for model specification data.

The DataModel class is used to provide a memory management handle for the data in DataModelRep. It is populated by IDRProblemDescDB::model_kwhandler() and is queried by the ProblemDescDB::get_<datatype>() functions. A list of DataModel objects is maintained in ProblemDescDB::dataModelList, one for each model specification in an input file.

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

• DataModel.hpp
• DataModel.cpp

13.25 DataModelRep Class Reference

Body class for model specification data.

Public Attributes

• String idModel
  string identifier for the model specification data set (from the id_model specification in ModelIndControl)

• String modelType
  model type selection: single, surrogate, or nested (from the model type specification in ModelIndControl)
• String `variablesPointer`
  - string pointer to the variables specification to be used by this model (from the `variables_pointer` specification in `ModelIndControl`)
• String `interfacePointer`
  - string pointer to the interface specification to be used by this model (from the `interface_pointer` specification in `ModelSingle` and the optional `interface_pointer` specification in `ModelNested`)
• String `responsesPointer`
  - string pointer to the responses specification to be used by this model (from the `responses_pointer` specification in `ModelIndControl`)
• bool `hierarchicalTags`
  - whether this model and its children will add hierarchy-based tags to eval ids
• String `subMethodPointer`
  - pointer to a sub-iterator used for global approximations (from the `dace_method_pointer` specification in `ModelSurrG`) or by nested models (from the `sub_method_pointer` specification in `ModelNested`)
• IntSet `surrogateFnIndices`
  - array specifying the response function set that is approximated
• String `surrogateType`
  - the selected surrogate type: local_taylor, multipoint_tana, global_neural_network,mars,orthogonal_polynomial,gaussian, polynomial,kriging), or hierarchical
• String `truthModelPointer`
  - pointer to the model specification for constructing the truth model used in building local, multipoint, and hierarchical approximations (from the `actual_model_pointer` specification in `ModelSurrL` and `ModelSurrMP` and the `high_fidelity_model_pointer` specification in `ModelSurrH`)
• String `lowFidelityModelPointer`
  - pointer to the low fidelity model specification used in hierarchical approximations (from the `low_fidelity_model_pointer` specification in `ModelSurrH`)
• int `pointsTotal`
  - user-specified lower bound on total points with which to build the model (if `reuse_points < pointsTotal`, new samples will make up the difference)
• short `pointsManagement`
  - points management configuration for `DataFitSurrModel`: DEFAULT_POINTS, MINIMUM_POINTS, or RECOMMENDED_POINTS
• String `approxPointReuse`
  - sample reuse selection for building global approximations: none, all, region, or file (from the `reuse_samples` specification in `ModelSurrG`)
• String `approxExportFile`
  - the file name from the `import_points_file` specification in `ModelSurrG`
  - bool `approxImportAnnotated`
    - whether the point import file is annotated (default true)
• String `approxExportModelFile`
  - the file name from the `export_model_file` specification in `ModelSurrG`
  - bool `approxExportAnnotated`
    - whether the point export file is annotated (default true)
- short `approxCorrectionType`
  correction type for global and hierarchical approximations: `NO_CORRECTION`, `ADDITIONAL_CORRECTION`, `MULTIPLICATIVE_CORRECTION`, or `COMBINED_CORRECTION` (from the `correction` specification in `ModelSurrG` and `ModelSurrH`)

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

- bool `modelUseDerivsFlag`
  flags the use of derivatives in building global approximations (from the `use_derivatives` specification in `ModelSurrG`)

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

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

- String `krigingOptMethod`
  optimization method to use in finding optimal correlation parameters: none, sampling, local, global

- short `krigingMaxTrials`
  maximum number of trials in optimization of kriging correlations

- `RealVector krigingMaxCorrelations`
  upper bound on kriging correlation vector

- `RealVector krigingMinCorrelations`
  lower bound on kriging correlation vector

- `Real krigingNugget`
  nugget value for kriging

- short `krigingFindNugget`
  option to have Kriging find the best nugget value to use

- short `mlsPolyOrder`
  polynomial order for moving least squares approximation

- short `mlsWeightFunction`
  weight function for moving least squares approximation

- short `rbfBases`
  bases for radial basis function approximation

- short `rbfMaxPts`
  maximum number of points for radial basis function approximation

- short `rbfMaxSubsets`
  maximum number of subsets for radial basis function approximation

- short `rbfMinPartition`
  minimum partition for radial basis function approximation

- short `marsMaxBases`
  maximum number of bases for MARS approximation

- String `marsInterpolation`
  interpolation type for MARS approximation
• short `annRandomWeight`
  random weight for artificial neural network approximation

• short `annNodes`
  number of nodes for artificial neural network approximation

• Real `annRange`
  range for artificial neural network approximation

• String `trendOrder`
  scalar integer indicating the order of the Gaussian process mean (0= constant, 1=linear, 2=quadratic, 3=cubic); from the `gaussian_process` specification in `ModelSurrG`

• bool `pointSelection`
  flag indicating the use of point selection in the Gaussian process

• StringArray `diagMetrics`
  List of diagnostic metrics the user requests to assess the goodness of fit for a surrogate model.

• bool `crossValidateFlag`
  flag indicating the use of cross validation on the metrics specified

• int `numFolds`
  number of folds to perform in cross validation

• Real `percentFold`
  percentage of data to withhold for cross validation process

• bool `pressFlag`
  flag indicating the use of PRESS on the metrics specified

• String `approxChallengeFile`
  the file name from the `challenge_points_file` specification in `ModelSurrG`

• bool `approxChallengeAnnotated`
  whether the challenge data file is annotated (default true)

• String `optionalInterfRespPointer`
  string pointer to the responses specification used by the optional interface in nested models (from the `optional INTERFACE responses_pointer` specification in `ModelNested`)

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

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

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

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

• int `subMethodServers`
  number of servers for concurrent sub-iterator parallelism
13.26. **DATARESPONSES CLASS REFERENCE**

- **int** `subMethodProcs`  
  *number of processors for each concurrent sub-iterator partition*
- **short** `subMethodScheduling`  
  *scheduling approach for concurrent sub-iterator parallelism: \{DEFAULT,MASTER,PEER\}_SCHEDULING*

**Private Member Functions**

- **`DataModelRep()`**  
  *constructor*
- **`~DataModelRep()`**  
  *destructor*
- **void** `write (std::ostream &s) const`  
  *write a `DataModelRep` object to an std::ostream*
- **void** `read (MPIUnpackBuffer &s)`  
  *read a `DataModelRep` object from a packed MPI buffer*
- **void** `write (MPIPackBuffer &s) const`  
  *write a `DataModelRep` object to a packed MPI buffer*

**Private Attributes**

- **int** `referenceCount`  
  *number of handle objects sharing this dataModelRep*

**Friends**

- **class** `DataModel`  
  *the handle class can access attributes of the body class directly*

13.25.1 **Detailed Description**

Body class for model specification data.

The `DataModelRep` class is used to contain the data from a model keyword specification. Default values are managed in the `DataModelRep` constructor. Data is public to avoid maintaining set/get functions, but is still encapsulated within `ProblemDescDB` since `ProblemDescDB::dataModelList` is private.

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

- `DataModel.hpp`
- `DataModel.cpp`

13.26 **DataResponses Class Reference**

Handle class for responses specification data.
CHAPTER 13. CLASS DOCUMENTATION

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
- **DataResponsesRep * data_rep ()**
  return dataRespRep

Static Public Member Functions

- static bool **id_compare (const DataResponses &dr, const std::string &id)**
  compares the idResponses attribute of DataResponses objects

Private Attributes

- **DataResponsesRep * dataRespRep**
  pointer to the body (handle-body idiom)

Friends

- class **ProblemDescDB**
- class **NIDRProblemDescDB**

13.26.1 Detailed Description

Handle class for responses specification data.

The DataResponses class is used to provide a memory management handle for the data in DataResponses-Rep. It is populated by IDRProblemDescDB::responses_kwhandler() and is queried by the ProblemDescDB::get_"<datatype>"() functions. A list of DataResponses objects is maintained in ProblemDescDB::dataResponsesList, one for each responses specification in an input file.

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

- DataResponses.hpp
- DataResponses.cpp
13.27 DataResponsesRep Class Reference

Body class for responses specification data.

Public Attributes

- **String `idResponses`**
  - string identifier for the responses specification data set (from the `id_responses` specification in RespSetId)
- **StringArray `responseLabels`**
  - the response labels array (from the `response_descriptors` specification in RespLabels)
- **size_t `numObjectiveFunctions`**
  - number of objective functions (from the `num_objective_functions` specification in RespFnOpt)
- **size_t `numNonlinearIneqConstraints`**
  - number of nonlinear inequality constraints (from the `num_nonlinear_inequality_constraints` specification in RespFnOpt)
- **size_t `numNonlinearEqConstraints`**
  - number of nonlinear equality constraints (from the `num_nonlinear_equality_constraints` specification in RespFnOpt)
- **size_t `numLeastSqTerms`**
  - number of least squares terms (from the `num_least_squares_terms` specification in RespFnLS)
- **size_t `numResponseFunctions`**
  - number of generic response functions (from the `num_response_functions` specification in RespFnGen)
- **StringArray `primaryRespFnSense`**
  - optimization sense for each objective function: minimize or maximize
- **RealVector `primaryRespFnWeights`**
  - vector of weightings for multiobjective optimization or weighted nonlinear least squares (from the `multi_objective_weights` specification in RespFnOpt and the `least_squares_weights` specification in RespFnLS)
- **RealVector `nonlinearIneqLowerBnds`**
  - vector of nonlinear inequality constraint lower bounds (from the `nonlinear_inequality_lower_bounds` specification in RespFnOpt)
- **RealVector `nonlinearIneqUpperBnds`**
  - vector of nonlinear inequality constraint upper bounds (from the `nonlinear_inequality_upper_bounds` specification in RespFnOpt)
- **RealVector `nonlinearEqTargets`**
  - vector of nonlinear equality constraint targets (from the `nonlinear_equality_targets` specification in RespFnOpt)
- **StringArray `primaryRespFnScaleTypes`**
  - vector of primary response function scaling types (from the `objective_function_scale_types` specification in RespFnOpt and the `least_squares_term_scale_types` specification in RespFnLS)
- **RealVector `primaryRespFScales`**
  - vector of primary response function scaling factors (from the `objective_function_scales` specification in RespFnOpt and the `least_squares_term_scales` specification in RespFnLS)
- **StringArray `nonlinearIneqScaleTypes`**
  - vector of nonlinear inequality constraint scaling types (from the `nonlinear_inequality_scale_types` specification in RespFnOpt)
• RealVector **nonlinearIneqScales**
  vector of nonlinear inequality constraint scaling factors (from the **nonlinear_inequality_scales** specification in RespFnOpt)

• StringArray **nonlinearEqScaleTypes**
  vector of nonlinear equality constraint scaling types (from the **nonlinear_equality_scale_types** specification in RespFnOpt)

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

• size_t **numExperiments**
  number of distinct experiments in experimental data

• IntVector **numReplicates**
  number of replicates in experimental data (e.g. one experiment run many times at the same configuration gives replicates)

• size_t **numExpConfigVars**
  number of experimental configuration vars (state variables) in each row of data

• size_t **numExpStdDeviations**
  whether to read num_responses standard deviations from each row of data file

• RealVector **expConfigVars**
  list of num_experiments x num_config_vars configuration variable values

• RealVector **expObservations**
  list of num_calibration_terms observation data

• RealVector **expStdDeviations**
  list of 1 or num_calibration_terms observation standard deviations

• String **expDataFileName**
  name of experimental data file containing response data (with optional state variable and sigma data) to read

• bool **expDataFileAnnotated**
  whether the experimental data is in annotated format

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

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

• bool **ignoreBounds**
  option to ignore bounds when doing finite differences (default is to honor bounds)

• bool **centralHess**
  Temporary(?) option to use old 2nd-order diffs when computing finite-difference Hessians; default is forward differences.

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

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

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

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

- **String fdGradStepType**
  type of finite difference step to use for numerical gradient: relative - step length is relative to \( x \) absolute - step length is what is specified bounds - step length is relative to range of \( x \)

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

- **String fdHessStepType**
  type of finite difference step to use for numerical Hessian: relative - step length is relative to \( x \) absolute - step length is what is specified bounds - step length is relative to range of \( x \)

- **IntSet idNumericalGrads**
  mixed gradient numerical identifiers (from the id_numerical_gradients specification in RespGradMixed)

- **IntSet idAnalyticGrads**
  mixed gradient analytic identifiers (from the id_analytic_gradients specification in RespGradMixed)

- **IntSet idNumericalHessians**
  mixed Hessian numerical identifiers (from the id_numerical_hessians specification in RespHessMixed)

- **IntSet idQuasiHessians**
  mixed Hessian quasi identifiers (from the id_quasi_hessians specification in RespHessMixed)

- **IntSet idAnalyticHessians**
  mixed Hessian analytic identifiers (from the id_analytic_hessians specification in RespHessMixed)

Private Member Functions

- **DataResponsesRep ()**
  constructor

- **~DataResponsesRep ()**
  destructor

- **void write (std::ostream &s) const**
  write a DataResponsesRep object to an std::ostream

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

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

Private Attributes

- **int referenceCount**
  number of handle objects sharing this dataResponsesRep
CHAPTER 13. CLASS DOCUMENTATION

Friends

- class DataResponses
  the handle class can access attributes of the body class directly

13.27.1 Detailed Description

Body class for responses specification data.

The DataResponsesRep class is used to contain the data from a responses keyword specification. Default values are managed in the DataResponsesRep constructor. Data is public to avoid maintaining set/get functions, but is still encapsulated within ProblemDescDB since ProblemDescDB::dataResponsesList is private.

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

- DataResponses.hpp
- DataResponses.cpp

13.28 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
- DataVariablesRep *data_rep ()
  return dataVarsRep
- 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
13.29. **DATAVARIABLESREP CLASS REFERENCE**

- `size_t uncertain ()`
  
  *return total number of uncertain variables*

- `size_t state ()`
  
  *return total number of state variables*

- `size_t continuous_variables ()`
  
  *return total number of continuous variables*

- `size_t discrete_variables ()`
  
  *return total number of discrete variables*

- `size_t total_variables ()`
  
  *return total number of variables*

### Static Public Member Functions

- `static bool id_compare (const DataVariables &dv, const std::string &id)`
  
  *compares the idVariables attribute of DataVariables objects*

### Private Attributes

- `DataVariablesRep * dataVarsRep`
  
  *pointer to the body (handle-body idiom)*

### Friends

- `class ProblemDescDB`
- `class NIDRProblemDescDB`

### 13.28.1 Detailed Description

Handle class for variables specification data.

The `DataVariables` class is used to provide a memory management handle for the data in `DataVariables-Rep`. It is populated by IDRProblemDescDB::variables_kwhandler() and is queried by the ProblemDescDB::get_<datatype>() functions. A list of `DataVariables` objects is maintained in ProblemDescDB::dataVariablesList, one for each variables specification in an input file.

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

- `DataVariables.hpp`
- `DataVariables.cpp`

### 13.29 DataVariablesRep Class Reference

Body class for variables specification data.
Public Attributes

- **String idVariables**
  - string identifier for the variables specification data set (from the id_variables specification in VarSetId)
- **short varsView**
  - user selection/override of variables view: {DEFAULT,ALL,DESIGN, UNCERTAIN,ALEATORY,UNCERTAIN,E-PISTEMIC,UNCERTAIN,STATE}_VIEW
- **short varsDomain**
  - user selection/override of variables domain: {DEFAULT,MIXED,RELAXED}_DOMAIN
- **bool uncertainVarsInitPt**
  - flag indicating user specification of initial points (for local optimization-based UQ methods) for at least one uncertain variable type
- **size_t numContinuousDesVars**
  - number of continuous design variables (from the continuous_design specification in VarDV)
- **size_t numDiscreteDesRangeVars**
  - number of discrete design variables defined by an integer range (from the discrete_design_range specification in VarDV)
- **size_t numDiscreteDesSetIntVars**
  - number of discrete design variables defined by a set of integers (from the discrete_design_set_integer specification in VarDV)
- **size_t numDiscreteDesSetRealVars**
  - number of discrete design variables defined by a set of reals (from the discrete_design_set_real specification in VarDV)
- **size_t numNormalUncVars**
  - number of normal uncertain variables (from the normal_uncertain specification in VarAUV)
- **size_t numLognormalUncVars**
  - number of lognormal uncertain variables (from the lognormal_uncertain specification in VarAUV)
- **size_t numUniformUncVars**
  - number of uniform uncertain variables (from the uniform_uncertain specification in VarAUV)
- **size_t numLoguniformUncVars**
  - number of loguniform uncertain variables (from the loguniform_uncertain specification in VarAUV)
- **size_t numTriangularUncVars**
  - number of triangular uncertain variables (from the triangular_uncertain specification in VarAUV)
- **size_t numExponentialUncVars**
  - number of exponential uncertain variables (from the exponential_uncertain specification in VarAUV)
- **size_t numBetaUncVars**
  - number of beta uncertain variables (from the beta_uncertain specification in VarAUV)
- **size_t numGammaUncVars**
  - number of gamma uncertain variables (from the gamma_uncertain specification in VarAUV)
- **size_t numGumbelUncVars**
  - number of gumbel uncertain variables (from the gumbel_uncertain specification in VarAUV)
- **size_t numFrechetUncVars**
  - number of frechet uncertain variables (from the frechet_uncertain specification in VarAUV)
- **size_t numWeibullUncVars**
number of weibull uncertain variables (from the \texttt{weibull\_uncertain} specification in \texttt{VarAUV})

- \texttt{size\_t numHistogramBinUncVars}
  number of histogram bin uncertain variables (from the \texttt{histogram\_bin\_uncertain} specification in \texttt{VarAUV})

- \texttt{size\_t numPoissonUncVars}
  number of Poisson uncertain variables (from the \texttt{poisson\_uncertain} specification in \texttt{VarAUV})

- \texttt{size\_t numBinomialUncVars}
  number of binomial uncertain variables (from the \texttt{binomial\_uncertain} specification in \texttt{VarAUV})

- \texttt{size\_t numNegBinomialUncVars}
  number of negative binomial uncertain variables (from the \texttt{negative\_binomial\_uncertain} specification in \texttt{VarAUV})

- \texttt{size\_t numGeometricUncVars}
  number of geometric uncertain variables (from the \texttt{geometric\_uncertain} specification in \texttt{VarAUV})

- \texttt{size\_t numHyperGeomUncVars}
  number of hypergeometric uncertain variables (from the \texttt{hypergeometric\_uncertain} specification in \texttt{VarAUV})

- \texttt{size\_t numHistogramPtUncVars}
  number of histogram point uncertain variables (from the \texttt{histogram\_point\_uncertain} specification in \texttt{VarAUV})

- \texttt{size\_t numContinuousIntervalUncVars}
  number of continuous epistemic interval uncertain variables (from the \texttt{continuous\_interval\_uncertain} specification in \texttt{VarEUV})

- \texttt{size\_t numDiscreteIntervalUncVars}
  number of discrete epistemic interval uncertain variables (from the \texttt{discrete\_interval\_uncertain} specification in \texttt{VarEUV})

- \texttt{size\_t numDiscreteUncSetIntVars}
  number of discrete epistemic uncertain integer set variables (from the \texttt{discrete\_uncertain\_set\_integer} specification in \texttt{VarEUV})

- \texttt{size\_t numDiscreteUncSetRealVars}
  number of discrete epistemic uncertain real set variables (from the \texttt{discrete\_uncertain\_set\_real} specification in \texttt{VarEUV})

- \texttt{size\_t numContinuousStateVars}
  number of continuous state variables (from the \texttt{continuous\_state} specification in \texttt{VarSV})

- \texttt{size\_t numDiscreteStateRangeVars}
  number of discrete state variables defined by an integer range (from the \texttt{discrete\_state\_range} specification in \texttt{VarDV})

- \texttt{size\_t numDiscreteStateSetIntVars}
  number of discrete state variables defined by a set of integers (from the \texttt{discrete\_state\_set\_integer} specification in \texttt{VarDV})

- \texttt{size\_t numDiscreteStateSetRealVars}
  number of discrete state variables defined by a set of reals (from the \texttt{discrete\_state\_set\_real} specification in \texttt{VarDV})

- \texttt{RealVector continuousDesignVars}
  initial values for the continuous design variables array (from the \texttt{continuous\_design\ initial\_point} specification in \texttt{VarDV})

- \texttt{RealVector continuousDesignLowerBnds}
CHAPTER 13. CLASS DOCUMENTATION

lower bounds array for the continuous design variables (from the continuous_design lower_bounds specification in VarDV)

- RealVector continuousDesignUpperBnds
  upper bounds array for the continuous design variables (from the continuous_design upper_bounds specification in VarDV)

- StringArray continuousDesignScaleTypes
  scale types array for the continuous design variables (from the continuous_design scale_types specification in VarDV)

- RealVector continuousDesignScales
  scales array for the continuous design variables (from the continuous_design scales specification in VarDV)

- IntVector discreteDesignRangeVars
  initial values for the discrete design variables defined by an integer range (from the discrete_design_range initial_point specification in VarDV)

- IntVector discreteDesignRangeLowerBnds
  lower bounds array for the discrete design variables defined by an integer range (from the discrete_design_range lower_bounds specification in VarDV)

- IntVector discreteDesignRangeUpperBnds
  upper bounds array for the discrete design variables defined by an integer range (from the discrete_design_range upper_bounds specification in VarDV)

- IntVector discreteDesignSetIntVars
  initial values for the discrete design variables defined by an integer set (from the discrete_design_set_integer initial_point specification in VarDV)

- RealVector discreteDesignSetRealVars
  initial values for the discrete design variables defined by a real set (from the discrete_design_set_real initial_point specification in VarDV)

- IntSetArray discreteDesignSetInt
  complete set of admissible values for each of the discrete design variables defined by an integer set (from the discrete_design_set_integer_set_values specification in VarDV)

- RealSetArray discreteDesignSetReal
  complete set of admissible values for each of the discrete design variables defined by a real set (from the discrete_design_set_real_set_values specification in VarDV)

- StringArray continuousDesignLabels
  labels array for the continuous design variables (from the continuous_design descriptors specification in VarDV)

- StringArray discreteDesignRangeLabels
  labels array for the discrete design variables defined by an integer range (from the discrete_design_range descriptors specification in VarDV)

- StringArray discreteDesignSetIntLabels
  labels array for the discrete design variables defined by an integer set (from the discrete_design_range descriptors specification in VarDV)

- StringArray discreteDesignSetRealLabels
  labels array for the discrete design variables defined by a real set (from the discrete_design_range descriptors specification in VarDV)

- RealVector normalUncMeans
  means of the normal uncertain variables (from the means specification in VarCAUV_Normal)
• RealVector normalUncStdDevs
  standard deviations of the normal uncertain variables (from the std.deviation specification in VarCAUV_Normal)
• RealVector normalUncLowerBnds
  distribution lower bounds for the normal uncertain variables (from the lower.bounds specification in VarCAUV_Normal)
• RealVector normalUncUpperBnds
  distribution upper bounds for the normal uncertain variables (from the upper.bounds specification in VarCAUV_Normal)
• RealVector normalUncVars
  initial values of the normal uncertain variables (from the initial.point specification in VarCAUV_Normal)
• RealVector lognormalUncLambdas
  lambdas (means of the corresponding normals) of the lognormal uncertain variables (from the lambdas specification in VarCAUV_Lognormal)
• RealVector lognormalUncZetas
  zetas (standard deviations of the corresponding normals) of the lognormal uncertain variables (from the zetas specification in VarCAUV_Lognormal)
• RealVector lognormalUncMeans
  means of the lognormal uncertain variables (from the means specification in VarCAUV_Lognormal)
• RealVector lognormalUncStdDevs
  standard deviations of the lognormal uncertain variables (from the std.deviation specification in VarCAUV_Lognormal)
• RealVector lognormalUncErrFacts
  error factors for the lognormal uncertain variables (from the error.factors specification in VarCAUV_Lognormal)
• RealVector lognormalUncLowerBnds
  distribution lower bounds for the lognormal uncertain variables (from the lower.bounds specification in VarCAUV_Lognormal)
• RealVector lognormalUncUpperBnds
  distribution upper bounds for the lognormal uncertain variables (from the upper.bounds specification in VarCAUV_Lognormal)
• RealVector lognormalUncVars
  initial values of the lognormal uncertain variables (from the initial.point specification in VarCAUV_Lognormal)
• RealVector uniformUncLowerBnds
  distribution lower bounds for the uniform uncertain variables (from the lower.bounds specification in VarCAUV_Uniform)
• RealVector uniformUncUpperBnds
  distribution upper bounds for the uniform uncertain variables (from the upper.bounds specification in VarCAUV_Uniform)
• RealVector uniformUncVars
  initial values of the uniform uncertain variables (from the initial.point specification in VarCAUV_Uniform)
• RealVector loguniformUncLowerBnds
  distribution lower bounds for the loguniform uncertain variables (from the lower.bounds specification in VarCAUV_Loguniform)
• RealVector loguniformUncUpperBnds
  distribution upper bounds for the loguniform uncertain variables (from the upper.bounds specification in VarCAUV_Loguniform)
• RealVector loguniformUncVars
  initial values of the loguniform uncertain variables (from the initial_point specification in VarCAUV_Loguniform)
• RealVector triangularUncModes
  modes of the triangular uncertain variables (from the modes specification in VarCAUV_Triangular)
• RealVector triangularUncLowerBnds
  distribution lower bounds for the triangular uncertain variables (from the lower_bounds specification in VarCAUV_Triangular)
• RealVector triangularUncUpperBnds
  distribution upper bounds for the triangular uncertain variables (from the upper_bounds specification in VarCAUV_Triangular)
• RealVector triangularUncVars
  initial values of the triangular uncertain variables (from the initial_point specification in VarCAUV_Triangular)
• RealVector exponentialUncBetas
  beta factors for the exponential uncertain variables (from the betas specification in VarCAUV_Exponential)
• RealVector exponentialUncVars
  initial values of the exponential uncertain variables (from the initial_point specification in VarCAUV_Exponential)
• RealVector betaUncAlphas
  alpha factors for the beta uncertain variables (from the means specification in VarCAUV_Beta)
• RealVector betaUncBetas
  beta factors for the beta uncertain variables (from the std_deviations specification in VarCAUV_Beta)
• RealVector betaUncLowerBnds
  distribution lower bounds for the beta uncertain variables (from the lower_bounds specification in VarCAUV_Beta)
• RealVector betaUncUpperBnds
  distribution upper bounds for the beta uncertain variables (from the upper_bounds specification in VarCAUV_Beta)
• RealVector betaUncVars
  initial values of the beta uncertain variables (from the initial_point specification in VarCAUV_Beta)
• RealVector gammaUncAlphas
  alpha factors for the gamma uncertain variables (from the alphas specification in VarCAUV_Gamma)
• RealVector gammaUncBetas
  beta factors for the gamma uncertain variables (from the betas specification in VarCAUV_Gamma)
• RealVector gammaUncVars
  initial values of the gamma uncertain variables (from the initial_point specification in VarCAUV_Gamma)
• RealVector gumbelUncAlphas
  alpha factors for the gumbel uncertain variables (from the alphas specification in VarCAUV_Gumbel)
• RealVector gumbelUncBetas
  beta factors for the gumbel uncertain variables (from the betas specification in VarCAUV_Gumbel)
• RealVector gumbelUncVars
  initial values of the gumbel uncertain variables (from the initial_point specification in VarCAUV_Gumbel)
• RealVector frechetUncAlphas
  alpha factors for the frechet uncertain variables (from the alphas specification in VarCAUV_Frechet)
• RealVector frechetUncBetas
beta factors for the frechet uncertain variables (from the `betas` specification in `VarCAUV_Frechet`)
- **RealVector** `frechetUncVars`
  - initial values of the frechet uncertain variables (from the `initial_point` specification in `VarCAUV_Frechet`)
- **RealVector** `weibullUncAlphas`
  - alpha factors for the weibull uncertain variables (from the `alphas` specification in `VarCAUV_Weibull`)
- **RealVector** `weibullUncBetas`
  - beta factors for the weibull uncertain variables (from the `betas` specification in `VarCAUV_Weibull`)
- **RealVector** `weibullUncVars`
  - initial values of the weibull uncertain variables (from the `initial_point` specification in `VarCAUV_Weibull`)
- **RealVectorArray** `histogramUncBinPairs`
  - an array containing a vector of (x,c) pairs for each bin-based histogram uncertain variable (see continuous linear histogram in LHS manual; from the `histogram_bin_uncertain` specification in `VarCAUV_Bin_Histogram`). (x,y) ordinate specifications are converted to (x,c) counts within NIDR.
- **RealVector** `histogramBinUncVars`
  - initial values of the histogram bin uncertain variables (from the `initial_point` specification in `VarCAUV_Bin_Histogram`)
- **RealVector** `poissonUncLambdas`
  - lambdas (rate parameter) for the poisson uncertain variables (from the `lambdas` specification in `VarDAUV_Poisson`)
- **IntVector** `poissonUncVars`
  - initial values of the poisson uncertain variables (from the `initial_point` specification in `VarDAUV_Poisson`)
- **RealVector** `binomialUncProbPerTrial`
  - probabilities per each trial (p) for the binomial uncertain variables from the `prob_per_trial` specification in `VarDAUV_Binomial`
- **IntVector** `binomialUncNumTrials`
  - Number of trials (N) for the binomial uncertain variables from the `num_trials` specification in `VarDAUV_Binomial`
- **IntVector** `binomialUncVars`
  - initial values of the binomial uncertain variables (from the `initial_point` specification in `VarDAUV_Binomial`)
- **RealVector** `negBinomialUncProbPerTrial`
  - probabilities per each trial (p) for the negative binomial uncertain variables from the `prob_per_trial` specification in `VarDAUV_Negative_Binomial`
- **IntVector** `negBinomialUncNumTrials`
  - Number of trials (N) for the negative binomial uncertain variables from the `num_trials` specification in `VarDAUV_Negative_Binomial`
- **IntVector** `negBinomialUncVars`
  - initial values of the negative binomial uncertain variables (from the `initial_point` specification in `VarDAUV_Negative_Binomial`)
- **RealVector** `geometricUncProbPerTrial`
  - probabilities per each trial (p) for the geometric uncertain variables from the `prob_per_trial` specification in `VarDAUV_Geometric`
- **IntVector** `geometricUncVars`
  - initial values of the geometric uncertain variables (from the `initial_point` specification in `VarDAUV_Geometric`)
- **IntVector** `hyperGeomUncTotalPop`
Size of total populations (N) for the hypergeometric uncertain variables from the total_population specification in VarDAUV_Hypergeometric)

- IntVector hyperGeomUncSelectedPop
  Size of selected populations for the hypergeometric uncertain variables from the selected_population specification in VarDAUV_Hypergeometric)

- IntVector hyperGeomUncNumDrawn
  Number failed in the selected populations for the hypergeometric variables from the num_drawn specification in VarDAUV_Hypergeometric)

- IntVector hyperGeomUncVars
  initial values of the hypergeometric uncertain variables (from the initial_point specification in VarDAUV_Hypergeometric)

- RealVectorArray histogramUncPointPairs
  an array containing a vector of (x,c) pairs for each point-based histogram uncertain variable (see discrete_histogram in LHS manual; from the histogram_point_uncertain specification in VarDAUV_Point_Histogram)

- RealVector histogramPointUncVars
  initial values of the histogram point uncertain variables (from the initial_point specification in VarDAUV_Point_Histogram)

- RealSymMatrix uncertainCorrelations
  correlation matrix for all uncertain variables (from the uncertain_correlation_matrix specification in VarAUV_Correlations). This matrix specifies rank correlations for LHS sampling and correlation coefficients (rho_{ij} = normalized covariance matrix) for other methods.

- RealVectorArray continuousIntervalUncBasicProbs
  Probability values per interval cell per epistemic interval uncertain variable (from the continuous_interval_uncertain_interval_probs specification in VarCEUV_Interval)

- RealVectorArray continuousIntervalUncLowerBounds
  lower bounds defining cells for each epistemic interval uncertain variable (from the continuous_interval_uncertain_lower_bounds specification in VarCEUV_Interval)

- RealVectorArray continuousIntervalUncUpperBounds
  upper bounds defining cells for each epistemic interval uncertain variable (from the continuous_interval_uncertain_upper_bounds specification in VarCEUV_Interval)

- RealVector continuousIntervalUncVars
  initial values of the continuous interval uncertain variables (from the initial_point specification in VarCEUV_Interval)

- RealVectorArray discreteIntervalUncBasicProbs
  Probability values per interval cell per epistemic interval uncertain variable (from the discrete_interval_uncertain_interval_probs specification in VarDIUV)

- IntVectorArray discreteIntervalUncLowerBounds
  lower bounds defining cells for each epistemic interval uncertain variable (from the discrete_interval_uncertain_lower_bounds specification in VarDIUV)

- IntVectorArray discreteIntervalUncUpperBounds
  upper bounds defining cells for each epistemic interval uncertain variable (from the discrete_interval_uncertain_upper_bounds specification in VarDIUV)

- IntVector discreteIntervalUncVars
  initial values of the discrete interval uncertain variables (from the initial_point specification in VarDIUV)

- IntRealMapArray discreteUncSetIntValuesProbs
complete set of admissible values with associated basic probability assignments for each of the discrete epistemic uncertain variables defined by an integer set (from the discrete_uncertain_set_integer_set_values specification in VarDUSIV)

- IntVector discreteUncSetIntVars
  initial values of the discrete uncertain set integer variables (from the initial_point specification in VarDUSIV)

- RealRealMapArray discreteUncSetRealValuesProbs
  complete set of admissible values with associated basic probability assignments for each of the discrete epistemic uncertain variables defined by a real set (from the discrete_uncertain_set_real_set_values specification in VarDUSRV)

- RealVector discreteUncSetRealVars
  initial values of the discrete uncertain set real variables (from the initial_point specification in VarDUSRV)

- RealVector continuousStateVars
  initial values for the continuous state variables array (from the continuous_state initial_point specification in VarSV)

- RealVector continuousStateLowerBnds
  lower bounds array for the continuous state variables (from the continuous_state lower_bounds specification in VarSV)

- RealVector continuousStateUpperBnds
  upper bounds array for the continuous state variables (from the continuous_state upper_bounds specification in VarSV)

- IntVector discreteStateRangeVars
  initial values for the discrete state variables defined by an integer range (from the discrete_state_range initial_point specification in VarSV)

- IntVector discreteStateRangeLowerBnds
  lower bounds array for the discrete state variables defined by an integer range (from the discrete_state_range lower_bounds specification in VarSV)

- IntVector discreteStateRangeUpperBnds
  upper bounds array for the discrete state variables defined by an integer range (from the discrete_state_range upper_bounds specification in VarSV)

- IntVector discreteStateSetIntVars
  initial values for the discrete state variables defined by an integer set (from the discrete_state_set_integer initial_point specification in VarSV)

- RealVector discreteStateSetRealVars
  initial values for the discrete state variables defined by a real set (from the discrete_state_set_real initial_point specification in VarSV)

- IntSetArray discreteStateSetInt
  complete set of admissible values for each of the discrete state variables defined by an integer set (from the discrete_state_set_integer_set_values specification in VarSV)

- RealSetArray discreteStateSetReal
  complete set of admissible values for each of the discrete state variables defined by a real set (from the discrete_state_set_real_set_values specification in VarSV)

- StringArray continuousStateLabels
  labels array for the continuous state variables (from the continuous_state descriptors specification in VarSV)

- StringArray discreteStateRangeLabels
labels array for the discrete state variables defined by an integer range (from the `discrete_state_range` descriptors specification in `VarSV`)

- **StringArray** `discreteStateSetIntLabels`
  labels array for the discrete state variables defined by an integer set (from the `discrete_state_range` descriptors specification in `VarSV`)

- **StringArray** `discreteStateSetRealLabels`
  labels array for the discrete state variables defined by a real set (from the `discrete_state_range` descriptors specification in `VarSV`)

- **IntVector** `discreteDesignSetIntLowerBnds`
  discrete design integer set lower bounds inferred from set values

- **IntVector** `discreteDesignSetIntUpperBnds`
  discrete design integer set upper bounds inferred from set values

- **RealVector** `discreteDesignSetRealLowerBnds`
  discrete design real set lower bounds inferred from set values

- **RealVector** `discreteDesignSetRealUpperBnds`
  discrete design real set upper bounds inferred from set values

- **RealVector** `continuousAleatoryUncVars`
  array of values for all continuous aleatory uncertain variables

- **RealVector** `continuousAleatoryUncLowerBnds`
  distribution lower bounds for all continuous aleatory uncertain variables (collected from `nvv_lower_bounds`, `lnvv_lower_bounds`, `uuv_lower_bounds`, `luuv_lower_bounds`, `tuv_lower_bounds`, and `buv_lower_bounds` specifications in `VarAUV`, and derived for gamma, gumbel, frechet, weibull and histogram bin specifications)

- **RealVector** `continuousAleatoryUncUpperBnds`
  distribution upper bounds for all continuous aleatory uncertain variables (collected from `nvv_upper_bounds`, `lnvv_upper_bounds`, `uuv_upper_bounds`, `luuv_upper_bounds`, `tuv_lower_bounds`, and `buv_upper_bounds` specifications in `VarAUV`, and derived for gamma, gumbel, frechet, weibull and histogram bin specifications)

- **StringArray** `continuousAleatoryUncLabels`
  labels for all continuous aleatory uncertain variables (collected from `nvv_descriptors`, `lnvv_descriptors`, `uuv_descriptors`, `luuv_descriptors`, `tuv_descriptors`, `buv_descriptors`, `gauv_descriptors`, `guuv_descriptors`, `fuv_descriptors`, `wuv_descriptors`, and `hbuv_descriptors` specifications in `VarAUV`)

- **IntVector** `discreteIntAleatoryUncVars`
  array of values for all discrete integer aleatory uncertain variables

- **IntVector** `discreteIntAleatoryUncLowerBnds`
  distribution lower bounds for all discrete integer aleatory uncertain variables

- **IntVector** `discreteIntAleatoryUncUpperBnds`
  distribution upper bounds for all discrete integer aleatory uncertain variables

- **StringArray** `discreteIntAleatoryUncLabels`
  labels for all discrete integer aleatory uncertain variables

- **RealVector** `discreteRealAleatoryUncVars`
  array of values for all discrete real aleatory uncertain variables

- **RealVector** `discreteRealAleatoryUncLowerBnds`
  distribution lower bounds for all discrete real aleatory uncertain variables
• RealVector `discreteRealAleatoryUncUpperBnds`
  distribution upper bounds for all discrete real aleatory uncertain variables
• StringArray `discreteRealAleatoryUncLabels`
  labels for all discrete real aleatory uncertain variables
• RealVector `continuousEpistemicUncVars`
  array of values for all continuous epistemic uncertain variables
• RealVector `continuousEpistemicUncLowerBnds`
  distribution lower bounds for all continuous epistemic uncertain variables
• RealVector `continuousEpistemicUncUpperBnds`
  distribution upper bounds for all continuous epistemic uncertain variables
• StringArray `continuousEpistemicUncLabels`
  labels for all continuous epistemic uncertain variables
• IntVector `discreteIntEpistemicUncVars`
  array of values for all discrete integer epistemic uncertain variables
• IntVector `discreteIntEpistemicUncLowerBnds`
  distribution lower bounds for all discrete integer epistemic uncertain variables
• IntVector `discreteIntEpistemicUncUpperBnds`
  distribution upper bounds for all discrete integer epistemic uncertain variables
• StringArray `discreteIntEpistemicUncLabels`
  labels for all discrete integer epistemic uncertain variables
• RealVector `discreteRealEpistemicUncVars`
  array of values for all discrete real epistemic uncertain variables
• RealVector `discreteRealEpistemicUncLowerBnds`
  distribution lower bounds for all discrete real epistemic uncertain variables
• RealVector `discreteRealEpistemicUncUpperBnds`
  distribution upper bounds for all discrete real epistemic uncertain variables
• StringArray `discreteRealEpistemicUncLabels`
  labels for all discrete real epistemic uncertain variables
• IntVector `discreteStateSetIntLowerBnds`
  discrete state integer set lower bounds inferred from set values
• IntVector `discreteStateSetIntUpperBnds`
  discrete state integer set upper bounds inferred from set values
• RealVector `discreteStateSetRealLowerBnds`
  discrete state real set lower bounds inferred from set values
• RealVector `discreteStateSetRealUpperBnds`
  discrete state real set upper bounds inferred from set values
CHAPTER 13. CLASS DOCUMENTATION

Private Member Functions

- DataVariablesRep()
  - default constructor
- ~DataVariablesRep()
  - destructor
- void write (std::ostream &s) const
  - write a DataVariablesRep object to an std::ostream
- void read (MPIUnpackBuffer &s)
  - read a DataVariablesRep object from a packed MPI buffer
- void write (MPIPackBuffer &s) const
  - write a DataVariablesRep object to a packed MPI buffer

Private Attributes

- int referenceCount
  - number of handle objects sharing dataVarsRep

Friends

- class DataVariables
  - the handle class can access attributes of the body class directly

13.29.1 Detailed Description

Body class for variables specification data.

The DataVariablesRep class is used to contain the data from a variables keyword specification. Default values are managed in the DataVariablesRep constructor. Data is public to avoid maintaining set/get functions, but is still encapsulated within ProblemDescDB since ProblemDescDB::dataVariablesList is private.

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

- DataVariables.hpp
- DataVariables.cpp

13.30 DDACEDesignCompExp Class Reference

Wrapper class for the DDACE design of experiments library.

Inheritance diagram for DDACEDesignCompExp:

```
  Iterator
  
  Analyzer
  
  PStudyDACE
  
  DDACEDesignCompExp
```
Public Member Functions

- DDACEDesignCompExp (ProblemDescDB &problem_db, Model &model)
  primary constructor for building a standard DACE iterator
- DDACEDesignCompExp (Model &model, int samples, int symbols, int seed, unsigned short sampling_method)
  alternate constructor used for building approximations
- ~DDACEDesignCompExp ()
  destructor
- void pre_run ()
  pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori
- void extract_trends ()
  Mapping of the core_run() virtual function for the PStudy/DACE branch.
- void post_input ()
  read tabular data for post-run mode
- void post_run (std::ostream &s)
  post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way
- int num_samples () const
- void sampling_reset (int min_samples, bool all_data_flag, bool stats_flag)
  reset sampling iterator to use at least min_samples
- unsigned short sampling_scheme () const
  return sampling name
- void vary_pattern (bool pattern_flag)
  sets varyPattern in derived classes that support it
- void get_parameter_sets (Model &model)
  Returns one block of samples (ndim * num_samples)

Private Member Functions

- void compute_main_effects ()
  builds a DDaceMainEffects::OneWayANOVA if mainEffectsFlag is set
- void resolve_samples_symbols ()
  convenience function for resolving number of samples and number of symbols from input.

Static Private Member Functions

- static void copy_data (const std::vector< DDaceSamplePoint > &dspa, Real *ptr, const int ptr_len)
  copy DDACE point to RealVector
CHAPTER 13. CLASS DOCUMENTATION

Private Attributes

- unsigned short daceMethod
  oas, lhs, oa_lhs, random, box_behnken, central_composite, or grid
- int samplesSpec
  initial specification of number of samples
- int symbolsSpec
  initial specification of number of symbols
- int numSamples
  current number of samples to be evaluated
- int numSymbols
  current number of symbols to be used in generating the sample set (inversely related to number of replications)
- const int seedSpec
  the user seed specification for the random number generator (allows repeatable results)
- int randomSeed
  current seed for the random number generator
- bool allDataFlag
  flag which triggers the update of allVars/allResponses for use by Iterator::all_variables() and Iterator::all_responses()
- size_t numDACERuns
  counter for number of executions for this object
- bool varyPattern
  flag for continuing the random number sequence from a previous execution (e.g., for surrogate-based optimization) so that multiple executions are repeatable but not correlated.
- bool mainEffectsFlag
  flag which specifies main effects
- std::vector< std::vector<int> > symbolMapping
  mapping of symbols for main effects calculations

Additional Inherited Members

13.30.1 Detailed Description

Wrapper class for the DDACE design of experiments library.

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

13.30.2 Constructor & Destructor Documentation

DDACEDesignCompExp ( ProblemDescDB & problem_db, Model & model )

primary constructor for building a standard DACE iterator

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

References Dakota::abort_handler(), DDACEDesignCompExp::daceMethod, DDACEDesignCompExp::mainEffectsFlag, Iterator::maxEvalConcurrency, Analyzer::numContinuousVars, and DDACEDesignCompExp::numSamples.
DDACEDesignCompExp (Model & model, int samples, int symbols, int seed, unsigned short sampling method)

alternate constructor used for building approximations

This alternate constructor is used for instantiations on-the-fly, using only the incoming data. No problem description database queries are used.

References Iterator::maxEvalConcurrency, DDACEDesignCompExp::numSamples, and DDACEDesignCompExp::resolve_samples_symbols().

13.30.3 Member Function Documentation

void pre_run ( ) [virtual]

pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori

pre-run phase, which a derived iterator may optionally reimplement; when not present, pre-run is likely integrated into the derived run function. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s pre_run(), if implemented, typically before performing its own implementation steps.

Reimplemented from Iterator.

References DDACEDesignCompExp::get_parameter_sets(), Iterator::iteratedModel, and PStudyDACE::var-BasedDecompFlag.

void post_run ( std::ostream & s ) [virtual]

post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way

Post-run phase, which a derived iterator may optionally reimplement; when not present, post-run is likely integrated into run. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s post_run(), typically after performing its own implementation steps.

Reimplemented from Analyzer.

References Analyzer::allResponses, Analyzer::allSamples, SensAnalysisGlobal::compute_correlations(), DDACEDesignCompExp::compute_main_effects(), DDACEDesignCompExp::mainEffectsFlag, Analyzer::post_run(), PStudyDACE::pStudyDACEsensGlobal, Iterator::subIteratorFlag, and PStudyDACE::varBasedDecompFlag.

int num_samples ( ) const [inline], [virtual]

Return current number of evaluation points. Since the calculation of samples, collocation points, etc. might be costly, provide a default implementation here that backs out from the maxEvalConcurrency.

Reimplemented from Analyzer.

References DDACEDesignCompExp::numSamples.

void resolve_samples_symbols ( ) [private]

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

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

References Dakota::abort_handler(), DDACEDesignCompExp::daceMethod, Analyzer::numContinuousVars, DDACEDesignCompExp::numSamples, and DDACEDesignCompExp::numSymbols.

Referenced by DDACEDesignCompExp::DDACEDesignCompExp(), and DDACEDesignCompExp::get_parameter_sets().
void copy_data ( const std::vector<DDaceSamplePoint> &dspa, Real *ptr, const int ptr_len )

[static], [private]

copy DDACE point to RealVector

copy DDACE point array to RealVectorArray copy DDACE point array to Real*

References Dakota::abort_handler().
Referenced by DDACEDesignCompExp::get_parameter_sets().
The documentation for this class was generated from the following files:

- DDACEDesignCompExp.hpp
- DDACEDesignCompExp.cpp

13.31 DirectApplicInterface Class Reference

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

Inheritance diagram for DirectApplicInterface:

Public Member Functions

- DirectApplicInterface (const ProblemDescDB &problem_db)
  constructor
- ~DirectApplicInterface ()
  destructor
- void derived_map (const Variables &vars, const ActiveSet &set, Response &response, int fn_eval_id)
  Called by map() and other functions to execute the simulation in synchronous mode. The portion of performing an evaluation that is specific to a derived class.
- void derived_map_asynch (const ParamResponsePair &pair)
  Called by map() and other functions to execute the simulation in asynchronous mode. The portion of performing an asynchronous evaluation that is specific to a derived class.
- void wait_local_evaluations (PRPQueue &prp_queue)
  For asynchronous function evaluations, this method is used to detect completion of jobs and process their results. It provides the processing code that is specific to derived classes. This version waits for at least one completion.
- void test_local_evaluations (PRPQueue &prp_queue)
  For asynchronous function evaluations, this method is used to detect completion of jobs and process their results. It provides the processing code that is specific to derived classes. This version is nonblocking and will return without any completions if none are immediately available.
- int synchronous_local_analysis (int analysis_id)
- const StringArray &analysis_drivers () const
  retrieve the analysis drivers specification for application interfaces
- void init_communicators_checks (int max_eval_concurrency)
- void set_communicators_checks (int max_eval_concurrency)
Protected Member Functions

- virtual int derived_map_if (const Dakota::String &if_name)
  
  execute the input filter portion of a direct evaluation invocation

- virtual int derived_map_ac (const Dakota::String &ac_name)
  
  execute an analysis code portion of a direct evaluation invocation

- virtual int derived_map_of (const Dakota::String &of_name)
  
  execute the output filter portion of a direct evaluation invocation

- void set_local_data (const Variables &vars, const ActiveSet &set)

  convenience function for local test simulators which sets per-evaluation variable and active set attributes

- void set_local_data (const Response &response)

  convenience function for local test simulators which sets per-evaluation response attributes

- void set_local_data (const Variables &vars, const ActiveSet &set, const Response &response)

  convenience function for local test simulators which sets per-evaluation variable, active set, and response attributes

- void overlay_response (Response &response)

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

Protected Attributes

- String iFilterName
  
  name of the direct function input filter

- String oFilterName
  
  name of the direct function output filter

- driver_t iFilterType

  enum type of the direct function input filter

- driver_t oFilterType

  enum type of the direct function output filter

- bool gradFlag

  signals use of fnGrads in direct simulator functions

- bool hessFlag

  signals use of fnHessians in direct simulator functions

- size_t numFns

  number of functions in fnVals

- size_t numVars

  total number of continuous and discrete variables

- size_t numACV

  total number of continuous variables

- size_t numADIV

  total number of discrete integer variables

- size_t numADRV

  total number of discrete real variables

- size_t numDerivVars

  number of active derivative variables
- unsigned short localDataView
  
  bit-wise record of which local data views are active; see enum local_data_t

- RealVector xC
  
  continuous variables used within direct simulator fns

- IntVector xDI
  
  discrete int variables used within direct simulator fns

- RealVector xDR
  
  discrete real variables used within direct simulator fns

- StringMultiArray xCLabels
  
  continuous variable labels

- StringMultiArray xDILabels
  
  discrete integer variable labels

- StringMultiArray xDRLabels
  
  discrete real variable labels

- std::map<String, var_t> varTypeMap
  
  map from variable label to enum

- std::map<String, driver_t> driverTypeMap
  
  map from driver name to enum

- std::map<var_t, Real> xCM
  
  map from var_t enum to continuous value

- std::map<var_t, int> xDIM
  
  map from var_t enum to discrete int value

- std::map<var_t, Real> xDRM
  
  map from var_t enum to discrete real value

- std::vector<var_t> varTypeDVV
  
  var_t enumerations corresponding to DVV components

- std::vector<var_t> xCMLabels
  
  var_t enumerations corresponding to continuous variable labels

- std::vector<var_t> xDIMLabels
  
  var_t enumerations corresponding to discrete integer variable labels

- std::vector<var_t> xDRMLabels
  
  var_t enumerations corresponding to discrete real variable labels

- ShortArray directFnASV
  
  class scope active set vector

- SizetArray directFnDVV
  
  class scope derivative variables vector

- RealVector fnVals

  response fn values within direct simulator fns

- RealMatrix fnGrads

  response fn gradients w/ direct simulator fns

- RealSymMatrixArray fnHessians

  response fn Hessians within direct fns

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

- std::vector<driver_t> analysisDriverTypes
  conversion of analysisDrivers to driver_t
- size_t analysisDriverIndex
  the index of the active analysis driver within analysisDrivers
- String2DArray analysisComponents
  the set of optional analysis components used by the analysis drivers (from the analysis_components interface specification)

13.31.1 Detailed Description

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

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

13.31.2 Member Function Documentation

int synchronous_local_analysis ( int analysis_id ) [inline], [virtual]

This code provides the derived function used by ApplicationInterface::serve_analyses_synch().
Reimplemented from ApplicationInterface.
References DirectApplicInterface::analysisDriverIndex, DirectApplicInterface::analysisDrivers, and DirectApplicInterface::derived_map_ac().

void init_communicators_checks ( int max_eval_concurrency ) [inline], [virtual]

Process init issues as warnings since some contexts (e.g., HierarchSurrModel) initialize more configurations than will be used and DirectApplicInterface allows override by derived plug-ins.
Reimplemented from ApplicationInterface.
References ApplicationInterface::check_asynchronous(), and ApplicationInterface::check_multiprocessor_asynchronous().

void set_communicators_checks ( int max_eval_concurrency ) [inline], [virtual]

Process run-time issues as hard errors.
Reimplemented from ApplicationInterface.
Reimplemented in SerialDirectApplicInterface, and ParallelDirectApplicInterface.
References Dakota::abort_handler(), ApplicationInterface::check_asynchronous(), and ApplicationInterface::check_multiprocessor_asynchronous().

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:

```cpp
// API declaration
int sim(const Variables& vars, const ActiveSets& set, Response& response);
// use of API within derived_map_ac()
if (ac_name == "sim")
  fail_code = sim(directFnVars, directFnActSet, directFnResponse);
```
Reimplemented in SerialDirectApplicInterface, ParallelDirectApplicInterface, MatlabInterface, PythonInterface, TestDriverInterface, and ScilabInterface.
References Dakota::abort_handler(), and ApplicationInterface::analysisServerId.
Referenced by DirectApplicInterface::derived_map(), and DirectApplicInterface::synchronous_local_analysis().
The documentation for this class was generated from the following files:

- DirectApplicInterface.hpp
- DirectApplicInterface.cpp

13.32 DiscrepancyCorrection Class Reference

Base class for discrepancy corrections.

Public Member Functions

- **DiscrepancyCorrection ()**
  *default constructor*
- **DiscrepancyCorrection (Model &surr_model, const IntSet &surr_fn_indices, short corr_type, short corr_order)**
  *standard constructor*
- **DiscrepancyCorrection (const IntSet &surr_fn_indices, size_t num_fns, size_t num-vars, short corr_type, short corr_order)**
  *alternate constructor*
- **~DiscrepancyCorrection ()**
  *destructor*
- void initialize (Model &surr_model, const IntSet &surr_fn_indices, short corr_type, short corr_order)
  *initialize the DiscrepancyCorrection data*
- void initialize (const IntSet &surr_fn_indices, size_t num_fns, size_t num_vars, short corr_type, short corr_order)
  *initialize the DiscrepancyCorrection data*
- **void compute (const Variables &vars, const Response &truth_response, const Response &approx_response, bool quiet_flag=false)**
  *compute the correction required to bring approx_response into agreement with truth_response and store in \{add,mul\} Corrections*
- void compute (const Response &truth_response, const Response &approx_response, Response &discrepancy_response, bool quiet_flag=false)
  *compute the correction required to bring approx_response into agreement with truth_response and store in discrepancy_response*
- void apply (const Variables &vars, Response &approx_response, bool quiet_flag=false)
  *apply the correction computed in compute() to approx_response*
- bool active () const
  *indicates an active correction via non-empty correctionType*
- **short correction_type () const**
  *return correctionType*
- short correction_order () const
  *return correctionOrder*
- short data_order () const
return dataOrder

- bool computed () const
  return correctionComputed

**Protected Attributes**

- IntSet surrogateFnIndices
  for mixed response sets, this array specifies the response function subset that is approximated

- short correctionType
  approximation correction approach to be used: NO_CORRECTION, ADDITIVE_CORRECTION, MULTIPLICATIVE_CORRECTION, or COMBINED_CORRECTION.

- short correctionOrder
  approximation correction order to be used: 0, 1, or 2

- short dataOrder
  order of correction data in 3-bit format: overlay of 1 (value), 2 (gradient), and 4 (Hessian)

- bool correctionComputed
  flag indicating whether or not a correction has been computed and is available for application

- size_t numFns
  total number of response functions (of which surrogateFnIndices may define a subset)

- size_t numVars
  number of continuous variables active in the correction

**Private Member Functions**

- void initialize_corrections ()
  internal convenience function shared by overloaded initialize() variants

- bool check_scaling (const RealVector &truth_fns, const RealVector &approx_fns)
  define badScalingFlag

- void compute_additive (const Response &truth_response, const Response &approx_response, int index, Real &discrep_fn, RealVector &discrep_grad, RealSymMatrix &discrep_hess)
  internal convenience function for computing additive corrections between truth and approximate responses

- void compute_multiplicative (const Response &truth_response, const Response &approx_response, int index, Real &discrep_fn, RealVector &discrep_grad, RealSymMatrix &discrep_hess)
  internal convenience function for computing multiplicative corrections between truth and approximate responses

- void apply_additive (const Variables &vars, Response &approx_response)
  internal convenience function for applying additive corrections to an approximate response

- void apply_multiplicative (const Variables &vars, Response &approx_response)
  internal convenience function for applying multiplicative corrections to an approximate response

- void apply_additive (const Variables &vars, RealVector &approx_fns)
  internal convenience function for applying additive corrections to a set of response functions

- void apply_multiplicative (const Variables &vars, RealVector &approx_fns)
  internal convenience function for applying multiplicative corrections to a set of response functions

- const Response & search_db (const Variables &search_vars, const ShortArray &search_asv)
  search data pairs for missing approximation data
Private Attributes

- **bool badScalingFlag**
  - flag used to indicate function values near zero for multiplicative corrections; triggers an automatic switch to additive corrections
- **bool computeAdditive**
  - flag indicating the need for additive correction calculations
- **bool computeMultiplicative**
  - flag indicating the need for multiplicative correction calculations
- **SharedApproxData sharedData**
  - data that is shared among all correction Approximations
- **std::vector< Approximation > addCorrections**
  - array of additive corrections; surrogate models of a model discrepancy function (formed from model differences)
- **std::vector< Approximation > multCorrections**
  - array of multiplicative corrections; surrogate models of a model discrepancy function (formed from model ratios)
- **Model surrModel**
  - shallow copy of the surrogate model instance as returned by Model::surrogate_model() (the DataFitSurrModel or HierarchSurrModel::lowFidelityModel instance)
- **RealVector combineFactors**
  - factors for combining additive and multiplicative corrections. Each factor is the weighting applied to the additive correction and 1.-factor is the weighting applied to the multiplicative correction. The factor value is determined by an additional requirement to match the high fidelity function value at the previous correction point (e.g., previous trust region center). This results in a multipoint correction instead of a strictly local correction.
- **Variables correctionPrevCenterPt**
  - copy of center point from the previous correction cycle
- **RealVector truthFnsCenter**
  - truth function values at the current correction point
- **RealVector approxFnsCenter**
  - Surrogate function values at the current correction point.
- **RealMatrix approxGradsCenter**
  - Surrogate gradient values at the current correction point.
- **RealVector truthFnsPrevCenter**
  - copy of truth function values at center of previous correction cycle
- **RealVector approxFnsPrevCenter**
  - copy of approximate function values at center of previous correction cycle

13.32.1 Detailed Description

Base class for discrepancy corrections.

The `DiscrepancyCorrection` class provides common functions for computing and applying corrections to approximations.
### 13.32.2 Member Function Documentation

```cpp
def void compute ( const Variables & vars, const Response & truth_response, const Response & approx_response, bool quiet_flag = false )
```

compute the correction required to bring `approx_response` into agreement with `truth_response` and store in `{add,mult}Corrections

Compute an additive or multiplicative correction that corrects the `approx_response` to have 0th-order consistency (matches values), 1st-order consistency (matches values and gradients), or 2nd-order consistency (matches values, gradients, and Hessians) with the `truth_response` at a single point (e.g., the center of a trust region). The 0th-order, 1st-order, and 2nd-order corrections use scalar values, linear scaling functions, and quadratic scaling functions, respectively, for each response function.

References `Response::active_set()`, `DiscrepancyCorrection::addCorrections`, `DiscrepancyCorrection::apply()`, `DiscrepancyCorrection::apply_additive()`, `DiscrepancyCorrection::apply_multiplicative()`, `DiscrepancyCorrection::approxFnsCenter`, `DiscrepancyCorrection::approxFnsPrevCenter`, `DiscrepancyCorrection::approxGradsCenter`, `DiscrepancyCorrection::badScalingFlag`, `DiscrepancyCorrection::check_scaling()`, `DiscrepancyCorrection::combineFactors`, `DiscrepancyCorrection::compute_additive()`, `DiscrepancyCorrection::compute_multiplicative()`, `DiscrepancyCorrection::computeAdditive`, `DiscrepancyCorrection::computeMultiplicative`, `Variables::continuous_variables()`, `Response::copy()`, `DiscrepancyCorrection::correctionComputed`, `DiscrepancyCorrection::correctionOrder`, `DiscrepancyCorrection::correctionPrevCenterPt`, `DiscrepancyCorrection::correctionType`, `DiscrepancyCorrection::dataOrder`, `Variables::discrete_int_variables()`, `Variables::discrete_real_variables()`, `Response::function_gradients()`, `Response::function_values()`, `Model::is_null()`, `DiscrepancyCorrection::multCorrections`, `DiscrepancyCorrection::numFns`, `DiscrepancyCorrection::numVars`, `ActiveSet::request_values()`, `DiscrepancyCorrection::sharedData`, `DiscrepancyCorrection::surrModel`, `DiscrepancyCorrection::surrogateFnIndices`, `DiscrepancyCorrection::truthFnsCenter`, and `DiscrepancyCorrection::truthFnsPrevCenter`.

Referenced by `HierarchSurrModel::derived_asynch_compute_response()`, `HierarchSurrModel::derived_compute_response()`, `DataFitSurrModel::derived_compute_response()`, `HierarchSurrModel::derived_synchronize()`, `DataFitSurrModel::derived_synchronize()`, `HierarchSurrModel::derived_synchronize_nowait()`, `DataFitSurrModel::derived_synchronize_nowait()`, and `SurrBasedLocalMinimizer::minimize_surrogates()`.

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

- DiscrepancyCorrection.hpp
- DiscrepancyCorrection.cpp

### 13.33 DOTOptimizer Class Reference

Wrapper class for the DOT optimization library.

Inheritance diagram for DOTOptimizer:
CHAPTER 13. CLASS DOCUMENTATION

Public Member Functions

- `DOTOptimizer (ProblemDescDB &problem_db, Model &model)`
  standard constructor
- `DOTOptimizer (const String &method_string, Model &model)`
  alternate constructor; construct without `ProblemDescDB`
- `~DOTOptimizer ()`
  destructor
- `void find_optimum ()`
  Used within the optimizer branch for computing the optimal solution. Redefines the run virtual function for the optimizer branch.

Protected Member Functions

- `void initialize_run ()`
  performs run-time set up

Private Member Functions

- `void initialize ()`
  Shared constructor code.
- `void allocate_workspace ()`
  Allocates workspace for the optimizer.
- `void allocate_constraints ()`
  Allocates constraint mappings.

Private Attributes

- `int dotInfo`
  INFO from DOT manual.
- `int dotFDSinfo`
  internal DOT parameter NGOTOZ
- `int dotMethod`
  METHOD from DOT manual.
- `int printControl`
  IPRINT from DOT manual (controls output verbosity)
- `RealArray realCntlParmArray`
  RPRM from DOT manual.
- `IntArray intCntlParmArray`
  IPRM from DOT manual.
- `RealVector designVars`
  array of design variable values passed to DOT
- `Real objFnValue`
  value of the objective function passed to DOT
- `RealVector constraintValues`
13.33. DOTOPTIMIZER CLASS REFERENCE

array of nonlinear constraint values passed to DOT

- `int realWorkSpaceSize`
  size of realWorkSpace
- `int intWorkSpaceSize`
  size of intWorkSpace
- `RealArray realWorkSpace`
  real work space for DOT
- `IntArray intWorkSpace`
  int work space for DOT
- `int numDotNlnConstr`
  total number of nonlinear constraints seen by DOT
- `int numDotLinConstr`
  total number of linear constraints seen by DOT
- `int numDotConstr`
  total number of linear and nonlinear constraints seen by DOT
- `SizetArray constraintMappingIndices`
  a container of indices for referencing the corresponding Response constraints used in computing the DOT constraints.
- `RealArray constraintMappingMultipliers`
  a container of multipliers for mapping the Response constraints to the DOT constraints.
- `RealArray constraintMappingOffsets`
  a container of offsets for mapping the Response constraints to the DOT constraints.

Additional Inherited Members

13.33.1 Detailed Description

Wrapper class for the DOT optimization library.

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

The user input mappings are as follows: max_iterations is mapped into DOT’s ITMAX parameter within its IPRM array, max_function_evaluations is implemented directly in the find_optimum() loop since there is no DOT parameter equivalent, convergence_tolerance is mapped into DOT’s DELOBJ parameter (the relative convergence tolerance) within its RPRM array, output verbosity is mapped into DOT’s IPRINT parameter within its function call parameter list (verbose: IPRINT = 7; quiet: IPRINT = 3), and optimization_type is mapped into DOT’s MINMAX parameter within its function call parameter list. Refer to [Vanderplaats Research and Development, 1995] for information on IPRM, RPRM, and the DOT function call parameter list.

13.33.2 Member Data Documentation

`int dotInfo` [private]

INFO from DOT manual.

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

Referenced by DOTOPTIMIZER::find_optimum(), and DOTOPTIMIZER::initialize_run().
int dotFDInfo  [private]

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

int dotMethod  [private]

METHOD from DOT manual.
   For nonlinear constraints: 0/1 = dot_mmfd, 2 = dot_slp, 3 = dot_sqp. For unconstrained: 0/1 = dot_bfgs, 2 =
   dot_frcg.
   Referenced by DOTOptimizer::allocate_constraints(), DOTOptimizer::allocate_workspace(), DOTOptimizer-
   ::DOTOptimizer(), and DOTOptimizer::find_optimum().

int printControl  [private]

IPRINT from DOT manual (controls output verbosity)
   Values range from 0 (least output) to 7 (most output).
   Referenced by DOTOptimizer::DOTOptimizer(), and DOTOptimizer::find_optimum().

RealArray realCntlParmArray  [private]

RPRM from DOT manual.
   Array of real control parameters.
   Referenced by DOTOptimizer::find_optimum(), and DOTOptimizer::initialize().

IntArray intCntlParmArray  [private]

IPRM from DOT manual.
   Array of integer control parameters.
   Referenced by DOTOptimizer::find_optimum(), and DOTOptimizer::initialize().

RealVector constraintValues  [private]

array of nonlinear constraint values passed to DOT
   This array must be of nonzero length and must contain only one-sided inequality constraints which are <= 0
   (which requires a transformation from 2-sided inequalities and equalities).
   Referenced by DOTOptimizer::allocate_constraints(), and DOTOptimizer::find_optimum().

SizetArray constraintMappingIndices  [private]

a container of indices for referencing the corresponding Response constraints used in computing the DOT con-
   straints.
   The length of the container corresponds to the number of DOT constraints, and each entry in the container
   points to the corresponding DAKOTA constraint.
   Referenced by DOTOptimizer::allocate_constraints(), and DOTOptimizer::find_optimum().
RealArray constraintMappingMultipliers [private]

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

The length of the container corresponds to the number of DOT constraints, and each entry in the container stores a multiplier for the DAKOTA constraint identified with constraintMappingIndices. These multipliers are currently +1 or -1.

Referenced by DOTOptimizer::allocate_constraints(), and DOTOptimizer::find_optimum().

RealArray constraintMappingOffsets [private]

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

The length of the container corresponds to the number of DOT constraints, and each entry in the container stores an offset for the DAKOTA constraint identified with constraintMappingIndices. These offsets involve inequality bounds or equality targets, since DOT assumes constraint allowables = 0.

Referenced by DOTOptimizer::allocate_constraints(), and DOTOptimizer::find_optimum().

The documentation for this class was generated from the following files:
- DOTOptimizer.hpp
- DOTOptimizer.cpp

13.34 JEGAOptimizer::Driver Class Reference

A subclass of the JEGA front end driver that exposes the individual protected methods to execute the algorithm.
Inherits Driver.

Public Member Functions

- GeneticAlgorithm * ExtractAllData (const AlgorithmConfig &algConfig)
  
  Reads all required data from the problem description database stored in the supplied algorithm config.

- DesignOFSortSet PerformIterations (GeneticAlgorithm *theGA)
  
  Performs the required iterations on the supplied GA.

- void DestroyAlgorithm (GeneticAlgorithm *theGA)
  
  Deletes the supplied GA.

- Driver (const ProblemConfig &probConfig)
  
  Default constructs a Driver.

13.34.1 Detailed Description

A subclass of the JEGA front end driver that exposes the individual protected methods to execute the algorithm.

This is necessary because DAKOTA requires that all problem information be extracted from the problem description DB at the time of Optimizer construction and the front end does it all in the execute algorithm method which must be called in find_optimum.

13.34.2 Constructor & Destructor Documentation

Driver ( const ProblemConfig & probConfig ) [inline]

Default constructs a Driver.
Parameters

| probConfig | The definition of the problem to be solved by this Driver whenever ExecuteAlgorithm is called. |

The problem can be solved in multiple ways by multiple algorithms even using multiple different evaluators by issuing multiple calls to ExecuteAlgorithm with different AlgorithmConfigs.

### 13.34.3 Member Function Documentation

**GeneticAlgorithm* ExtractAllData ( const AlgorithmConfig & algConfig )** [inline]

Reads all required data from the problem description database stored in the supplied algorithm config. The returned GA is fully configured and ready to be run. It must also be destroyed at some later time. You MUST call DestroyAlgorithm for this purpose. Failure to do so could result in a memory leak and an eventual segmentation fault! Be sure to call DestroyAlgorithm prior to destroying the algorithm config that was used to create it!

This is just here to expose the base class method to users.

**Parameters**

| algConfig | The fully loaded configuration object containing the database of parameters for the algorithm to be run on the known problem. |

**Returns**

The fully configured and loaded GA ready to be run using the PerformIterations method.

Referenced by JEGAOptimizer::find_optimum().

**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.

Referenced by JEGAOptimizer::find_optimum().

**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.
13.35  EFFGLOBALMINIMIZER CLASS REFERENCE

Parameters

| theGA | The algorithm that is no longer needed and thus must be destroyed. |

Referenced by JEGAOptimizer::find optimum(). The documentation for this class was generated from the following file:

- JEGAOptimizer.cpp

13.35  EffGlobalMinimizer Class Reference

Implementation of Efficient Global Optimization/Least Squares algorithms.

Inheritance diagram for EffGlobalMinimizer:

```
EffGlobalMinimizer
  SurrBasedMinimizer
    Minimizer
      Iterator
```

Public Member Functions

- EffGlobalMinimizer (ProblemDescDB &problem_db, Model &model)
  standard constructor
- ~EffGlobalMinimizer ()
  alternate constructor for instantiations "on the fly"
- void init_communicators ()
  initialize the communicators associated with iteratedModel as well as any subordinate iterators or subordinate models
- void free_communicators ()
  free the communicators associated with iteratedModel as well as any subordinate iterators or subordinate models
- void minimize_surrogates ()
  Used for computing the optimal solution using a surrogate-based approach. Redefines the Iterator::core run() virtual function.
- const Model & algorithm_space_model () const

Private Member Functions

- void minimize_surrogates_on_model ()
  called by minimize_surrogates for setUpType == "model"
- void get_best_sample ()
  called by minimize_surrogates for setUpType == "user functions"
• Real expected\_improvement (const RealVector &means, const RealVector &variances)
  expected improvement function for the GP
• RealVector expected\_violation (const RealVector &means, const RealVector &variances)
  expected violation function for the constraint functions
• void update\_penalty ()
  initialize and update the penaltyParameter

**Static Private Member Functions**

• static void EIF\_objective\_eval (const Variables &sub\_model\_vars, const Variables &recast\_vars, const Response &sub\_model\_response, Response &recast\_response)
  static function used as the objective function in the Expected Improvement (EIF) problem formulation for PMA

**Private Attributes**

• String setUpType
  controls iteration mode: "model" (normal usage) or "user\_functions" (user-supplied functions mode for "on the fly" instantiations).
• Model fHatModel
  GP model of response, one approximation per response function.
• Model eifModel
  recast model which assimilates mean and variance to solve the max(EIF) sub-problem
• Real meritFnStar
  minimum penalized response from among true function evaluations
• RealVector truthFnStar
  true function values corresponding to the minimum penalized response
• RealVector varStar
  point that corresponds to the optimal value meritFnStar
• short dataOrder
  order of the data used for surrogate construction, in ActiveSet request vector 3-bit format; user may override responses spec

**Static Private Attributes**

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

**Additional Inherited Members**

13.35.1 Detailed Description

Implementation of Efficient Global Optimization/Least Squares algorithms.

The EffGlobalMinimizer class provides an implementation of the Efficient Global Optimization algorithm developed by Jones, Schonlau, & Welch as well as adaptation of the concept to nonlinear least squares.
13.36. EFFICIENTSUBSPACEMETHOD CLASS REFERENCE

13.35.2 Constructor & Destructor Documentation

~EffGlobalMinimizer()

alternate constructor for instantiations "on the fly"

destructor

This is an alternate constructor for instantiations on the fly using a Model but no ProblemDescDB.

13.35.3 Member Function Documentation

const Model & algorithm_space_model() const [inline], [virtual]

default definition that gets redefined in selected derived Minimizers

Reimplemented from Minimizer.

References EffGlobalMinimizer::fHatModel.

void get_best_sample() [private]

called by minimize_surrogates for setUpType == "user_functions"

determine best solution from among sample data for expected improvement function

References Model::approximation_data(), SurrBasedMinimizer::augmented_lagrangian_merit(), Model::compute_response(), Model::continuous_variables(), Dakota::copy_data(), Model::current_response(), EffGlobalMinimizer::fHatModel, Response::function_values(), Iterator::iteratedModel, EffGlobalMinimizer::meritFnStar, Minimizer::numFunctions, SurrBasedMinimizer::origNonlinEqTargets, SurrBasedMinimizer::origNonlinIneqLowerBnds, SurrBasedMinimizer::origNonlinIneqUpperBnds, Model::primary_response_fn_sense(), Model::primary_response_fn_weights(), EffGlobalMinimizer::truthFnStar, and EffGlobalMinimizer::varStar.

Referenced by EffGlobalMinimizer::minimize_surrogates_on_model().

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

- EffGlobalMinimizer.hpp
- EffGlobalMinimizer.cpp

13.36 EfficientSubspaceMethod Class Reference

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

Inheritance diagram for EfficientSubspaceMethod:

```
  Iterator
    Analyzer
      NonD

EfficientSubspaceMethod
```
Public Member Functions

- **EfficientSubspaceMethod** (ProblemDescDB &problem_db, Model &model)
  
  Standard, model-based constructor.
- ~EfficientSubspaceMethod ()
  
  Destructor.
- void init_communicators ()
  
  specialization of init comm due to varied use of the original model
- void free_communicators ()
  
  specialization of free comm due to varied use of the original model
- void quantify_uncertainty ()
  
  ESM re-implementation of the virtual UQ iterator function.

Private Member Functions

- void validate_inputs ()
  
  validate user-supplied input values, setting defaults, aborting on error
- void init_fullspace_sampler ()
  
  initialize the native problem space Monte Carlo sampler
- void expand_basis (bool &mach_svtol_met, bool &user_svtol_met)
  
  generate fullspace samples, append to matrix, and factor, returning whether tolerance met
- unsigned int calculate_fullspace_samples ()
  
  determine the number of full space samples for next iteration, based on batchSize, limiting by remaining function
evaluation budget
- void generate_fullspace_samples (unsigned int diff_samples)
  
  sample the derivative at diff_samples points and leave temporary in dace_iterator
- void append_sample_matrices (unsigned int diff_samples)
  
  append the fullSpaceSampler samples to the derivative and vars matrices
- void compute_svd (bool &mach_svtol_met, bool &user_svtol_met)
  
  factor the derivative matrix and analyze singular values, assessing convergence and rank, returning whether toler-
ance met
- void print_svd_stats ()
  
  print inner iteration stats after SVD
- void assess_reconstruction (bool &recon_tol_met)
  
  determine if the reduced basis yields acceptable reconstruction error, based on sampling in the orthogonal comple-
ment of the reduced basis
- void reduced_space_uq ()
  
  experimental method to demonstrate creating a RecastModel and perform sampling-based UQ in the reduced space
- void uncertain_vars_to_subspace (Model &native_model, Model &vars_transform_model)
  
  translate the characterization of uncertain variables in the native_model to the reduced space of the transformed
model

Static Private Member Functions

- static void map_xi_to_x (const Variables &recast_xi_vars, Variables &sub_model_x_vars)
  
  map the active continuous recast variables to the active submodel variables (linear transformation)
Private Attributes

- int seedSpec
  - seed controlling all samplers
- int initialSamples
  - initial number of samples at which to query the truth model
- int batchSize
  - number of points to add at each iteration
- int subspaceSamples
  - number of UQ samples to perform in the reduced space
- unsigned int currIter
  - current iteration
- unsigned int totalSamples
  - total construction samples evaluated so far
- unsigned int totalEvals
  - total evaluations of model (accounting for UQ phase)
- double userSVTol
  - user-specified tolerance on singular value ratio
- double nullspaceTol
  - user-specified tolerance on nullspace
- double svRatio
  - current singular value ratio ($\sigma_k/\sigma_0$)
- unsigned int reducedRank
  - current approximation of system rank
- RealMatrix reducedBasis
  - basis for the reduced subspace
- RealMatrix derivativeMatrix
  - matrix of derivative data with numFunctions columns per fullspace sample; each column contains the gradient of one function at one sample point, so total matrix size is numContinuousVars * (numFunctions * numSamples) [ D1 | D2 | ... | Dnum_samples ] [ dy1 / dx(k=1) | dy2 / dx(k=1) | ... | dyM / dx(k=1) | k=2 | ... | k=n_s ]
- RealMatrix varsMatrix
  - matrix of fullspace variable points samples size numContinuousVars * (numSamples)
- Iterator fullSpaceSampler
  - Monte Carlo sampler for the full parameter space.

Static Private Attributes

- static EfficientSubspaceMethod * esmInstance
  - instance of this class for use in static member functions

Additional Inherited Members

13.36.1 Detailed Description

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

ESM uses random sampling to construct a low-dimensional subspace of the full dimensional parameter space, then performs UQ in the reduced space.
13.36.2 Member Function Documentation

void init_communicators() [virtual]
specialization of init comms due to varied use of the original model
This specialization is because the model is used in multiple contexts in this iterator, depending on build phase.
Note that this overrides the default behavior at Iterator which recurses into any submodels.
Reimplemented from Iterator.
References EfficientSubspaceMethod::batchSize, Model::init_communicators(), EfficientSubspaceMethod::initialSamples, and Iterator::iteratedModel.

void assess_reconstruction ( bool & recon_tol_met ) [private]
determine if the reduced basis yields acceptable reconstruction error, based on sampling in the orthogonal complement of the reduced basis
This function is experimental and needs to be carefully reviewed and cleaned up
References Iterator::activeSet, Model::aleatory_distribution_parameters(), Model::compute_response(), Model::continuous_variables(), Model::current_response(), Response::function_values(), Iterator::iteratedModel, Iterator::maxFunctionEvals, EfficientSubspaceMethod::nullspaceTol, Analyzer::numContinuousVars, Analyzer::numFunctions, Iterator::outputLevel, EfficientSubspaceMethod::reducedBasis, EfficientSubspaceMethod::reducedRank, ActiveSet::request_values(), EfficientSubspaceMethod::totalEvals, EfficientSubspaceMethod::totalSamples, and EfficientSubspaceMethod::varsMatrix.
Referenced by EfficientSubspaceMethod::quantify_uncertainty().

void reduced_space_uq ( ) [private]
experimental method to demonstrate creating a RecastModel and perform sampling-based UQ in the reduced space
This function is experimental and needs to be reviewed and cleaned up. In particular the translation of the correlations from full to reduced space is likely wrong. Transformation may be correct for covariance, but likely not correlations.
References Model::assign_rep(), NonD::construct_lhs(), Model::free_communicators(), NonD::generate_system_seed(), Model::init_communicators(), Iterator::iteratedModel, EfficientSubspaceMethod::map_xi_to_x(), Analyzer::numContinuousVars, Analyzer::numFunctions, Iterator::print_results(), EfficientSubspaceMethod::reducedBasis, EfficientSubspaceMethod::run(), Iterator::sampling_reset(), EfficientSubspaceMethod::seedSpec, Iterator::sub_iterator_flag(), EfficientSubspaceMethod::subspaceSamples, EfficientSubspaceMethod::uncertain_vars_to_subspace(), and Analyzer::vary_pattern().
Referenced by EfficientSubspaceMethod::quantify_uncertainty().

void uncertain_vars_to_subspace ( Model & native_model, Model & vars_transform_model ) [private]
translate the characterization of uncertain variables in the native_model to the reduced space of the transformed model
transform and set the distribution parameters in the reduced model
Convert the user-specified normal random variables to the appropriate reduced space variables, based on the orthogonal transformation.
TODO: Generalize to convert other random variable types
References Dakota::abort_handler(), Model::aleatory_distribution_parameters(), Analyzer::numContinuousVars, Iterator::outputLevel, EfficientSubspaceMethod::reducedBasis, and EfficientSubspaceMethod::reducedRank.
Referenced by EfficientSubspaceMethod::reduced_space_uq().
13.37. EMBEDHYBRIDMETAITERATOR CLASS REFERENCE

void map_xi_to_x ( const Variables & recast_xi_vars, Variables & sub_model_x_vars ) [static], [private]

map the active continuous recast variables to the active submodel variables (linear transformation)
    Perform the variables mapping from recast reduced dimension variables xi to original model x variables via
linear transformation. Maps only continuous variables.
References Variables::continuous_variables(), Dakota::copy_data(), EfficientSubspaceMethod::esmInstance,
Iterator::outputLevel, and EfficientSubspaceMethod::reducedBasis.
Referenced by EfficientSubspaceMethod::reduced_space_uq().
The documentation for this class was generated from the following files:

- EfficientSubspaceMethod.hpp
- EfficientSubspaceMethod.cpp

13.37 EmbedHybridMetaIterator Class Reference

Meta-iterator for closely-coupled hybrid iteration, typically involving the embedding of local search methods
within global search methods.
Inheritance diagram for EmbedHybridMetaIterator:

```
+-----------------+             +-----------------+
|       Iterator  |   MetaIterator   |
|                 |   EmbedHybridMetaIterator |
```

Public Member Functions

- EmbedHybridMetaIterator (ProblemDescDB &problem_db)  
  standard constructor
- EmbedHybridMetaIterator (ProblemDescDB &problem_db, Model &model)  
  alternate constructor
- ~EmbedHybridMetaIterator ()
  destructor

Protected Member Functions

- void core_run ()
  Performs the hybrid iteration by executing global and local iterators, using a set of models that may vary in fidelity.
- const Variables & variables_results () const
  return the final solution from the embedded hybrid (variables)
- const Response & response_results () const
  return the final solution from the embedded hybrid (response)
Private Attributes

- **Iterator globalIterator**
  the top-level outer iterator (e.g., global minimizer)
- **Model globalModel**
  the model employed by the top-level outer iterator
- **Iterator localIterator**
  the inner iterator (e.g., local minimizer)
- **Model localModel**
  the model employed by the inner iterator
- **Real localSearchProb**
  the probability of running a local search refinement within phases of the global minimization for tightly-coupled hybrids

Additional Inherited Members

13.37.1 Detailed Description

Meta-iterator for closely-coupled hybrid iteration, typically involving the embedding of local search methods within global search methods.

This meta-iterator 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:

- EmbedHybridMetaIterator.hpp
- EmbedHybridMetaIterator.cpp

13.38 Environment Class Reference

Base class for the environment class hierarchy.

Inheritance diagram for Environment:

```
Environment
    ^
   /\  \   \  \
Environment
    \  \  \  
ExecutableEnvironment LibraryEnvironment
```

Public Member Functions

- **Environment ()**
  default constructor: empty envelope
- **Environment (int argc, char *argv[])**
  envelope constructor for ExecutableEnvironment letter
- **Environment (ProgramOptions prog_opts)**
- **Environment (MPI_Comm dakota_mpi_comm, ProgramOptions prog_opts=ProgramOptions())**
- **Environment** (const String &env_type)
  
  *envelope constructor for letter type identified by String*

- **Environment** (const Environment &env)
  
  *copy constructor*

- **virtual ~Environment ()
  
  *destructor*

- **Environment operator= (const Environment &env)
  
  *assignment operator*

- **virtual void execute ()
  
  *the run function for the environment: invoke the iterator(s) on the model(s). Called from main.cpp.***

- **bool check () const
  
  *Print status of check and return true if in a ”check” mode, including version and help. Return false if proceeding to a run mode.***

- **MPIManager & mpi_manager ()
  
  *return mpiManager*

- **ProgramOptions & program_options ()
  
  *return programOptions*

- **OutputManager & output_manager ()
  
  *return outputManager*

- **ParallelLibrary & parallel_library ()
  
  *return parallelLib*

- **ProblemDescDB & problem_description_db ()
  
  *return probDescDB*

- **const Variables & variables_results () const
  
  *return the final environment solution (variables)*

- **const Response & response_results () const
  
  *return the final environment solution (response)*

### Protected Member Functions

- **Environment (BaseConstructor)
  
  *constructor initializes the base class part of default-constructed letters*

- **Environment (BaseConstructor, int argc, char *argv[])
  
  *constructor initializes the base class part of executable letter classes*

- **Environment (BaseConstructor, ProgramOptions prog_opts, MPI_Comm dakota_mpi_comm=MPI_COMM_M_WORLD)
  
  *constructor initializes the base class part of library letter classes*

- **void parse (bool check, bcast_database=true, DbCallbackFunctionPtr callback=NULL, void *callback_data= NULL)
  
  *parse inputs, callbacks, and optionally check and broadcast*

- **void construct ()
  
  *Instantiate topLevelIterator:*

- **void destruct ()
  
  *Deallocate parallel partitioning for topLevelIterator:***
Protected Attributes

- MPIManager mpiManager
  the MPI manager instance
- ProgramOptions programOptions
  the command line options manager
- OutputManager outputManager
  (tagged) output stream manager
- ParallelLibrary parallelLib
  the parallel library instance
- ProblemDescDB probDescDB
  the parser database instance
- Iterator topLevelIterator
  the top level (meta-)iterator

Private Member Functions

- Environment * get_environment (const String &env_type)
  Used by the envelope to instantiate the correct letter class.

Private Attributes

- Environment * environmentRep
  pointer to the letter (initialized only for the envelope)
- int referenceCount
  number of objects sharing environmentRep

13.38.1 Detailed Description

Base class for the environment class hierarchy.

The Environment class is the base class for the class hierarchy providing the top level control in DAKOTA. The environment is responsible for creating and managing iterators and models. For memory efficiency and enhanced polymorphism, the environment hierarchy employs the "letter/envelope idiom" (see Coplien "Advanced C++", p. 133), for which the base class (Environment) serves as the envelope and one of the derived classes (selected in Environment::get_environment()) serves as the letter.

13.38.2 Constructor & Destructor Documentation

Environment ()
default constructor: empty envelope

Default envelope constructor. environmentRep is NULL in this case, which makes it necessary to check for NULL in the copy constructor, assignment operator, and destructor.
Environment (int argc, char *argv[])  
Envelop constructor for ExecutableEnvironment letter  
   Envelope constructor for ExecutableEnvironment. Selection of derived type by get_environment() is not necessary in this case.  
   References Dakota::abort_handler(), and Environment::environmentRep.

Environment (ProgramOptions prog_opts)  
Envelop constructor for LibraryEnvironment. Selection of derived type by get_environment() is not necessary in this case.  
   References Dakota::abort_handler(), and Environment::environmentRep.

Environment (MPI_Comm dakota_mpi_comm, ProgramOptions prog_opts = ProgramOptions())  
Envelop constructor for LibraryEnvironment. Selection of derived type by get_environment() is not necessary in this case.  
   References Dakota::abort_handler(), and Environment::environmentRep.

Environment (const String &env_type)  
envelope constructor for letter type identified by String  
   Alternate construction by String. Envelope constructor invokes get_environment() which instantiates a derived class letter; the derived constructor selects a BaseConstructor constructor in its initialization list to avoid the recursion of a base class constructor calling get_environment() again.  
   References Dakota::abort_handler(), Environment::environmentRep, and Environment::get_environment().

Environment (const Environment &env)  
copy constructor  
   Copy constructor manages sharing of environmentRep and incrementing of referenceCount.  
   References Environment::environmentRep, and Environment::referenceCount.

~Environment() [virtual]  
destructor  
   Destructor decrements referenceCount and only deletes environmentRep when referenceCount reaches zero.  
   References Environment::destruct(), Environment::environmentRep, and Environment::referenceCount.

Environment (BaseConstructor) [protected]  
constructor initializes the base class part of default-constructed letters  
   This letter constructor initializes base class data for inherited environments that are default constructed. Since the letter IS the representation, its representation pointer is set to NULL (an uninitialized pointer causes problems in ~Environment).  
   Use cases: library with no options, no MPI comm
Environment ( BaseConstructor, int argc, char * argv[] ) [protected]

constructor initializes the base class part of executable letter classes

This letter constructor initializes base class data for inherited environments: instantiate/initialize the environment, options, parallel library, and problem description database objects. Since the letter IS the representation, its representation pointer is set to NULL (an uninitialized pointer causes problems in ~Environment).

Use cases: executable with command-line args
References nidr_save_exedir().

Environment ( BaseConstructor, ProgramOptions prog_opts, MPI_Comm dakota_mpi_comm = MPI_COMM_WORLD ) [protected]

constructor initializes the base class part of library letter classes

This letter constructor initializes base class data for inherited environments. Since the letter IS the representation, its representation pointer is set to NULL (an uninitialized pointer causes problems in ~Environment).

Use cases: library with program options library with program options and MPI comm

13.38.3 Member Function Documentation

Environment operator= ( const Environment & env )

assignment operator


References Environment::environmentRep, and Environment::referenceCount.

void parse ( bool check_bcast_database = true, DbCallbackFunctionPtr callback = NULL, void * callback_data = NULL ) [protected]

parse inputs, callbacks, and optionally check and broadcast

Parse input file and invoked any callbacks, then optionally check and sync database if check_bcast_database = true

References ProblemDescDB::check_and_bcast(), ProgramOptions::input_file(), ProgramOptions::input_string(), ProblemDescDB::parse_inputs(), Environment::probDescDB, and Environment::programOptions.

Referenced by ExecutableEnvironment::ExecutableEnvironment(), and LibraryEnvironment::LibraryEnvironment().

Environment * get_environment ( const String & env_type ) [private]

Used by the envelope to instantiate the correct letter class.

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

Referenced by Environment::Environment().

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

- DakotaEnvironment.hpp
- DakotaEnvironment.cpp

13.39 NomadOptimizer::Evaluator Class Reference

NOMAD-based Evaluator class.

Inherits Evaluator.
Public Member Functions

- **Evaluator** (const NOMAD::Parameters &p, Model &model)
  Constructor.
- **~Evaluator** (void)
  Destructor.
- **bool eval_x** (NOMAD::Eval_Point &x, const NOMAD::Double &h_max, bool &count_eval) const
  Main Evaluation Method.
- **void set_constraint_map** (int numNomadNonlinearIneqConstraints, int numNomadNonlinearEqConstraints,
  std::vector<int> constrMapIndices, std::vector<double> constrMapMultipliers, std::vector<double> constrMapOffsets)
  publishes constraint transformation

Private Attributes

- **Model & _model**
- **int n_cont**
- **int n_disc_int**
- **int n_disc_real**
- **int numNomadNonlinearIneqConstr**
  Number of nonlinear constraints after put into Nomad format.
- **int numNomadNonlinearEqConstr**
- **std::vector<int> constrMapIndices**
  map from Dakota constraint number to Nomad constraint number
- **std::vector<double> constrMapMultipliers**
  multipliers for constraint transformations
- **std::vector<double> constrMapOffsets**
  offsets for constraint transformations

13.39.1 Detailed Description

NOMAD-based Evaluator class.

The NOMAD process requires an evaluation step, which calls the Simulation program. In the simplest version
of this call, NOMAD executes the black box executable, which proceeds to write a file in a NOMAD-compatible
format, which NOMAD reads to continue the process.

Because DAKOTA files are different form NOMAD files, and the simulations processed by DAKOTA already
produce DAKOTA-compatible files, we cannot use this method for NOMAD. Instead, we implement the Nomad-
Evaluator class, which takes the NOMAD inputs and passes them to DAKOTA’s Interface for processing. The
evaluator then passes the evaluation Responses into the NOMAD objects for further analysis.

13.39.2 Constructor & Destructor Documentation

**Evaluator** ( const NOMAD::Parameters & p, Model & model )

Constructor.

NOMAD Evaluator Constructor
Parameters

<table>
<thead>
<tr>
<th>p</th>
<th>NOMAD Parameters object</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>DAKOTA Model object</td>
</tr>
</tbody>
</table>

### 13.39.3 Member Function Documentation

```cpp
bool eval ( NOMAD::Eval_Point & x, const NOMAD::Double & h_max, bool & count_eval ) const
```

Main Evaluation Method.

Method that handles the communication between the NOMAD search process and the Black Box Evaluation managed by DAKOTA's Interface.

Parameters

<table>
<thead>
<tr>
<th>x</th>
<th>Object that contains the points that need to evaluated. Once the evaluation is completed, this object also stores the output back to be read by NOMAD.</th>
</tr>
</thead>
<tbody>
<tr>
<td>h_max</td>
<td>Current value of the barrier parameter. Not used in this implementation.</td>
</tr>
<tr>
<td>count_eval</td>
<td>Flag that indicates whether this evaluation counts towards the max number of evaluations, often set to false when the evaluation does not meet certain costs during expensive evaluations. Not used in this implementation.</td>
</tr>
</tbody>
</table>

Returns

- `true` if the evaluation was successful; `false` otherwise.

References Dakota::set_index_to_value().

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

- NomadOptimizer.hpp
- NomadOptimizer.cpp

### 13.40 JEGAOptimizer::Evaluator Class Reference

An evaluator specialization that knows how to interact with Dakota. Inherits GeneticAlgorithmEvaluator.

**Public Member Functions**

- virtual bool Evaluate (DesignGroup &group)
  
  *Does evaluation of each design in group.*
- virtual bool Evaluate (Design &des)
  
  *This method cannot be used!!*
- virtual std::string GetName () const
  
  *Returns the proper name of this operator.*
- virtual std::string GetDescription () const
  
  *Returns a full description of what this operator does and how.*
- virtual GeneticAlgorithmOperator * Clone (GeneticAlgorithm &algorithm) const
Creates and returns a pointer to an exact duplicate of this operator.

- **Evaluator** (GeneticAlgorithm &algorithm, Model &model)
  Constructs a Evaluator for use by algorithm.

- **Evaluator** (const Evaluator &copy)
  Copy constructs a Evaluator.

- **Evaluator** (const Evaluator &copy, GeneticAlgorithm &algorithm, Model &model)
  Copy constructs a Evaluator for use by algorithm.

### Static Public Member Functions

- static const std::string & Name ()
  Returns the proper name of this operator.

- static const std::string & Description ()
  Returns a full description of what this operator does and how.

### Protected Member Functions

- void SeparateVariables (const Design &from, RealVector &intoCont, IntVector &intoDiscInt, RealVector &intoDiscReal) const
  This method fills intoCont, intoDiscInt and intoDiscReal appropriately using the values of from.

- void RecordResponses (const RealVector &from, Design &into) const
  Records the computed objective and constraint function values into into.

- std::size_t GetNumberNonLinearConstraints () const
  Returns the number of non-linear constraints for the problem.

- std::size_t GetNumberLinearConstraints () const
  Returns the number of linear constraints for the problem.

### Private Member Functions

- **Evaluator** (GeneticAlgorithm &algorithm)
  This constructor has no implementation and cannot be used.

### Private Attributes

- Model & _model
  The Model known by this evaluator.

### 13.40.1 Detailed Description

An evaluator specialization that knows how to interact with Dakota. This evaluator knows how to use the model to do evaluations both in synchronous and asynchronous modes.

### 13.40.2 Constructor & Destructor Documentation

**Evaluator** ( GeneticAlgorithm & algorithm, Model & model ) [inline]

Constructs a Evaluator for use by algorithm.

The optimizer is needed for purposes of variable scaling.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>algorithm</code></td>
<td>The GA for which the new evaluator is to be used.</td>
</tr>
<tr>
<td><code>model</code></td>
<td>The model through which evaluations will be done.</td>
</tr>
</tbody>
</table>

Evaluator ( `const Evaluator & copy` ) [inline]

Copy constructs a `Evaluator`.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>copy</code></td>
<td>The evaluator from which properties are to be duplicated into this.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>copy</code></td>
<td>The existing <code>Evaluator</code> from which to retrieve properties.</td>
</tr>
<tr>
<td><code>algorithm</code></td>
<td>The GA for which the new evaluator is to be used.</td>
</tr>
<tr>
<td><code>model</code></td>
<td>The model through which evaluations will be done.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>algorithm</code></td>
<td>The GA for which the new evaluator is to be used.</td>
</tr>
</tbody>
</table>

13.40.3 Member Function Documentation

static const std::string& Name ( ) [inline],[static]

Returns the proper name of this operator.

Returns

The string "DAKOTA JEGA Evaluator".

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.

```cpp
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>from</td>
<td>The Design class object from which to extract the discrete design variable values.</td>
</tr>
<tr>
<td>intoDiscInt</td>
<td>The vector into which to place the extracted discrete integer values.</td>
</tr>
<tr>
<td>intoDiscReal</td>
<td>The vector into which to place the extracted discrete real values.</td>
</tr>
<tr>
<td>intoCont</td>
<td>The vector into which to place the extracted continuous values.</td>
</tr>
</tbody>
</table>

References JEGAOptimizer::Evaluator::model, Model::cv(), Model::discrete_int_sets(), Model::div(), and Model::drv().

```cpp
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>from</td>
<td>The vector of responses to install into into.</td>
</tr>
<tr>
<td>into</td>
<td>The Design to which the responses belong and into which they must be written.</td>
</tr>
</tbody>
</table>

```cpp
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.

```cpp
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.
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

ture if all evaluations completed and false otherwise.

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.

virtual std::string GetName ( ) const [inline], [virtual]

Returns the proper name of this operator.

Returns

See Name().

virtual std::string GetDescription ( ) const [inline], [virtual]

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

Returns

See Description().

virtual GeneticAlgorithmOperator* Clone ( GeneticAlgorithm & algorithm ) const [inline], [virtual]

Creates and returns a pointer to an exact duplicate of this operator.
13.41. JEGAOPTIMIZER::EVALUATORCREATOR CLASS REFERENCE

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>algorithm</td>
<td>The GA for which the clone is being created.</td>
</tr>
</tbody>
</table>

Returns

A clone of this operator.

13.40.4 Member Data Documentation

Model& _model [private]

The Model known by this evaluator.

It is through this model that evaluations will take place.

Referenced by JEGAOptimizer::Evaluator::SeparateVariables().

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

- JEGAOptimizer.cpp

13.41 JEGAOptimizer::EvaluatorCreator Class Reference

A specialization of the JEGA::FrontEnd::EvaluatorCreator that creates a new instance of a Evaluator.

Inherits EvaluatorCreator.

Public Member Functions

- virtual GeneticAlgorithmEvaluator* CreateEvaluator (GeneticAlgorithm &alg)
  
  *Overridden to return a newly created Evaluator.*

- EvaluatorCreator (Model &theModel)
  
  construct an EvaluatorCreator using the supplied model.

Private Attributes

- Model& _theModel
  
  The user defined model to be passed to the constructor of the Evaluator.

13.41.1 Detailed Description

A specialization of the JEGA::FrontEnd::EvaluatorCreator that creates a new instance of a Evaluator.

13.41.2 Constructor & Destructor Documentation

EvaluatorCreator ( Model &theModel ) [inline]

Constructs an EvaluatorCreator using the supplied model.
Parameters

| theModel | The Dakota::Model this creator will pass to the created evaluator. |

13.41.3 Member Function Documentation

virtual GeneticAlgorithmEvaluator* CreateEvaluator ( GeneticAlgorithm & alg ) [inline], [virtual]

Overridden to return a newly created Evaluator.

The GA will assume ownership of the evaluator so we needn’t worry about keeping track of it for destruction. The additional parameters needed by the Evaluator are stored as members of this class at construction time.

Parameters

| alg | The GA for which the evaluator is to be created. |

Returns

A pointer to a newly created Evaluator.

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

- JEGAOptimizer.cpp

13.42 ExecutableEnvironment Class Reference

Environment corresponding to execution as a stand-alone application.

Inheritance diagram for ExecutableEnvironment:

```
Environment
   +--- ExecutableEnvironment
```

Public Member Functions

- ExecutableEnvironment ()
  
  *default constructor*

- ExecutableEnvironment (int argc, char *argv[])
  
  *constructor*

- ~ExecutableEnvironment ()
  
  *destructor*

- void execute ()
  
  *the run function for the environment: invoke the iterator(s) on the model(s). Called from main.cpp.*
13.43. EXPERIMENTDATA CLASS REFERENCE

Private Attributes

- boost::shared_ptr<TrackerHTTP> usageTracker
  
  posts usage data to Web server; using shared_ptr due to potentially incomplete type and requirements for checked-delete in debug builds (scoped_ptr would suffice)

Additional Inherited Members

13.42.1 Detailed Description

Environment corresponding to execution as a stand-alone application.

This environment corresponds to a stand-alone executable program, e.g., main.cpp. It sets up the ParallelLibrary, ProgramOptions, and ProblemDescDB objects based on access to command line arguments.

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

- ExecutableEnvironment.hpp
- ExecutableEnvironment.cpp

13.43 ExperimentData Class Reference

Public Member Functions

- void load scalar (const std::string &expDataFilename, const std::string &context_message, size_t numExperiments, IntVector &numReplicates, size_t numExpConfigVars, size_t numFunctions, size_t numExpStdDeviationsRead, bool expDataFileAnnotated, bool calc_sigma_from_data, short verbosity)
  
  constructor

- const RealVector & config vars (size_t response, size_t experiment)
  
  retrieve the vector of configuration variables for the given response and experiment number

- Real scalar data (size_t response, size_t experiment, size_t replicate)
  
  retrieve the data value for the given response, for the given experiment and replicate

- Real scalar sigma (size_t response, size_t experiment, size_t replicate)
  
  retrieve the standard deviation value for the given response, for the given experiment and replicate

Public Attributes

- std::vector<ExpDataPerResponse> allExperiments
  
  At the outer level, ExperimentData will just be a vector of ExpDataPerResponse;

13.43.1 Detailed Description

The ExperimentData class is used to read and populate data (currently from user-specified files and/or the input spec) relating to experimental (physical observations) data for the purposes of calibration. Such data may include (for example): number of experiments, number of replicates, configuration variables, type of data (scalar vs. functional), treatment of sigma (experimental uncertainties). This class also provides an interpolation capability to interpolate between simulation or experimental data so that the differencing between simulation and experimental data may be performed properly.
13.43.2 Member Function Documentation

```cpp
void load_scalar ( const std::string & expDataFilename, const std::string & context_message, size_t numExperiments, IntVector & numReplicates, size_t numExpConfigVars, size_t numFunctions, size_t numExpStdDeviationsRead, bool expDataFileAnnotated, bool calc_sigma_from_data, short verbosity )
```

**constructor**
- Constructor from legacy file format
- References Dakota::read_historical_data().
- Referenced by Minimizer::data_transform_model(), NonDQUESOBayesCalibration::quantify_uncertainty(), NonDDREAMBayesCalibration::quantify_uncertainty(), and NonDGPM SabhaCalibration::quantify_uncertainty().
- The documentation for this class was generated from the following files:
  - ExperimentData.hpp
  - ExperimentData.cpp

13.44 ForkApplicInterface Class Reference

Derived application interface class which spawns simulation codes using fork/execvp/waitpid.

Inheritance diagram for ForkApplicInterface:

```
<table>
<thead>
<tr>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplicationInterface</td>
</tr>
<tr>
<td>ProcessApplicInterface</td>
</tr>
<tr>
<td>ProcessHandleApplicInterface</td>
</tr>
<tr>
<td>ForkApplicInterface</td>
</tr>
</tbody>
</table>
```

**Public Member Functions**
- `ForkApplicInterface (const ProblemDescDB &problem_db)`
  - constructor
- `~ForkApplicInterface ()`
  - destructor

**Protected Member Functions**
- `void wait_local_evaluations (PRPQueue &prp_queue)`
  - For asynchronous function evaluations, this method is used to detect completion of jobs and process their results. It provides the processing code that is specific to derived classes. This version waits for at least one completion.
- `void test_local_evaluations (PRPQueue &prp_queue)`
For asynchronous function evaluations, this method is used to detect completion of jobs and process their results. It provides the processing code that is specific to derived classes. This version is nonblocking and will return without any completions if none are immediately available.

- **pid_t create_analysis_process (bool block_flag, bool new_group)**
  
  spawn a child process for an analysis component within an evaluation using fork()/vfork()/execvp() and wait for completion using waitpid() if block_flag is true

- **size_t wait_local_analyses ()**
  
  wait for asynchronous analyses on the local processor, completing at least one job

- **size_t test_local_analyses_send (int analysis_id)**
  
  test for asynchronous analysis completions on the local processor and return results for any completions by sending messages

- **void join_evaluation_process_group (bool new_group)**
  
  create (if new_group) and join the process group for asynch evaluations

- **void join_analysis_process_group (bool new_group)**
  
  create (if new_group) and join the process group for asynch analyses

- **void evaluation_process_group_id (pid_t pgid)**
  
  set evalProcGroupId

- **pid_t evaluation_process_group_id () const**
  
  return evalProcGroupId

- **void analysis_process_group_id (pid_t pgid)**
  
  set analysisProcGroupId

- **pid_t analysis_process_group_id () const**
  
  return analysisProcGroupId

- **pid_t wait_evaluation (bool block_flag)**
  
  process all available completions within the evaluation process group; if block_flag = true, wait for at least one completion

- **pid_t wait_analysis (bool block_flag)**
  
  process all available completions within the analysis process group; if block_flag = true, wait for at least one completion

- **void check_group (int err, pid_t proc_group_id)**
  
  check the exit status of setpgid and abort if an error code was returned

### Private Member Functions

- **pid_t wait (pid_t proc_group_id, std::map< pid_t, int > &process_id_map, bool block_flag)**
  
  core code used by wait_{evaluation,analysis}()

- **void join_process_group (pid_t &process_group_id, bool new_group)**
  
  core code used by join_{evaluation,analysis}_process_group()

### Private Attributes

- **pid_t evalProcGroupId**
  
  the process group id used to identify a set of child evaluation processes used by this interface instance (to distinguish from other interface instances that could be running at the same time)

- **pid_t analysisProcGroupId**
  
  the process group id used to identify a set of child analysis processes used by this interface instance (to distinguish from other interface instances that could be running at the same time)
Additional Inherited Members

13.44.1 Detailed Description

Derived application interface class which spawns simulation codes using fork/execvp/waitpid.

ForkApplicInterface is used on Unix systems and is a peer to SpawnApplicInterface for Windows systems.

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

- ForkApplicInterface.hpp
- ForkApplicInterface.cpp

13.45 FSUDesignCompExp Class Reference

Wrapper class for the FSUDace QMC/CVT library.

Inheritance diagram for FSUDesignCompExp:

```
    Iterator
     │
     │ Analyzer
      └── PStudyDACE
           └── FSUDesignCompExp
```

Public Member Functions

- **FSUDesignCompExp (ProblemDescDB &problem_db, Model &model)**
  primary constructor for building a standard DACE iterator
- **FSUDesignCompExp (Model &model, int samples, int seed, unsigned short sampling_method)**
  alternate constructor for building a DACE iterator on-the-fly
- **~FSUDesignCompExp ()**
  destructor
- **void pre_run ()**
  pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori
- **void extract_trends ()**
  Mapping of the core_run() virtual function for the PSudy/DACE branch.
- **void post_input ()**
  read tabular data for post-run mode
- **void post_run (std::ostream &s)**
  post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way
- **int num_samples () const**
- **void sampling_reset (int min_samples, bool all_data_flag, bool stats_flag)**
reset sampling iterator to use at least min_samples

- unsigned short sampling_scheme () const
  return sampling name
- void vary_pattern (bool pattern_flag)
  sets varyPattern in derived classes that support it
- void get_parameter_sets (Model &model)
  Returns one block of samples (ndim * num_samples)

Private Member Functions

- void enforce_input_rules ()
  enforce sanity checks/modifications for the user input specification

Private Attributes

- int samplesSpec
  initial specification of number of samples
- int numSamples
  current number of samples to be evaluated
- bool allDataFlag
  flag which triggers the update of allVars/allResponses for use by Iterator::all_variables() and Iterator::all_responses()
- size_t numDACERuns
  counter for number of executions for this object
- bool latinizeFlag
  flag which specifies latinization of QMC or CVT sample sets
- IntVector sequenceStart
  Integer vector defining a starting index into the sequence for random variable sampled. Default is 0 0 0 (e.g. for three random variables).
- IntVector sequenceLeap
  Integer vector defining the leap number for each sequence being generated. Default is 1 1 1 (e.g. for three random vars.)
- IntVector primeBase
  Integer vector defining the prime base for each sequence being generated. Default is 2 3 5 (e.g., for three random vars.)
- int seedSpec
  the user seed specification for the random number generator (allows repeatable results)
- int randomSeed
  current seed for the random number generator
- bool varyPattern
  flag for continuing the random number or QMC sequence from a previous execution (e.g., for surrogate-based optimization) so that multiple executions are repeatable but not identical.
- int numCVTTrials
  specifies the number of sample points taken at internal CVT iteration
- int trialType
  Trial type in CVT. Specifies where the points are placed for consideration relative to the centroids. Choices are grid (2), halton (1), uniform (0), or random (-1). Default is random.
Additional Inherited Members

13.45.1 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.

13.45.2 Constructor & Destructor Documentation

FSUDesignCompExp ( ProblemDescDB & problem db, Model & model )

primary constructor for building a standard DACE iterator

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

References Dakota::abort_handler(), ProblemDescDB::get_bool(), ProblemDescDB::get_int(), ProblemDescDB::get_iv(), ProblemDescDB::get_string(), Iterator::maxEvalConcurrency, Iterator::methodName, Analyzer::numContinuousVars, FSUDesignCompExp::numCVTTrials, FSUDesignCompExp::numSamples, FSUDesignCompExp::primeBase, Iterator::probDescDB, FSUDesignCompExp::randomSeed, FSUDesignCompExp::seedSpec, FSUDesignCompExp::sequenceLeap, FSUDesignCompExp::sequenceStart, FSUDesignCompExp::trialType, and FSUDesignCompExp::varyPattern.

FSUDesignCompExp ( Model & model, int samples, int seed, unsigned short sampling_method )

alternate constructor for building a DACE iterator on-the-fly

This alternate constructor is used for instantiations on-the-fly, using only the incoming data. No problem description database queries are used.

References Dakota::abort_handler(), Iterator::maxEvalConcurrency, Iterator::methodName, Analyzer::numContinuousVars, FSUDesignCompExp::numCVTTrials, FSUDesignCompExp::numSamples, FSUDesignCompExp::primeBase, Iterator::probDescDB, FSUDesignCompExp::randomSeed, FSUDesignCompExp::seedSpec, FSUDesignCompExp::sequenceLeap, FSUDesignCompExp::sequenceStart, and FSUDesignCompExp::trialType.

13.45.3 Member Function Documentation

void pre_run ( ) [virtual]

pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori

pre-run phase, which a derived iterator may optionally reimplement; when not present, pre-run is likely integrated into the derived run function. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s pre_run(), if implemented, typically before performing its own implementation steps.

Reimplemented from Iterator.

References FSUDesignCompExp::get_parameter_sets(), Iterator::iteratedModel, and PStudyDACE::varBasedDecompFlag.

void post_run ( std::ostream & s ) [virtual]

post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way
Post-run phase, which a derived iterator may optionally reimplement; when not present, post-run is likely integrated into run. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s `post_run()`, typically after performing its own implementation steps.

Reimplemented from `Analyzer`.

References `Analyzer::allResponses`, `Analyzer::allSamples`, `SensAnalysisGlobal::compute_correlations()`, `Analyzer::post_run()`, `PStudyDACE::pStudyDACESensGlobal`, `Iterator::subIteratorFlag`, and `PStudyDACE::varBasedDecompFlag`.

```cpp
int num_samples() const [inline], [virtual]
```

Return current number of evaluation points. Since the calculation of samples, collocation points, etc. might be costly, provide a default implementation here that backs out from the maxEvalConcurrency.

Reimplemented from `Analyzer`.

References `FSUDesignCompExp::numSamples`.

```cpp
void enforce_input_rules() [private]
```

enforce sanity checks/modifications for the user input specification

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

References `Dakota::abort_handler()`, `Iterator::methodName`, `Analyzer::numContinuousVars`, `FSUDesignCompExp::numSamples`, and `FSUDesignCompExp::primeBase`.

Referenced by `FSUDesignCompExp::get_parameter_sets()`.

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

- `FSUDesignCompExp.hpp`
- `FSUDesignCompExp.cpp`

### 13.46 GaussProcApproximation Class Reference

Derived approximation class for Gaussian Process implementation.

Inheritance diagram for `GaussProcApproximation`:

```
  Approximation
     |         GaussProcApproximation
```

**Public Member Functions**

- `GaussProcApproximation()`  
  *default constructor*
- `GaussProcApproximation(const SharedApproxData &shared_data)`  
  *alternate constructor*
- `GaussProcApproximation(const ProblemDescDB &problem_db, const SharedApproxData &shared_data)`  
  *standard constructor*
- `~GaussProcApproximation()`  
  *destructor*
CHAPTER 13. CLASS DOCUMENTATION

Protected Member Functions

- int min_coefficients () const
  return the minimum number of samples (unknowns) required to build the derived class approximation type in num-
  Vars dimensions
- int num_constraints () const
  return the number of constraints to be enforced via an anchor point
- void build ()
  find the covariance parameters governing the Gaussian process response
- Real value (const Variables &vars)
  retrieve the function value for a given parameter set
- const RealVector & gradient (const Variables &vars)
  retrieve the function gradient at the predicted value for a given parameter set
- Real prediction_variance (const Variables &vars)
  retrieve the variance of the predicted value for a given parameter set

Private Member Functions

- void GPmodel_build ()
  Function to compute hyperparameters governing the GP.
- void GPmodel_apply (const RealVector &new_x, bool variance_flag, bool gradients_flag)
  Function returns a response value using the GP surface.
- void normalize_training_data ()
  Normalizes the initial inputs upon which the GP surface is based.
- void get_trend ()
  Gets the trend (basis) functions for the calculation of the mean of the GP. If the order = 0, the trend is a con-
  stant, if the order = 1, trend is linear, if order = 2, trend is quadratic.
- void get_beta_coefficients ()
  Gets the beta coefficients for the calculation of the mean of the GP.
- int get_cholesky_factor ()
  Gets the Cholesky factorization of the covariance matrix, with error checking.
- void get_process_variance ()
  Gets the estimate of the process variance given the values of beta and the correlation lengthscales.
- void get_cov_matrix ()
  calculates the covariance matrix for a given set of input points
- void get_cov_vector ()
  calculates the covariance vector between a new point x and the set of inputs upon which the GP is based
- void optimize_theta_global ()
  sets up and performs the optimization of the negative log likelihood to determine the optimal values of the covari-
  ance parameters using NCSUDirect
- void optimize_theta_multipoint ()
  sets up and performs the optimization of the negative log likelihood to determine the optimal values of the covari-
  ance 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 \texttt{calc\_nll} ()
  
  calculates the negative log likelihood function (based on covariance matrix)

- void \texttt{calc\_grad\_nll} ()
  
  Gets the gradient of the negative log likelihood function with respect to the correlation lengthscales, theta.

- void \texttt{get\_grad\_cov\_vector} ()
  
  Calculate the derivatives of the covariance vector, with respect to each component of x.

- void \texttt{run\_point\_selection} ()
  
  Runs the point selection algorithm, which will choose a subset of the training set with which to construct the GP model, and estimate the necessary parameters.

- void \texttt{initialize\_point\_selection} ()
  
  Initializes the point selection routine by choosing a small initial subset of the training points.

- void \texttt{pointsel\_get\_errors} (RealArray &delta)
  
  Uses the current GP model to compute predictions at all of the training points and find the errors.

- int \texttt{addpoint} (int, IntArray &added\_index)
  
  Adds a point to the effective training set. Returns 1 on success.

- int \texttt{pointsel\_add\_sel} (const RealArray &delta)
  
  Accepts a vector of unsorted prediction errors, determines which points should be added to the effective training set, and adds them.

- Real \texttt{maxval} (const RealArray &) const
  
  Return the maximum value of the elements in a vector.

- void \texttt{pointsel\_write\_points} ()
  
  Writes out the training set before and after point selection.

- void \texttt{lhood\_2d\_grid\_eval} ()
  
  For problems with 2D input, evaluates the negative log likelihood on a grid.

- void \texttt{write\_x} (const char[])
  
  Writes out the current training set (in original units) to a specified file.

- void \texttt{writeCovMat} (char[])
  
  Writes out the covariance matrix to a specified file.

**Static Private Member Functions**

- static void \texttt{negloglik} (int mode, int n, const Teuchos::SerialDenseVector<int, double> &X, Real &fx, Teuchos::SerialDenseVector<int, double> &grad_x, int &result_mode)
  
  static function used by \texttt{OPT++} as the objective function to optimize the hyperparameters in the covariance of the GP by minimizing the negative log likelihood

- static void \texttt{constraint\_eval} (int mode, int n, const Teuchos::SerialDenseVector<int, double> &X, Teuchos::SerialDenseVector<int, double> &g, Teuchos::SerialDenseMatrix<int, double> &gradC, int &result_mode)
  
  static function used by \texttt{OPT++} as the constraint function in the optimization of the negative log likelihood. Currently this function is empty: it is an unconstrained optimization.

- static double \texttt{negloglikNCSU} (const RealVector &x)
  
  function used by \texttt{NCSU\_Optimizer} to optimize negloglik objective
Private Attributes

- **Real approxValue**
  
  Value of the approximation returned by `value()`

- **Real approxVariance**
  
  Value of the approximation returned by `prediction_variance()`

- **RealMatrix trainPoints**
  
  A 2-D array (num sample sites = rows, num vars = columns) used to create the Gaussian process.

- **RealMatrix trainValues**
  
  An array of response values; one response value per sample site.

- **RealVector trainMeans**
  
  The mean of the input columns of trainPoints.

- **RealVector trainStdvs**
  
  The standard deviation of the input columns of trainPoints.

- **RealMatrix normTrainPoints**
  
  Current working set of normalized points upon which the GP is based.

- **RealMatrix trendFunction**
  
  Matrix to hold the trend function

- **RealMatrix betaCoeffs**
  
  Matrix to hold the beta coefficients for the trend function

- **RealSymMatrix covMatrix**
  
  The covariance matrix where each element (i,j) is the covariance between points Xi and Xj in the initial set of samples.

- **RealMatrix covVector**
  
  The covariance vector where each element (j,0) is the covariance between a new point X and point Xj from the initial set of samples.

- **RealMatrix approxPoint**
  
  Point at which a prediction is requested. This is currently a single point, but it could be generalized to be a vector of points.

- **RealMatrix gradNegLogLikTheta**
  
  Matrix to hold the gradient of the negative log likelihood with respect to the theta correlation terms

- **Teuchos::SerialSpdDenseSolver< int, Real > covSlvr**
  
  The global solver for all computations involving the inverse of the covariance matrix.

- **RealMatrix gradCovVector**
  
  A matrix, where each column is the derivative of the covVector with respect to a particular component of X.

- **RealMatrix normTrainPointsAll**
  
  Set of all original samples available.

- **RealMatrix trainValuesAll**
  
  All original samples available.

- **RealMatrix trendFunctionAll**
  
  Trend function values corresponding to all original samples.

- **RealMatrix Rinv_YFB**
  
  Matrix for storing inverse of correlation matrix Rinv*(Y-FB)
• size_t numObs
  The number of observations on which the GP surface is built.

• size_t numObsAll
  The original number of observations.

• short trendOrder
  The number of variables in each X variable (number of dimensions of the problem).

• RealVector thetaParams
  Theta is the vector of covariance parameters for the GP. We determine the values of theta by optimization. Currently, the covariance function is $\theta_0 \exp(-0.5 \times \text{sume}) + \delta \times \text{pow}(\sigma_e, 2)$. sume is the sum squared of weighted distances: it involves a sum of $\theta_1 (X_i(1) - X_j(1))^2 + \theta_2 (X_i(2) - X_j(2))^2 + ...$ where $X_i(1)$ is the first dimension value of multi-dimensional variable $X_i$. $\delta \times \text{pow}(\sigma_e, 2)$ is a jitter term used to improve matrix computations. $\delta$ is zero for the covariance between different points and 1 for the covariance between the same point. $\sigma_e$ is the underlying process error.

• Real procVar
  The process variance, the multiplier of the correlation matrix.

• IntArray pointsAddedIndex
  Used by the point selection algorithm, this vector keeps track all points which have been added.

• int cholFlag
  A global indicator for success of the Cholesky factorization.

• bool usePointSelection
  A flag to indicate the use of point selection

Static Private Attributes

• static GaussProcApproximation * GPInstance
  pointer to the active object instance used within the static evaluator

Additional Inherited Members

13.46.1 Detailed Description

Derived approximation class for Gaussian Process implementation.

The GaussProcApproximation class provides a global approximation (surrogate) based on a Gaussian process. The Gaussian process is built after normalizing the function values, with zero mean. Opt++ is used to determine the optimal values of the covariance parameters, those which minimize the negative log likelihood function.

13.46.2 Constructor & Destructor Documentation

GaussProcApproximation( ) [inline]

default constructor

alternate constructor used by EffGlobalOptimization and NonDGlobalReliability that does not use a problem database defaults here are no point selectinn and quadratic trend function.
13.46.3 Member Function Documentation

```cpp
void GPmodel_apply ( const RealVector & new_x, bool variance_flag, bool gradients_flag ) [private]
```

Function returns a response value using the GP surface.

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

References Dakota::abort_handler(), GaussProcApproximation::approxPoint, GaussProcApproximation::get_cov_vector(), SharedApproxData::numVars, GaussProcApproximation::predict(), Approximation::sharedDataRep, GaussProcApproximation::trainMeans, and GaussProcApproximation::trainStdvs.

Referenced by GaussProcApproximation::gradient(), GaussProcApproximation::pointsel_get_errors(), GaussProcApproximation::prediction_variance(), and GaussProcApproximation::value().

13.46.4 Member Data Documentation

```cpp
short trendOrder [private]
```

The number of variables in each X variable (number of dimensions of the problem).

The order of the basis function for the mean of the GP If the order = 0, the trend is a constant, if the order = 1, trend is linear, if order = 2, trend is quadratic.

Referenced by GaussProcApproximation::GaussProcApproximation(), GaussProcApproximation::get_beta_coefficients(), GaussProcApproximation::get_trend(), GaussProcApproximation::GPmodel_build(), and GaussProcApproximation::predict().

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

* GaussProcApproximation.hpp
* GaussProcApproximation.cpp

13.47 GetLongOpt Class Reference

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

Inheritance diagram for GetLongOpt:

```
  GetLongOpt
    CommandLineHandler
```

Public Types

```cpp
enum OptType { Valueless, OptionalValue, MandatoryValue }
```

enum for different types of values associated with command line options.

Public Member Functions

```cpp
GetLongOpt (const char optmark='\'-\')
```

Constructor.

```cpp
~GetLongOpt ()
```

Constructor.
13.47. GETLONGOPT CLASS REFERENCE

Destructor.

- **int parse (int argc, char const *argv)**
  
  Parse the command line args (argc, argv).

- **int parse (char const str, char const p)**
  
  Parse a string of options (typically given from the environment).

- **int enroll (const char const opt, const OptType t, const char const desc, const char const val)**
  
  Add an option to the list of valid command options.

- **const char * retrieve (const char const opt) const**
  
  Retrieve value of option.

- **void usage (std::ostream &outfile=Cout) const**
  
  Print usage information to outfile.

- **void usage (const char *str)**
  
  Change header of usage output to str.

- **void store (const char *name, const char *value)**
  
  Store a specified option value.

Private Member Functions

- **char * basename (char const p) const**
  
  Extract the base name from a string as delimited by '/'

- **int setcell (Cell *c, char *valtoken, char *nexttoken, const char *p)**
  
  Internal convenience function for setting Cell::value

Private Attributes

- **Cell * table**
  
  Option table

- **const char * ustring**
  
  Usage message

- **char * pname**
  
  Program basename

- **char optmarker**
  
  Option marker

- **int enroll_done**
  
  Finished enrolling

- **Cell * last**
  
  Last entry in option table

13.47.1 Detailed Description

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

GetLongOpt manages the definition and parsing of "long options." Command line options can be abbreviated as long as there is no ambiguity. If an option requires a value, the value should be separated from the option either by whitespace or an "=".
13.47.2 Member Enumeration Documentation

enum OptType
enum for different types of values associated with command line options.

- **Valueless** option that may never have a value
- **OptionalValue** option with optional value
- **MandatoryValue** option with required value

13.47.3 Constructor & Destructor Documentation

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.

13.47.4 Member Function Documentation

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.
- Referenced by CommandLineHandler::check_usage().

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.

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 string. The second is an enum saying if the option is a flag (Valueless), if it requires a mandatory value (MandatoryValue) or if it takes an optional value (OptionalValue). The third argument is a string giving a brief description of the option. This description will be used by GetLongOpt::usage. GetLongOpt for usage-printing, uses ${val} to represent values needed by the options. {<$val>} is a mandatory value and {$val} is an optional value. The final argument to enroll is the default string to be returned if the option is not specified. For flags (options with Valueless), use "" (empty string, or in fact any arbitrary string) for specifying TRUE and 0 (null pointer) to specify FALSE.
- References GetLongOpt::enroll, GetLongOpt::last, and GetLongOpt::table.
- Referenced by CommandLineHandler::initialize_options().
const char * retrieve ( const char * const opt ) const

Retrieve value of option.

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

References GetLongOpt::optmarker, and GetLongOpt::table.

Referenced by CommandLineHandler::check_usage(), ProgramOptions::manage_run_modes(), ProgramOptions::ProgramOptions(), and CommandLineHandler::read_restart_evals().

void usage ( const char * str ) [inline]

Change header of usage output to str.

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

References GetLongOpt::ustring.

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

- CommandLineHandler.hpp
- CommandLineHandler.cpp

13.48  Graphics Class Reference

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

Public Member Functions

- Graphics ()
  constructor
- ~Graphics ()
  destructor
- void create_plots_2d (const Variables &vars, const Response &response)
  creates the 2d graphics window and initializes the plots
- void create_tabular_datastream (const Variables &vars, const Response &response, const std::string &tabular_data_file)
  opens the tabular data file stream and prints the headings
- void add_datapoint (const Variables &vars, const Response &response)
  adds data to each window in the 2d graphics and adds a row to the tabular data file based on the results of a model evaluation
- void add_datapoint (int i, double x, double y)
  adds data to a single window in the 2d graphics
- void new_dataset (int i)
  creates a separate line graphic for subsequent data points for a single window in the 2d graphics
- void close ()
close graphics windows

- **void close_tabular ()**

  close tabular data stream

- **void set_x_labels2d (const char *x_label)**
  set x label for each plot equal to x_label

- **void set_y_labels2d (const char *y_label)**
  set y label for each plot equal to y_label

- **void set_x_label2d (int i, const char *x_label)**
  set x label for ith plot equal to x_label

- **void set_y_label2d (int i, const char *y_label)**
  set y label for ith plot equal to y_label

- **void graphics_counter (int cntr)**
  set graphicsCntr equal to cntr

- **int graphics_counter () const**
  return graphicsCntr

- **void tabular_counter_label (const std::string &label)**
  set tabularCntrLabel equal to label

**Private Attributes**

- **Graphics2D * graphics2D**
  pointer to the 2D graphics object

- **bool win2dOn**
  flag to indicate if 2D graphics window is active

- **bool tabularDataFlag**
  flag to indicate if tabular data stream is active

- **int graphicsCntr**
  used for x axis values in 2D graphics and for 1st column in tabular data

- **std::string tabularCntrLabel**
  label for counter used in first line comment w/ the tabular data file

- **std::ofstream tabularDataFStream**
  file stream for tabulation of graphics data within compute_response

### 13.48.1 Detailed Description

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

There is only one **Graphics** object (dakotaGraphics) and it is global (for convenient access from strategies, models, and approximations).
13.48.2 Member Function Documentation

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).

References Variables::continuous_variable_labels(), Variables::cv(), Variables::discrete_int_variable_labels(),
Variables::discrete_real_variable_labels(), Variables::div(), Variables::drv(), Response::function_labels(), Graphics-::graphics2D, Response::num_functions(), Dakota::re_match(), and Graphics::win2dOn.

Referenced by SurrBasedMinimizer::initialize_graphics(), NonDReliability::initialize_graphics(), and Iterator-::initialize_graphics().

void create_tabular_datastream ( const Variables & vars, const Response & response, const std::string & tabular_data_file )

opens the tabular data file stream and prints the headings

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

References Graphics::tabularCntrLabel, Graphics::tabularDataFlag, and Graphics::tabularDataFStream.

Referenced by SurrBasedMinimizer::initialize_graphics(), and Iterator::initialize_graphics().

void add_datapoint ( const Variables & vars, const Response & response )

adds data to each window in the 2d graphics and adds a row to the tabular data file based on the results of a model
evaluation

Adds data to each 2d plot and each tabular data column (one for each active variable and for each response
function). graphicsCntr is used for the x axis in the graphics and the first column in the tabular data.

References Response::active_set_request_vector(), Variables::continuous_variables(), Variables::discrete_int-
variables(), Variables::discrete_real_variables(), Response::function_values(), Graphics::graphics2D, Graphics-::graphicsCntr, Graphics::tabularDataFlag, Graphics::tabularDataFStream, and Graphics::win2dOn.

Referenced by Model::compute_response(), NonDLocalReliability::mean_value(), SurrBasedLocalMinimizer-::minimize_surrogates(), Model::synchronize(), Model::synchronize_nowait(), and NonDLocalReliability::update_level_data().

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

adds data to a single window in the 2d graphics

Allows complete flexibility in defining other kinds of x-y plotting in the 2D graphics.

References Graphics::graphics2D, and Graphics::win2dOn.

void new_dataset ( int i )

creates a separate line graphic for subsequent data points for a single window in the 2d graphics

Used for displaying multiple data sets within the same plot.

References Graphics::graphics2D, and Graphics::win2dOn.

Referenced by NonDLocalReliability::update_level_data().

The documentation for this class was generated from the following files:
• DakotaGraphics.hpp
• DakotaGraphics.cpp

13.49 GridApplicInterface Class Reference

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

Inheritance diagram for GridApplicInterface:

```
Interface
      |          ApplicationInterface
      |                        ProcessApplicInterface
      |                          SysCallApplicInterface
      |                            GridApplicInterface
```

Public Member Functions

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

- **~GridApplicInterface** ()
  
  *destructor*

- void **derived_map** (const Variables &vars, const ActiveSet &set, Response &response, int fn_eval_id)
  
  *Called by map() and other functions to execute the simulation in synchronous mode. The portion of performing an evaluation that is specific to a derived class.*

- void **derived_map_asynch** (const ParamResponsePair &pair)
  
  *Called by map() and other functions to execute the simulation in asynchronous mode. The portion of performing an asynchronous evaluation that is specific to a derived class.*

- void **wait_local_evaluations** (PRPQueue &prp_queue)
  
  *For asynchronous function evaluations, this method is used to detect completion of jobs and process their results. It provides the processing code that is specific to derived classes. This version waits for at least one completion.*

- void **test_local_evaluations** (PRPQueue &prp_queue)
  
  *Convenience function for common code between wait and nowait case.*

- int **synchronous_local_analysis** (int analysis_id)

Protected Member Functions

- bool **grid_file_test** (const String &root_file)
  
  *test file(s) for existence based on root_file name*
13.50. HIERARCHSURRMODEL CLASS REFERENCE 457

Protected Attributes

- IntSet idSet
  
  Set of function evaluation id’s for active asynchronous system call evaluations.

- IntShortMap failCountMap
  
  map linking function evaluation id’s to number of response read failures

- start_grid_computing_t start_grid_computing
  
  handle to dynamically linked start_grid_computing function

- perform_analysis_t perform_analysis
  
  handle to dynamically linked perform_analysis grid function

- get_jobs_completed_t get_jobs_completed
  
  handle to dynamically linked get_jobs_completed grid function

- stop_grid_computing_t stop_grid_computing
  
  handle to dynamically linked stop_grid_computing function

13.49.1 Detailed Description

Derived application interface class which spawns simulation codes using grid services such as Condor or Globus. This class is currently a modified copy of SysCallApplicInterface adapted for use with an external grid services library which was dynamically linked using dlopen() services.

13.49.2 Member Function Documentation

int synchronous_local_analysis ( int analysis_id ) [inline], [virtual]

This code provides the derived function used by ApplicationInterface::serve_analyses_synch(). TODO - allow local analyses????

Reimplemented from ApplicationInterface.

References SysCallApplicInterface::spawn_analysis_to_shell().

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

- GridApplicInterface.hpp
- GridApplicInterface.cpp

13.50 HierarchSurrModel Class Reference

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

Inheritance diagram for HierarchSurrModel:

```
   Model
   /   /
  SurrogateModel
  /   /
HierarchSurrModel
```
Public Member Functions

- **HierarchSurrModel** (ProblemDescDB &problem_db)
  
  - * constructor
- **~HierarchSurrModel** ()
  
  - * destructor

Protected Member Functions

- **void derived_compute_response** (const ActiveSet &set)
  
  - * portion of compute_response() specific to HierarchSurrModel
- **void derived_async_compute_response** (const ActiveSet &set)
  
  - * portion of async_compute_response() specific to HierarchSurrModel
- **const IntResponseMap & derived_synchronize ()**
  
  - * portion of synchronize() specific to HierarchSurrModel
- **const IntResponseMap & derived_synchronize_nowait ()**
  
  - * portion of synchronize_nowait() specific to HierarchSurrModel
- **Model & surrogate_model ()**
  
  - * return lowFidelityModel
- **Model & truth_model ()**
  
  - * return highFidelityModel
- **void derived_subordinate_models** (ModelList &ml, bool recurse_flag)
  
  - * return lowFidelityModel and highFidelityModel
- **void primary_response_fn_weights** (const RealVector &wts, bool recurse_flag=true)
  
  - * set the relative weightings for multiple objective functions or least squares terms and optionally recurses into LF/HF models
- **void surrogate_response_mode** (short mode)
  
  - * set responseMode and pass any bypass request on to highFidelityModel for any lower-level surrogate recursions.
- **void surrogate_function_indices** (const IntSet &surr_fn_indices)
  
  - * (re)set the surrogate index set in SurrogateModel::surrogateFnIndices
- **void build_approximation ()**
  
  - * use highFidelityModel to compute the truth values needed for correction of lowFidelityModel results
- **void component_parallel_mode** (short mode)
  
  - * update component parallel mode for supporting parallelism in lowFidelityModel and highFidelityModel
- **void derived_init_communicators** (int max_eval_concurrency, bool recurse_flag=true)
  
  - * set up lowFidelityModel and highFidelityModel for parallel operations
- **void derived_init_serial ()**
  
  - * set up lowFidelityModel and highFidelityModel for serial operations.
- **void derived_set_communicators** (int max_eval_concurrency, bool recurse_flag=true)
  
  - * set active parallel configuration within lowFidelityModel and highFidelityModel
- **void derived_free_communicators** (int max_eval_concurrency, bool recurse_flag=true)
  
  - * deallocate communicator partitions for the HierarchSurrModel (request forwarded to lowFidelityModel and high-FidelityModel)
- **void serve** (int max_eval_concurrency)
Service lowFidelityModel and highFidelityModel job requests received from the master. Completes when a termination message is received from stop_serve().

- **void stop_servers()**
  
  Executed by the master to terminate lowFidelityModel and highFidelityModel server operations when iteration on the HierarchSurrModel is complete.

- **void inactive_view(short view, bool recurse_flag=true)**
  
  update the Model's inactive view based on higher level (nested) context and optionally recurse into

- **int evaluation_id() const**
  
  Return the current evaluation id for the HierarchSurrModel.

- **void set_evaluation_reference()**
  
  set the evaluation counter reference points for the HierarchSurrModel (request forwarded to lowFidelityModel and highFidelityModel)

- **void fine_grained_evaluation_counts()**
  
  request fine-grained evaluation reporting within lowFidelityModel and highFidelityModel

- **void print_evaluation_summary(ostream &s, bool minimal_header=false, bool relative_count=true) const**
  
  print the evaluation summary for the HierarchSurrModel (request forwarded to lowFidelityModel and highFidelityModel)

- **virtual void eval_tag_prefix(const String &eval_id_str)**
  
  set the hierarchical eval ID tag prefix

**Private Member Functions**

- **void update_model(Model &model)**
  
  update the incoming model (lowFidelityModel or highFidelityModel) with current variable values/bounds/labels

**Private Attributes**

- **int hierModelEvalCntr**
  
  number of calls to derived_compute_response() / derived_asynch_compute_response()

- **IntResponseMap cachedTruthRespMap**
  
  map of high-fidelity responses retrieved in derived_synchronize_nowait() that could not be returned since corresponding low-fidelity response portions were still pending.

- **Model lowFidelityModel**
  
  provides approximate low fidelity function evaluations. Model is of arbitrary type and supports recursions (e.g., lowFidelityModel can be a data fit surrogate on a low fidelity model).

- **Model highFidelityModel**
  
  provides truth evaluations for computing corrections to the low fidelity results. Model is of arbitrary type and supports recursions.

- **Response highFidRefResponse**
  
  the reference high fidelity response computed in build_approximation() and used for calculating corrections.

- **String evalTagPrefix**
  
  cached evalTag Prefix from parents to use at compute response time
Additional Inherited Members

13.50.1 Detailed Description

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

The **HierarchSurrModel** class manages hierarchical models of varying fidelity. In particular, it uses a low fidelity model as a surrogate for a high fidelity model. The class contains a lowFidelityModel which performs the approximate low fidelity function evaluations and a highFidelityModel which provides truth evaluations for computing corrections to the low fidelity results.

13.50.2 Member Function Documentation

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

portion of `compute_response()` specific to **HierarchSurrModel**

Compute the response synchronously using lowFidelityModel, highFidelityModel, or both (mixed case). For the lowFidelityModel portion, compute the high fidelity response if needed with `build_approximation()`, and, if correction is active, correct the low fidelity results.

Reimplemented from **Model**.

References **Response::active_set()**, **DiscrepancyCorrection::apply()**, **SurrogateModel::approxBuilds**, **SurrogateModel::asv_mapping()**, **HierarchSurrModel::build_approximation()**, **HierarchSurrModel::component_parallel_mode()**, **DiscrepancyCorrection::compute()**, **Model::compute_response()**, **DiscrepancyCorrection::computed()**, **Response::copy()**, **Model::current_response()**, **Model::currentResponse**, **Model::currentVariables**, **SurrogateModel::delta_Corr**, **Model::eval_tag_prefix()**, **HierarchSurrModel::evalTagPrefix**, **SurrogateModel::force_rebuild()**, **Model::hierarchicalTagging**, **HierarchSurrModel::highFidRefResponse**, **HierarchSurrModel::highFidelityModel**, **HierarchSurrModel::highFidRefResponse**, **HierarchSurrModel::lowFidelityModel**, **Model::outputLevel**, **ActiveSet::request_vector()**, **SurrogateModel::response_mapping()**, **SurrogateModel::responseMode**, **Response::update()**, and **HierarchSurrModel::update_model()**.

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

portion of `asynch_compute_response()` specific to **HierarchSurrModel**

Compute the response asynchronously using lowFidelityModel, highFidelityModel, or both (mixed case). For the lowFidelityModel portion, compute the high fidelity response with `build_approximation()` (for correcting the low fidelity results in `derived_synchronize()` and `derived_synchronize_nowait()`) if not performed previously.

Reimplemented from **Model**.

References **DiscrepancyCorrection::apply()**, **SurrogateModel::approxBuilds**, **SurrogateModel::asv_mapping()**, **Model::asynch_compute_response()**, **Model::asynch_flag()**, **HierarchSurrModel::build_approximation()**, **SurrogateModel::cachedApproxRespMap**, **HierarchSurrModel::cachedTruthRespMap**, **HierarchSurrModel::component_parallel_mode()**, **DiscrepancyCorrection::compute()**, **Model::compute_response()**, **DiscrepancyCorrection::computed()**, **Variables::copy()**, **Response::copy()**, **Model::current_response()**, **Model::currentVariables**, **SurrogateModel::deltaCorr**, **ActiveSet::derivative_vector()**, **Model::eval_tag_prefix()**, **HierarchSurrModel::evalTagPrefix**, **Model::evaluation_id()**, **SurrogateModel::force_rebuild()**, **Model::hierarchicalTagging**, **HierarchSurrModel::highFidRefResponse**, **HierarchSurrModel::highFidelityModel**, **HierarchSurrModel::highFidRefResponse**, **HierarchSurrModel::lowFidelityModel**, **Model::outputLevel**, **SurrogateModel::rawVarsMap**, **ActiveSet::request_vector()**, **SurrogateModel::responseMode**, **SurrogateModel::surrIdMap**, **SurrogateModel::truthIdMap**, and **HierarchSurrModel::update_model()**.

```cpp
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. 

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.

References DiscrepancyCorrection::apply(), SurrogateModel::cachedApproxRespMap, HierarchSurrModel::cachedTruthRespMap, HierarchSurrModel::component_parallel_mode(), DiscrepancyCorrection::compute(), DiscrepancyCorrection::computed(), SurrogateModel::deltaCorr, HierarchSurrModel::derived_synchronize_nowait(), HierarchSurrModel::highFidelityModel, HierarchSurrModel::highFidRefResponse, HierarchSurrModel::lowFidelityModel, Model::outputLevel, SurrogateModel::rawVarsMap, SurrogateModel::response_mapping(), SurrogateModel::responseMode, SurrogateModel::surrIdMap, SurrogateModel::surrResponseMap, Model::synchronize(), and SurrogateModel::truthIdMap.

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

Return the current evaluation id for the HierarchSurrModel.

return the hierarchical model evaluation count. Due to possibly intermittent use of surrogate bypass, this is not the same as either the loFi or hiFi model evaluation counts. It also does not distinguish duplicate evals.

Reimplemented from Model.

References HierarchSurrModel::hierModelEvalCntr.

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

- HierarchSurrModel.hpp
- HierarchSurrModel.cpp

13.51 Interface Class Reference

Base class for the interface class hierarchy.

Inheritance diagram for Interface:
Public Member Functions

- **Interface ()**
  
  default constructor

- **Interface (ProblemDescDB &problem_db)**
  
  standard constructor for envelope

- **Interface (const Interface &interface_in)**
  
  copy constructor

- **virtual ~Interface ()**
  
  destructor

- **Interface operator= (const Interface &interface_in)**
  
  assignment operator

- **virtual void map (const Variables &vars, const ActiveSet &set, Response &response, bool asynch_flag=false)**
  
  the function evaluator: provides a "mapping" from the variables to the responses.

- **virtual const IntResponseMap & synch ()**
  
  recovers data from a series of asynchronous evaluations (blocking)

- **virtual const IntResponseMap & synch_nowait ()**
  
  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, int max_eval_concurrency)**
  
  allocate communicator partitions for concurrent evaluations within an iterator and concurrent multiprocessor analyses within an evaluation.

- **virtual void set_communicators (const IntArray &message_lengths, int max_eval_concurrency)**
  
  set the local parallel partition data for an interface (the partitions are already allocated in ParallelLibrary).

- **virtual void free_communicators ()**
deallocate communicator partitions for concurrent evaluations within an iterator and concurrent multiprocessor analyses within an evaluation.

- virtual void init_serial ()
  reset certain defaults for serial interface objects.
- virtual int asynch_local_evaluation_concurrency () const
  return the user-specified concurrency for asynch local evaluations
- virtual String interface_synchronization () const
  return the user-specified interface synchronization
- virtual int minimum_points (bool constraint_flag) const
  returns the minimum number of points required to build a particular ApproximationInterface (used by DataFitSurfModels).
- virtual int recommended_points (bool constraint_flag) const
  returns the recommended number of points required to build a particular ApproximationInterface (used by DataFitSurfModels).
- virtual void approximation_function_indices (const IntSet &approx_fn_indices)
  set the (currently active) approximation function index set
- virtual void update_approximation (const Variables &vars, const IntResponsePair &response_pr)
  updates the anchor point for an approximation
- virtual void update_approximation (const RealMatrix &samples, const IntResponseMap &resp_map)
  updates the current data points for an approximation
- virtual void update_approximation (const VariablesArray &vars_array, const IntResponseMap &resp_map)
  updates the current data points for an approximation
- virtual void append_approximation (const Variables &vars, const IntResponsePair &response_pr)
  appends a single point to an existing approximation
- virtual void append_approximation (const RealMatrix &samples, const IntResponseMap &resp_map)
  appends multiple points to an existing approximation
- virtual void append_approximation (const VariablesArray &vars_array, const IntResponseMap &resp_map)
  appends multiple points to an existing approximation
- virtual void build_approximation (const RealVector &c_l_bnds, const RealVector &c_u_bnds, const IntVector &di_l_bnds, const IntVector &di_u_bnds, const RealVector &dr_l_bnds, const RealVector &dr_u_bnds)
  builds the approximation
- virtual void rebuild_approximation (const BoolDeque &rebuild_deque)
  rebuilds the approximation after a data update
- virtual void pop_approximation (bool save_surr_data)
  removes data from last append from the approximation
- virtual void restore_approximation ()
  restores the approximation to a selected previous state
- virtual bool restore_available ()
  queries the approximation for the ability to restore a previous increment
- virtual void finalize_approximation ()
  finalizes the approximation by applying all trial increments
- virtual void store_approximation ()
  move the current approximation into storage for later combination
• virtual void combine_approximation (short corr_type)
  combine the current approximation with one previously stored
• virtual void clear_current ()
  clears current data from an approximation interface
• virtual void clear_all ()
  clears all data from an approximation interface
• virtual void clear_saved ()
  clears saved data (from pop invocations) from an approximation interface
• virtual SharedApproxData & shared_approximation ()
  retrieve the SharedApproxData within an ApproximationInterface
• virtual std::vector<Approximation> & approximations ()
  retrieve the Approximations within an ApproximationInterface
• virtual const Pecos::SurrogateData & approximation_data (size_t index)
  retrieve the approximation data from a particular Approximation within an ApproximationInterface
• virtual const RealVectorArray & approximation_coefficients ()
  retrieve the approximation coefficients from each Approximation within an ApproximationInterface
• virtual void approximation_coefficients (const RealVectorArray &approx_coeffs)
  set the approximation coefficients within each Approximation within an ApproximationInterface
• virtual const RealVector & approximation_variances (const Variables &vars)
  retrieve the approximation variances from each Approximation within an ApproximationInterface
• virtual const StringArray & analysis_drivers () const
  retrieve the analysis drivers specification for application interfaces
• virtual bool evaluation_cache () const
  return flag indicating usage of the global evaluation cache
• virtual void file_cleanup () const
  clean up any interface parameter/response files when aborting
• void assign_rep (Interface *interface_rep, bool ref_count_incr=true)
  replaces existing letter with a new one
• const String & interface_type () const
  returns the interface type
• const String & interface_id () const
  returns the interface identifier
• int evaluation_id () const
  returns the value of the (total) evaluation id counter for the interface
• void fine_grained_evaluation_counters (size_t num_fns)
  set fineGrainEvalCounters to true and initialize counters if needed
• void init_evaluation_counters (size_t num_fns)
  initialize fine grained evaluation counters
• void set_evaluation_reference ()
  set evaluation count reference points for the interface
• void print_evaluation_summary (std::ostream &s, bool minimal_header, bool relative_count) const
print an evaluation summary for the interface

- bool multi_proc_eval () const
  returns a flag signaling the use of multiprocessor evaluation partitions
- bool iterator_eval_dedicated_master () const
  returns a flag signaling the use of a dedicated master processor at the iterator-evaluation scheduling level
- bool is_null () const
  function to check interfaceRep (does this envelope contain a letter?)
- void eval_tag_prefix (const String &eval_id_str, bool append_iface_id=true)
  set the evaluation tag prefix (does not recurse)

Protected Member Functions

- Interface (BaseConstructor, const ProblemDescDB &problem_db)
  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)
- Interface (NoDBBaseConstructor, size_t num_fns, short output_level)
  constructor initializes the base class part of letter classes (NoDBBaseConstructor used for on the fly instantiations without a DB)
- void init_algebraic_mappings (const Variables &vars, const Response &response)
  Define algebraicACVIndices, algebraicACVIds, and algebraicFnIndices.
- void asv_mapping (const ActiveSet &total_set, ActiveSet &algebraic_set, ActiveSet &core_set)
  define the evaluation requirements for algebraic_mappings() (algebraic_set) and the core Application/Approximation mapping (core_set) from the total Interface evaluation requirements (total_set)
- void asv_mapping (const ActiveSet &algebraic_set, ActiveSet &total_set)
  map an algebraic ASV back to original total ordering for asynch recovery
- void algebraic_mappings (const Variables &vars, const ActiveSet &algebraic_set, Response &algebraic_response)
  evaluate the algebraic_response using the AMPL solver library and the data extracted from the algebraic_mappings file
- void response_mapping (const Response &algebraic_response, const Response &core_response, Response &total_response)
  combine the response from algebraic_mappings() with the response from derived_map() to create the total response
- String final_eval_id_tag (int fn_eval_id)
  form and return the final evaluation ID tag, appending iface ID if needed

Protected Attributes

- String interfaceType
  the interface type: system, fork, direct, grid, or approximation
- String interfaceId
  the interface specification identifier string from the DAKOTA input file
- bool algebraicMappings
  flag for the presence of algebraic_mappings that define the subset of an Interface’s parameter to response mapping that is explicit and algebraic.
- bool coreMappings
flag for the presence of non-algebraic mappings that define the core of an Interface’s parameter to response mapping using analysis drivers for ApplicationInterface or functionSurfaces for ApproximationInterface).

- int currEvalId
  identifier for the current evaluation, which may differ from the evaluation counters in the case of evaluation scheduling; used on iterator master as well as server processors. Currently, this is set prior to all invocations of derived_map() for all processors.

- bool fineGrainEvalCounters
  controls use of fn val/grad/hess counters

- int evalIdCntr
  total interface evaluation counter

- int newEvalIdCntr
  new (non-duplicate) interface evaluation counter

- int evalIdRefPt
  iteration reference point for evalIdCntr

- int newEvalIdRefPt
  iteration reference point for newEvalIdCntr

- IntArray fnValCounter
  number of value evaluations by resp fn

- IntArray fnGradCounter
  number of gradient evaluations by resp fn

- IntArray fnHessCounter
  number of Hessian evaluations by resp fn

- IntArray newFnValCounter
  number of new value evaluations by resp fn

- IntArray newFnGradCounter
  number of new gradient evaluations by resp fn

- IntArray newFnHessCounter
  number of new Hessian evaluations by resp fn

- IntArray fnValRefPt
  iteration reference point for fnValCounter

- IntArray fnGradRefPt
  iteration reference point for fnGradCounter

- IntArray fnHessRefPt
  iteration reference point for fnHessCounter

- IntArray newFnValRefPt
  iteration reference point for newFnValCounter

- IntArray newFnGradRefPt
  iteration reference point for newFnGradCounter

- IntArray newFnHessRefPt
  iteration reference point for newFnHessCounter

- IntResponseMap rawResponseMap
  Set of responses returned after either a blocking or nonblocking schedule of asynchronous evaluations.

- StringArray fnLabels
response function descriptors from the DAKOTA input file (used in print_evaluation_summary() and derived direct interface classes)

- **bool multiProcEvalFlag**
  flag for multiprocessor evaluation partitions (evalComm)

- **bool ieDedMasterFlag**
  flag for dedicated master partitioning at the iterator level

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

- **String evalTagPrefix**
  set of period-delimited evaluation ID tags to use in evaluation tagging

- **bool appendIfaceId**
  whether to append the interface ID to the prefix during map (default true)

### Private Member Functions

- **Interface * get_interface (ProblemDescDB &problem_db)**
  Used by the envelope to instantiate the correct letter class.

- **int algebraic_function_type (String)**
  Used by algebraic mappings to determine the correct AMPL function evaluation call to make.

### Private Attributes

- **StringArray algebraicVarTags**
  set of variable tags from AMPL stub.col

- **SizetArray algebraicACVIndices**
  set of indices mapping AMPL algebraic variables to DAKOTA all continuous variables

- **SizetArray algebraicACVIds**
  set of ids mapping AMPL algebraic variables to DAKOTA all continuous variables

- **StringArray algebraicFnTags**
  set of function tags from AMPL stub.row

- **IntArray algebraicFnTypes**
  function type: > 0 = objective, < 0 = constraint |value|-1 is the objective (constraint) index when making AMPL objval (conival) calls

- **SizetArray algebraicFnIndices**
  set of indices mapping AMPL algebraic objective functions to DAKOTA response functions

- **RealArray algebraicConstraintWeights**
  set of weights for computing Hessian matrices for algebraic constraints;

- **int numAlgebraicResponses**
  number of algebraic responses (objectives+constraints)

- **Interface * interfaceRep**
  pointer to the letter (initialized only for the envelope)

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

- **ASL * asl**
  pointer to an AMPL solver library (ASL) object
13.51.1 Detailed Description

Base class for the interface class hierarchy.

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

13.51.2 Constructor & Destructor Documentation

Interface ( )
default constructor
used in Model envelope class instantiations

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

References Dakota::abort_handler(), Interface::get_interface(), and Interface::interfaceRep.

Interface ( const Interface & interface_in )
copy constructor
Copy constructor manages sharing of interfaceRep and incrementing of referenceCount.

References Interface::interfaceRep, and Interface::referenceCount.

~Interface ( ) [virtual]
destructor
Destructor decrements referenceCount and only deletes interfaceRep if referenceCount is zero.

References Interface::interfaceRep, and Interface::referenceCount.

Interface ( BaseConstructor, const ProblemDescDB & problem_db ) [protected]
constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

This constructor is the one which must build the base class data for all inherited interfaces. get_interface() instantiates a derived class letter and the derived constructor selects this base class constructor in its initialization list (to avoid the recursion of the base class constructor calling get_interface() again). Since this is the letter and the letter IS the representation, interfaceRep is set to NULL (an uninitialized pointer causes problems in ~Interface).

References Dakota::abort_handler(), Interface::algebraic_function_type(), Interface::algebraicConstraintWeights, Interface::algebraicFnTags, Interface::algebraicFnTypes, Interface::algebraicMappings, Interface::algebraicVarTags, Interface::asl, Interface::fineGrainEvalCounters, Interface::fnLabels, ProblemDescDB::get_sa(), ProblemDescDB::get_string(), Interface::init_evaluation_counters(), Interface::outputLevel, and Dakota::strends().
13.51.3 Member Function Documentation

**Interface operator= ( const Interface & interface_in )**

assignment operator


References Interface::interfaceRep, and Interface::referenceCount.

**void assign_rep ( Interface * interface_rep, bool ref_countincr = 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_countincr:

- ref_countincr = 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_countincr = false: the incoming letter is instantiated on the fly and has no envelope. This case is modeled after get_interface(): a letter is dynamically allocated using new and passed into assign_rep, the letter’s reference count is not incremented, and the letter is not remotely deleted (its memory management is passed over to the envelope).

References Dakota::abort_handler(), Interface::interfaceRep, and Interface::referenceCount.

Referenced by DataFitSurrModel::DataFitSurrModel(), parallel_interface_plugin(), LibraryEnvironment::plugin_interface(), and run_dakota().

**void eval_tag_prefix ( const String & eval_id_str, bool append_iface_id = true )**

set the evaluation tag prefix (does not recurse)

default implementation just sets the list of eval ID tags; derived classes containing additional models or interfaces should override (currently no use cases)

References Interface::append_ifaceId, Interface::eval_tag_prefix(), Interface::evalTagPrefix, and Interface::interfaceRep.

Referenced by NestedModel::derived_compute_response(), SingleModel::eval_tag_prefix(), and Interface::eval_tag_prefix().

**void response_mapping ( const Response & algebraic_response, const Response & core_response, Response & total_response ) [protected]**

combine the response from algebraic_mappings() with the response from derived_map() to create the total response

This function will get invoked even when only algebraic mappings are active (no core mappings from derived_map), since the AMPL algebraic_response may be ordered differently from the total_response. In this case, the core_response object is unused.

References Dakota::NPOS, Dakota::abort_handler(), Response::active_set_derivative_vector(), Response::active_set_request_vector(), Interface::algebraicACVIds, Interface::algebraicFnIndices, Interface::coreMappings, Dakota::find_index(), Response::function_gradient(), Response::function_gradient_view(), Response::function_gradients(),
CHAPTER 13. CLASS DOCUMENTATION

Response::function_hessian(), Response::function_hessian_view(), Response::function_hessians(), Response::function_value(), Response::function_values(), Response::function_values_view(), Interface::outputLevel, Response::reset(), and Response::reset_inactive().

Referenced by ApproximationInterface::map(), ApplicationInterface::map(), ApplicationInterface::synch(), and ApplicationInterface::synch_nowait().

**Interface * get_interface ( ProblemDescDB & problem_db ) [private]**

Used by the envelope to instantiate the correct letter class.

used only by the envelope constructor to initialize interfaceRep to the appropriate derived type.

References ProblemDescDB::get_string(), and Interface::interface_type().

Referenced by Interface::Interface().

13.51.4 Member Data Documentation

**IntResponseMap rawResponseMap [protected]**

Set of responses returned after either a blocking or nonblocking schedule of asynchronous evaluations.

The map is a full/partial set of completions which are identified through their evalIdCntr key. The raw set is postprocessed (i.e., finite diff grads merged) in Model::synchronize() where it becomes responseMap.

Referenced by ApplicationInterface::asynchronous_local_evaluations(), ApplicationInterface::process_asynch_local(), ApplicationInterface::process_synch_local(), ApplicationInterface::receive_evaluation(), ApplicationInterface::synch(), ApproximationInterface::synch(), ApplicationInterface::synch_nowait(), ApproximationInterface::synch_nowait(), ApplicationInterface::test_local_backfill(), and ApplicationInterface::test_receives_backfill().

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

- DakotaInterface.hpp
- DakotaInterface.cpp

13.52 Iterator Class Reference

Base class for the iterator class hierarchy.

Inheritance diagram for Iterator:

```
                   Iterator
                   |    Analyzer    |    Minimizer    |
                   |                |                |
                   NonD           CollabHybridMetaIterator | LeastSq
                   PithyDACE       ConcurrentMetaIterator   Optimizer
                   Verification  EmbedHybridMetaIterator  SurrBasedMinimizer
                   SeqHybridMetaIterator
```

**Public Member Functions**

- **Iterator ()**
  
  *default constructor*

- **Iterator (ProblemDescDB & problem_db)**
standard envelope constructor, which constructs its own model(s)

- **Iterator** (ProblemDescDB &problem_db, Model &model)
  alternate envelope constructor which uses the ProblemDescDB but accepts a model from a higher level (meta-
  iterator) context, instead of constructing its own

- **Iterator** (const String &method_string, Model &model)
  alternate envelope constructor for instantiations by name without the ProblemDescDB

- **Iterator** (const Iterator &iterator)
  copy constructor

- virtual ~Iterator ()
  destructor

- **Iterator operator=** (const Iterator &iterator)
  assignment operator

- virtual void initialize_run ()
  utility function to perform common operations prior to pre_run(); typically memory initialization; setting of instance
  pointers

- virtual void pre_run ()
  pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a
  priori

- virtual void core_run ()
  core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post

- virtual void post_run (std::ostream &s)
  post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/
  Responses and perform final analysis phase in a standalone way

- virtual void finalize_run ()
  utility function to perform common operations following post_run(); deallocation and resetting of instance pointers

- virtual void pre_output ()
  write variables to file, following pre-run

- virtual void post_input ()
  read tabular data for post-run mode

- virtual void init_communicators ()
  initialize the communicators associated with iteratedModel as well as any subordinate iterators or subordinate
  models

- virtual void set_communicators ()
  set the communicators associated with iteratedModel

- virtual void free_communicators ()
  free the communicators associated with iteratedModel as well as any subordinate iterators or subordinate models

- virtual void reset ()
  restore initial state for repeated sub-iterator executions

- virtual void initialize_iterator (int job_index)
  used by IteratorScheduler to set the starting data for a run

- virtual void pack_parameters_buffer (MPIPackBuffer &send_buffer, int job_index)
  used by IteratorScheduler to pack starting data for an iterator run

- virtual void unpack_parameters_buffer (MPIUnpackBuffer &recv_buffer)
  used by IteratorScheduler to unpack starting data for an iterator run
• virtual void pack_results_buffer (MPIPackBuffer &send_buffer, int job_index)
  
  used by IteratorScheduler to pack results data from an iterator run

• virtual void unpack_results_buffer (MPIUnpackBuffer &recv_buffer, int job_index)
  
  used by IteratorScheduler to unpack results data from an iterator run

• virtual void update_local_results (int job_index)

  used by IteratorScheduler to update local results arrays

• virtual const Variables & variables_results () const

  return a single final iterator solution (variables)

• virtual const Response & response_results () const

  return a single final iterator solution (response)

• virtual const VariablesArray & variables_array_results ()

  return multiple final iterator solutions (variables). This should only be used if returns_multiple_points() returns true.

• virtual const ResponseArray & response_array_results ()

  return multiple final iterator solutions (response). This should only be used if returns_multiple_points() returns true.

• virtual bool accepts_multiple_points () const

  indicates if this iterator accepts multiple initial points. Default return is false. Override to return true if appropriate.

• virtual bool returns_multiple_points () const

  indicates if this iterator returns multiple final points. Default return is false. Override to return true if appropriate.

• virtual void initial_points (const VariablesArray &pts)

  sets the multiple initial points for this iterator. This should only be used if accepts_multiple_points() returns true.

• virtual void response_results_active_set (const ActiveSet &set)

  set the requested data for the final iterator response results

• virtual void initialize_graphics (int iterator_server_id=1)

  initialize the 2D graphics window and the tabular graphics data

• virtual void print_results (std::ostream &s)

  print the final iterator results

• virtual const Model & algorithm_space_model () const

  return the result of any recasting or surrogate model recursion layered on top of iteratedModel by the derived Iterator ctor chain

• virtual unsigned short uses_method () const

  return name of any enabling iterator used by this iterator

• virtual void method_recourse ()

  perform a method switch, if possible, due to a detected conflict

• virtual const VariablesArray & all_variables ()

  return the complete set of evaluated variables

• virtual const RealMatrix & all_samples ()

  return the complete set of evaluated samples

• virtual const IntResponseMap & all_responses () const

  return the complete set of computed responses

• virtual int num_samples () const

  get the current number of samples

• virtual void sampling_reset (int min_samples, bool all_data_flag, bool stats_flag)
reset sampling iterator to use at least min_samples

- virtual void sampling_reference (int samples_ref)
  set reference number of samples, which is a lower bound during reset

- virtual unsigned short sampling_scheme () const
  return sampling name

- virtual bool compact_mode () const
  returns Analyzer::compactMode

- void run (std::ostream &s)
  orchestrate initialize/pre/core/post/finalize phases

- void assign_re (Iterator *iterator_rep, bool ref_count=true)
  replaces existing letter with a new one

- void iterated_model (const Model &model)
  set the iteratedModel (iterators and meta-iterators using a single model instance)

- Model & iterated_model ()
  return the iteratedModel (iterators & meta-iterators using a single model instance)

- ProblemDescDB & problem_description_db () const
  return the problem description database (probDescDB)

- void method_name (unsigned short m_name)
  set the method name to an enumeration value

- unsigned short method_name () const
  return the method name via its native enumeration value

- void method_string (const String &m_str)
  set the method name by string

- String method_string () const
  return the method name by string

- String method_enum_to_string (unsigned short method_name) const
  convert a method name enumeration value to a string

- unsigned short method_string_to_enum (const String &method_name) const
  convert a method name string to an enumeration value

- String submethod_enum_to_string (unsigned short submethod_name) const
  convert a method name enumeration value to a string

- const String & method_id () const
  return the method identifier (methodId)

- int maximum_evaluation_concurrency () const
  return the maximum evaluation concurrency supported by the iterator

- void maximum_evaluation_concurrency (int max_conc)
  set the maximum evaluation concurrency supported by the iterator

- void convergence_tolerance (Real conv_tol)
  set the method convergence tolerance (convergenceTol)

- Real convergence_tolerance () const
  return the method convergence tolerance (convergenceTol)

- void output_level (short out_lev)
  set the method output level (outputLevel)
• short output_level () const
  return the method output level (outputLevel)

• void summary_output (bool summary_output_flag)
  Set summary output control; true enables evaluation/results summary.

• size_t num_final_solutions () const
  return the number of solutions to retain in best variables/response arrays

• void num_final_solutions (size_t num_final)
  set the number of solutions to retain in best variables/response arrays

• void active_set (const ActiveSet &set)
  set the default active set vector (for use with iterators that employ evaluate_parameter_sets())

• const ActiveSet & active_set () const
  return the default active set vector (used by iterators that employ evaluate_parameter_sets())

• void sub_iterator_flag (bool si_flag)
  set subIteratorFlag (and update summaryOutputFlag if needed)

• void active_variable_mappings (const SizetArray &c_index1, const SizetArray &di_index1, const SizetArray &dr_index1, const ShortArray &c_target2, const ShortArray &di_target2, const ShortArray &dr_target2)
  set primaryA{CV, DIV, DRV}MapIndices, secondaryA{CV, DIV, DRV}MapTargets

• bool is_null () const
  function to check iteratorRep (does this envelope contain a letter?)

• Iterator * iterator_rep () const
  returns iteratorRep for access to derived class member functions that are not mapped to the top Iterator level

• virtual void eval_tag_prefix (const String &eval_id_str)
  set the hierarchical eval ID tag prefix

Protected Member Functions

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

• Iterator (NoDBBaseConstructor, unsigned short method_name)
  alternate constructor for base iterator classes constructed on the fly

• virtual void update_from_model (const Model &model)
  set inherited data attributes based on extractions from incoming model

• virtual const VariablesArray & initial_points () const
  gets the multiple initial points for this iterator. This will only be meaningful after a call to initial_points mutator.

• StrStrSizet run_identifier () const
  get the unique run identifier based on method name, id, and number of executions
Protected Attributes

- **ProblemDescDB** & **probDescDB**
  class member reference to the problem description database
- **Model iteratedModel**
  the model to be iterated (for iterators and meta-iterators employing a single model instance)
- **unsigned short 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 maxEvalConcurrency**
  maximum number of concurrent model evaluations
- **ActiveSet activeSet**
  the response data requirements on each function evaluation
- **size_t numFinalSolutions**
  number of solutions to retain in best variables/response arrays
- **VariablesArray bestVariablesArray**
  collection of N best solution variables found during the study; always in context of Model originally passed to the Iterator (any in-flight Recasts must be undone)
- **ResponseArray bestResponseArray**
  collection of N best solution responses found during the study; always in context of Model originally passed to the Iterator (any in-flight Recasts must be undone)
- **bool subIteratorFlag**
  flag indicating if this Iterator is a sub-iterator (NestedModel::subIterator or DataFitSurrModel::daceIterator)
- **SizetArray primaryACVarMapIndices**
  “primary” all continuous variable mapping indices flowed down from higher level iteration
- **SizetArray primaryADIVarMapIndices**
  “primary” all discrete int variable mapping indices flowed down from higher level iteration
- **SizetArray primaryADRVarMapIndices**
  “primary” all discrete real variable mapping indices flowed down from higher level iteration
- **ShortArray secondaryACVarMapTargets**
  “secondary” all continuous variable mapping targets flowed down from higher level iteration
- **ShortArray secondaryADIVarMapTargets**
  “secondary” all discrete int variable mapping targets flowed down from higher level iteration
- **ShortArray secondaryADRVarMapTargets**
  “secondary” all discrete real variable mapping targets flowed down from higher level iteration
- **short outputLevel**
  output verbosity level: {SILENT,QUIET,NORMAL,VERBOSE,DEBUG}.OUTPUT
- **bool summaryOutputFlag**
flag for summary output (evaluation stats, final results); default true, but false for on-the-fly (helper) iterators and sub-iterator use cases

- **ResultsManager & resultsDB**
  reference to the global iterator results database

- **ResultsNames resultsNames**
  valid names for iterator results

### Private Member Functions

- **Iterator * get_iterator (ProblemDescDB &problem_db)**
  Used by the envelope to instantiate the correct letter class.

- **Iterator * get_iterator (ProblemDescDB &problem_db, Model &model)**
  Used by the envelope to instantiate the correct letter class.

- **Iterator * get_iterator (const String &method_string, Model &model)**
  Used by the envelope to instantiate the correct letter class.

### Private Attributes

- **String methodId**
  method identifier string from the input file

- **size_t execNum**
  an execution number for this instance of the class, unique across all instances of same methodName/methodId

- **Iterator * iteratorRep**
  pointer to the letter (initialized only for the envelope)

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

### 13.52.1 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.

### 13.52.2 Constructor & Destructor Documentation

**Iterator ( )**

default constructor

The default constructor is used in Vector<Iterator> instantiations and for initialization of **Iterator** objects contained in meta-Iterators and **Model** recursions. iteratorRep is NULL in this case, making it necessary to check for NULL pointers in the copy constructor, assignment operator, and destructor.

Referenced by SurrBasedGlobalMinimizer::SurrBasedGlobalMinimizer(), and SurrBasedLocalMinimizer::SurrBasedLocalMinimizer().
Iterator ( ProblemDescDB & problem_db )

standard envelope constructor, which constructs its own model(s)

Envelope constructor only needs to extract enough data to properly execute get_iterator(), since letter holds the actual base class data. This version is used for top-level ProblemDescDB-driven construction of all Iterators and MetaIterators, which construct their own Model instances.

References Dakota::abort_handler, Iterator::get_iterator(), and Iterator::iteratorRep.

Iterator ( ProblemDescDB & problem_db, Model & model )

alternate envelope constructor which uses the ProblemDescDB but accepts a model from a higher level (meta-iterator) context, instead of constructing its own

Envelope constructor only needs to extract enough data to properly execute get_iterator(), since letter holds the actual base class data. This version is used for ProblemDescDB-driven construction of Iterators that are passed a Model from a higher-level context (e.g., a MetaIterator instantiates its sub-iterator(s) by name instead of pointer and passes in its iteratedModel, since these sub-iterators lack their own model pointers).

References Dakota::abort_handler(), Iterator::get_iterator(), and Iterator::iteratorRep.

Iterator ( const String & method_string, Model & model )

alternate envelope constructor for instantiations by name without the ProblemDescDB

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. This version is used for lightweight constructions without the ProblemDescDB.

References Dakota::abort_handler(), Iterator::get_iterator(), and Iterator::iteratorRep.

Iterator ( const Iterator & iterator )

copy constructor

Copy constructor manages sharing of iteratorRep and incrementing of referenceCount.

References Iterator::iteratorRep, and Iterator::referenceCount.

~Iterator ( ) [virtual]

destructor

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

References Iterator::iteratorRep, and Iterator::referenceCount.

Iterator ( BaseConstructor, ProblemDescDB & problem_db ) [protected]

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

This constructor builds the base class data for all inherited iterators, including meta-iterators. get_iterator() instantiates a derived class and the derived class selects this base class constructor in its initialization list (to avoid the recursion of the base class constructor calling get_iterator() again). Since the letter IS the representation, its representation pointer is set to NULL (an uninitialized pointer causes problems in ~Iterator).

References Iterator::method_enum_to_string(), Iterator::methodName, and Iterator::outputLevel.
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, and is not used for construction of meta-iterators. 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.

13.52.3 Member Function Documentation

Iterator operator= ( const Iterator & iterator )

assignment operator


References Iterator::iteratorRep, and Iterator::referenceCount.

void initialize_run ( ) [virtual]

utility function to perform common operations prior to pre_run(); typically memory initialization; setting of instance pointers

Perform initialization phases of run sequence, like allocating memory and setting instance pointers. Commonly used in sub-iterator executions. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s initialize_run(), typically before performing its own implementation steps.

Reimplemented in SNLLOptimizer, NLPQLPOptimizer, NonD, SNLLLeastSq, Analyzer, Minimizer, CONMINOptimizer, DOTOptimizer, Optimizer, and LeastSq.

References Iterator::initialize_run(), and Iterator::iteratorRep.

Referenced by Iterator::initialize_run(), Iterator::run(), and SeqHybridMetaIterator::run_sequential_adaptive().

void pre_run ( ) [virtual]

pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori

pre-run phase, which a derived iterator may optionally reimplement; when not present, pre-run is likely integrated into the derived run function. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s pre_run(), if implemented, typically before performing its own implementation steps.

Reimplemented in NonDLHSSampling, DDACEDesignCompExp, ParamStudy, FSUDesignCompExp, and PSUADesignCompExp.

References Iterator::iteratorRep, and Iterator::pre_run().

Referenced by ParamStudy::pre_run(), Iterator::pre_run(), and Iterator::run().

void core_run ( ) [virtual]

core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post Virtual run function for the iterator class hierarchy. All derived classes need to redefine it.

Reimplemented in NonD, SeqHybridMetaIterator, Optimizer, ConcurrentMetaIterator, EmbedHybridMetaIterator, CollabHybridMetaIterator, LeastSq, PStudyDACE, Verification, and SurrBasedMinimizer.

References Dakota::abort_handler(), Iterator::core_run(), and Iterator::iteratorRep.

Referenced by Iterator::core_run(), and Iterator::run().
void post_run ( std::ostream & s ) [virtual]

post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way

Post-run phase, which a derived iterator may optionally reimplement; when not present, post-run is likely integrated into run. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s post_run(), typically after performing its own implementation steps.

Reimplemented in SNLLOptimizer, COLINOptimizer, SNLLLeastSq, Analyzer, Minimizer, NonDLHSSampling, Optimizer, DDACEDesignCompExp, ParamStudy, FSUDesignCompExp, MetaIterator, LeastSq, and PSUADE-DesignCompExp.

References Iterator::iteratorRep, and Iterator::post_run().

Referenced by Iterator::post_run(), and Iterator::run().

void finalize_run ( ) [virtual]

utility function to perform common operations following post_run(): deallocation and resetting of instance pointers

Optional: perform finalization phases of run sequence, like deallocating memory and resetting instance pointers. Commonly used in sub-iterator executions. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s finalize_run(), typically after performing its own implementation steps.

Reimplemented in SNLLOptimizer, SNLLLeastSq, NonD, Minimizer, Optimizer, and LeastSq.

References Iterator::finalize_run(), and Iterator::iteratorRep.

Referenced by Minimizer::finalize_run(), Iterator::finalize_run(), NonD::finalize_run(), Iterator::run(), and SeqHybridMetaIterator::run_sequential_adaptive().

void initialize_graphics ( int iterator_server_id = 1 ) [virtual]

initialize the 2D graphics window and the tabular graphics data

This is a convenience function for encapsulating graphics initialization operations.

Reimplemented in NonDReliability, and SurrBasedMinimizer.

References Model::auto_graphics(), Graphics::create_plots_2d(), Graphics::create_tabular_datastream(), Model::current_response(), Model::current_variables(), Dakota::dakota_graphics, OutputManager::graph2DFlag, Iterator::initialize_graphics(), Iterator::iteratedModel, Iterator::iteratorRep, ParallelLibrary::output_manager(), Model::parallel_library(), OutputManager::tabularDataFile, and OutputManager::tabularDataFlag.

Referenced by CollabHybridMetaIterator::core_run(), EmbedHybridMetaIterator::core_run(), ConcurrentMetaIterator::core_run(), Environment::execute(), Iterator::initialize_graphics(), SeqHybridMetaIterator::run_sequential(), and SeqHybridMetaIterator::run_sequential_adaptive().

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, NonDLHSSampling, NonDPOFDarts, NonDLLocalReliability, NonDAdaptiveSampling, NonDGPlmpSampling, Optimizer, SeqHybridMetaIterator, ConcurrentMetaIterator, NonDAdaptImpSampling, NonDIncrmlHSSampling, NonDGlobalReliability, NonDInterval, NonDExpansion, LeastSq, PStudyDAE, RichExtrapVerification, Verification, and SurrBasedMinimizer.

References Iterator::iteratorRep, and Iterator::print_results().

Referenced by MetaIterator::post_run(), Minimizer::post_run(), Iterator::print_results(), and EfficientSubspaceMethod::reduced_space_uq().
**unsigned short uses_method ( ) const [virtual]**

return name of any enabling iterator used by this iterator

This is used to avoid clashes in state between non-object-oriented (i.e., F77, C) iterator executions, when such iterators could potentially be executing simultaneously (e.g., nested execution). It is not an issue (and a used method is not reported) in cases where a helper execution is completed before a lower level one could be initiated; an example of this is DIRECT for maximization of expected improvement: the EIF maximization is completed before a new point evaluation (which could include nested iteration) is performed.

Reimplemented in NonDLocalReliability, and NonDLocalInterval.

References Iterator::iteratorRep, and Iterator::uses_method().

Referenced by DOTOptimizer::initialize(), CONMINOptimizer::initialize(), NCSUOptimizer::initialize(), NLPQLPOptimizer::initialize(), NonDLocalInterval::NonDLocalInterval(), NonDLocalReliability::NonDLocalReliability(), SOLBase::SOLBase(), and Iterator::uses_method().

**void run ( std::ostream & s )**

orchestrate initialize/pre/core/post/finalize phases

Iterator supports a construct/initialize-run/pre-run/core-run/post-run/finalize-run/destroy progression. This member (non-virtual) function sequences these run phases; it accepts an ostream, but controls verbosity with outputLevel.

References ParallelLibrary::command_line_post_run(), ParallelLibrary::command_line_pre_run(), ParallelLibrary::command_line_run(), Iterator::core_run(), Iterator::execNum, Iterator::finalize_run(), ResultsID::increment_id(), Iterator::initialize_run(), ResultsID::instance(), Iterator::iteratedModel, Iterator::iteratorRep, Iterator::method_enum_to_string(), Iterator::method_id(), Iterator::method_string(), Iterator::methodName, Iterator::outputLevel, Model::parallel_library(), Iterator::post_input(), Iterator::post_run(), Iterator::pre_output(), Iterator::pre_run(), Iterator::run(), and Iterator::summaryOutputFlag.

Referenced by DataFitSurrModel::build_global(), NonExpansion::compute_statistics(), NestedModel::derived_compute_response(), Environment::execute(), EfficientSubspaceMethod::generate_fullspace_samples(), NonDGlobalReliability::importance_sampling(), SurrBasedLocalMinimizer::minimize_surrogates(), SurrBasedGlobalMinimizer::minimize_surrogates(), EffGlobalMinimizer::minimize_surrogates_on_model(), NonDLocalReliability::mpp_search(), NonDGlobalReliability::optimize_gaussian_process(), GaussProcApproximation::optimize_theta_global(), GaussProcApproximation::optimize_theta_multipoint(), NonDAdaptiveSampling::pick_new_candidates(), NonDLocalReliability::probability(), NonDLHSInterval::quantify_uncertainty(), NonDBayesCalibration::quantify_uncertainty(), NonDLocalInterval::quantify_uncertainty(), NonDGlobalInterval::quantify_uncertainty(), NonDAdaptiveSampling::quantify_uncertainty(), NonDGPImpSampling::quantify_uncertainty(), NonDGPSABayesSampling::quantify_uncertainty(), EfficientSubspaceMethod::reduced_space_uq(), SurrBasedLocalMinimizer::relax_constraints(), Iterator::run(), and IteratorScheduler::run_iterator().

**void assign_rep ( Iterator * iterator_rep, bool ref_count_incr = true )**

replaces existing letter with a new one

Similar to the assignment operator, the assign_rep() function decrements referenceCount for the old iteratorRep and assigns the new iteratorRep. It is different in that it is used for publishing derived class letters to existing envelopes, as opposed to sharing representations among multiple envelopes (in particular, assign_rep is passed a letter object and operator= is passed an envelope object). Letter assignment supports two models as governed by ref_count_incr:

- ref_count_incr = true (default): the incoming letter belongs to another envelope. In this case, increment the reference count in the normal manner so that deallocation of the letter is handled properly.
• ref_count_incr = false: the incoming letter is instantiated on the fly and has no envelope. This case is modeled after get_iterator(): a letter is dynamically allocated using new and passed into assign_rep, the letter’s reference count is not incremented, and the letter is not remotely deleted (its memory management is passed over to the envelope).

References Dakota::abort_handler(), Iterator::iterator_rep(), Iterator::iteratorRep, and Iterator::referenceCount.

Referenced by NonDExpansion::construct_cubature(), NonDExpansion::construct_expansion_sampler(), NonDAdaptiveSampling::construct_fsu_sampler(), NonD::construct_lhs(), NonDExpansion::construct_quadrature(), NonDExpansion::construct_sparse_grid(), EffGlobalMinimizer::EffGlobalMinimizer(), EfficientSubspaceMethod::init_fullspace_sampler(), NonDLocalInterval::method_recourse(), NonDLocalReliability::method_recourse(), NonDAdaptiveSampling::NonDAdaptiveSampling(), NonDBayesCalibration::NonDBayesCalibration(), NonDGGlobalInterval::NonDGGlobalInterval(), NonDGGlobalReliability::NonDGGlobalReliability(), NonDGPImpSampling::NonDGPImpSampling(), NonDGPMSABayesCalibration::NonDGPMSABayesCalibration(), NonDLHSInterval::NonDLHSInterval(), NonDLocalInterval::NonDLocalInterval(), NonDLocalReliability::NonDLocalReliability(), GaussProcApproximation::optimize_theta_global(), GaussProcApproximation::optimize_theta_multipoint(), and SurrBasedLocalMinimizer::relax_constraints().

```cpp
void eval_tag_prefix ( const String & eval_id_str ) [virtual]
```

set the hierarchical eval ID tag prefix

This prepend may need to become a virtual function if the tagging should propagate to other subModels or helper Iterators an Iterator may contain.

References Model::eval_tag_prefix(), Iterator::eval_tag_prefix(), Iterator::iteratedModel, and Iterator::iteratorRep.

Referenced by DataFitSurrModel::build_global(), NestedModel::derived_compute_response(), and Iterator::eval_tag_prefix().

```cpp
Iterator * get_iterator ( ProblemDescDB & problem_db ) [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 DB’s method_name. Supports all iterators and meta-iterators. These instantiations will NOT recurse on the Iterator(problem_db) constructor due to the use of BaseConstructor.

References ProblemDescDB::get_model(), ProblemDescDB::get_ushort(), Iterator::method_enum_to_string(), Iterator::method_name(), and Dakota::SUBMETHOD_COLLABORATIVE.

Referenced by Iterator::Iterator().

```cpp
Iterator * get_iterator ( ProblemDescDB & problem_db, 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. Alternate construction of meta-iterators is supported to enable use of meta-iterators as components. These instantiations will NOT recurse on the Iterator(problem_db, model) constructor due to the use of BaseConstructor.

References ProblemDescDB::get_ushort(), Iterator::method_enum_to_string(), Iterator::method_name(), Iterator::probDescDB, and Dakota::SUBMETHOD_COLLABORATIVE.

```cpp
Iterator * get_iterator ( const String & method_string, 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_string. Lightweight instantiations by name are supported by a subset of Iterators (primarily Minimizers).

References Dakota::strbegins(), and Dakota::strends().

13.52.4 Member Data Documentation

ProblemDescDB& probDescDB [protected]

class member reference to the problem description database

Iterator and Model cannot use a shallow copy of ProblemDescDB due to circular destruction dependency (reference counts can’t get to 0), since ProblemDescDB contains {iterator,model}List.

Referenced by MetaIterator::allocate_by_name(), MetaIterator::allocate_by_pointer(), Analyzer::Analyzer(), COLINOptimizer::COLINOptimizer(), CollabHybridMetaIterator::CollabHybridMetaIterator(), NonDExpansion::construct_expansion_sampler(), Minimizer::data_transform_model(), MetaIterator::deallocate(), EffGlobalMinimizer::EffGlobalMinimizer(), MetaIterator::estimate_by_name(), MetaIterator::estimate_by_pointer(), FSUDesignCompExp::FSUDesignCompExp(), Iterator::get_iterator(), ConcurrentMetaIterator::initialize(), ConcurrentMetaIterator::initialize_model(), Minimizer::initialize_scaling(), JEGAOptimizer::JEGAOptimizer(), NLSSOLLestSq::NLSSOLLestSq(), NomadOptimizer::NomadOptimizer(), NonD::NonD(), NonDAdaptiveSampling::NonDAdaptiveSampling(), NonDBayesCalibration::NonDBayesCalibration(), NonDCalibration::NonDCalibration(), NonDExpansion::NonDExpansion(), NonDGlobalInterval::NonDGlobalInterval(), NonDGlobalReliability::NonDGlobalReliability(), NonDGPlmpSampling::NonDGPlmpSampling(), NonDGPlmpSampling::NonDGPlmpSampling(), NonDGPMSABayesCalibration::NonDGPMSABayesCalibration(), NonDInterval::NonDInterval(), NonDLocalInterval::NonDLocalInterval(), NonDLocalReliability::NonDLocalReliability(), NonD::NonD(), NonDAdaptiveSampling::NonDAdaptiveSampling(), Iterator::free_communicators(), Iterator::free_communicators(), FSUDesignCompExp::FSUDesignCompExp(), EffGlobalMinimizer::EffGlobalMinimizer(), SerialMetaIterator::SerialMetaIterator(), SeqHybridMetaIterator::SeqHybridMetaIterator(), APPSOptimizer::set_apps_parameters(), COLINOptimizer::set_solver_parameters(), SNLLLeastSq::SNLLLeastSq(), SNLLOptimizer::SNLLOptimizer(), COLINOptimizer::solver_setup(), SurrBasedGlobalMinimizer::SurrBasedGlobalMinimizer(), SurrBasedLocalMinimizer::SurrBasedLocalMinimizer(), SurrBasedMinimizer::SurrBasedMinimizer(), ConcurrentMetaIterator::~ConcurrentMetaIterator().

int maxEvalConcurrency [protected]

maximum number of concurrent model evaluations

This is important for parallel configuration init/set/free and may be set within empty envelope instances. Therefore, it cannot be pushed down into Analyzer/Minimizer derived classes.

Referenced by DDACEDesignCompExp::DDACEDesignCompExp(), EffGlobalMinimizer::EffGlobalMinimizer(), EfficientSubspace::EfficientSubspaceMethod(), EfficientSubspace::EfficientSubspaceMethod(), NonD::NonD(), NonDAdaptiveSampling::NonDAdaptiveSampling(), ListenCommunicator::ListenCommunicator(), ParallelCommunicator::ParallelCommunicator(), ConcurrentCommunicator::ConcurrentCommunicator(), NonD::NonD(), NonDAdaptiveSampling::NonDAdaptiveSampling(), Iterator::free_communicators(), FSUDesignCompExp::FSUDesignCompExp(), EffGlobalMinimizer::EffGlobalMinimizer(), SerialMetaIterator::SerialMetaIterator(), SeqHybridMetaIterator::SeqHybridMetaIterator(), APPSOptimizer::set_apps_parameters(), COLINOptimizer::set_solver_parameters(), SNLLLeastSq::SNLLLeastSq(), SNLLOptimizer::SNLLOptimizer(), COLINOptimizer::solver_setup(), SurrBasedGlobalMinimizer::SurrBasedGlobalMinimizer(), SurrBasedLocalMinimizer::SurrBasedLocalMinimizer(), SurrBasedMinimizer::SurrBasedMinimizer(), ConcurrentMetaIterator::~ConcurrentMetaIterator().

NonD::NonD(), NonDAdaptiveSampling::NonDAdaptiveSampling(), NonD::NonD(), NonDAdaptiveSampling::NonDAdaptiveSampling(), NonDGPlmpSampling::NonDGPlmpSampling(), NonDGPMSABayesCalibration::NonDGPMSABayesCalibration(), NonD::NonD(), NonDAdaptiveSampling::NonDAdaptiveSampling(), Iterator::free_communicators(), FSUDesignCompExp::FSUDesignCompExp(), EffGlobalMinimizer::EffGlobalMinimizer(), SerialMetaIterator::SerialMetaIterator(), SeqHybridMetaIterator::SeqHybridMetaIterator(), APPSOptimizer::set_apps_parameters(), COLINOptimizer::set_solver_parameters(), SNLLLeastSq::SNLLLeastSq(), SNLLOptimizer::SNLLOptimizer(), COLINOptimizer::solver_setup(), SurrBasedGlobalMinimizer::SurrBasedGlobalMinimizer(), SurrBasedLocalMinimizer::SurrBasedLocalMinimizer(), SurrBasedMinimizer::SurrBasedMinimizer(), ConcurrentMetaIterator::~ConcurrentMetaIterator().

NonD::NonD(), NonDAdaptiveSampling::NonDAdaptiveSampling(), NonD::NonD(), NonDAdaptiveSampling::NonDAdaptiveSampling(), NonDGPlmpSampling::NonDGPlmpSampling(), NonDGPMSABayesCalibration::NonDGPMSABayesCalibration(), NonD::NonD(), NonDAdaptiveSampling::NonDAdaptiveSampling(), Iterator::free_communicators(), FSUDesignCompExp::FSUDesignCompExp(), EffGlobalMinimizer::EffGlobalMinimizer(), SerialMetaIterator::SerialMetaIterator(), SeqHybridMetaIterator::SeqHybridMetaIterator(), APPSOptimizer::set_apps_parameters(), COLINOptimizer::set_solver_parameters(), SNLLLeastSq::SNLLLeastSq(), SNLLOptimizer::SNLLOptimizer(), COLINOptimizer::solver_setup(), SurrBasedGlobalMinimizer::SurrBasedGlobalMinimizer(), SurrBasedLocalMinimizer::SurrBasedLocalMinimizer(), SurrBasedMinimizer::SurrBasedMinimizer(), ConcurrentMetaIterator::~ConcurrentMetaIterator().

NonD::NonD(), NonDAdaptiveSampling::NonDAdaptiveSampling(), NonD::NonD(), NonDAdaptiveSampling::NonDAdaptiveSampling(), NonDGPlmpSampling::NonDGPlmpSampling(), NonDGPMSABayesCalibration::NonDGPMSABayesCalibration(), NonD::NonD(), NonDAdaptiveSampling::NonDAdaptiveSampling(), Iterator::free_communicators(), FSUDesignCompExp::FSUDesignCompExp(), EffGlobalMinimizer::EffGlobalMinimizer(), SerialMetaIterator::SerialMetaIterator(), SeqHybridMetaIterator::SeqHybridMetaIterator(), APPSOptimizer::set_apps_parameters(), COLINOptimizer::set_solver_parameters(), SNLLLeastSq::SNLLLeastSq(), SNLLOptimizer::SNLLOptimizer(), COLINOptimizer::solver_setup(), SurrBasedGlobalMinimizer::SurrBasedGlobalMinimizer(), SurrBasedLocalMinimizer::SurrBasedLocalMinimizer(), SurrBasedMinimizer::SurrBasedMinimizer(), ConcurrentMetaIterator::~ConcurrentMetaIterator().
13.53. **ITERATORSCHEDULER CLASS REFERENCE**

Analyzer::initialize_run(), NonDExpansion::initialize_u_space_model(), JEGAOptimizer::JEGAOptimizer(), Iterator-::maximum_evaluation_concurrency(), NonDAdaptImpSampling::NonDAdaptImpSampling(), NonDBayesCalibration-::NonDBayesCalibration(), NonDHuber::NonDHuber(), NonDLocalInterval::NonDLocalInterval(), NonDGlobalInterval::NonDGlobalInterval(), NonDGlobalReliability::NonDGlobalReliability(), NonDLHSInterval::NonDLHSInterval(), NonDPolynomialChaos-::NonDPolynomialChaos(), NonDQuadrature::NonDQuadrature(), NonDSampling::NonDSampling(), NonDSparse-Grid::NonDSparseGrid(), Analyzer::num_samples(), ParamStudy::ParamStudy(), PSUADEDesignCompExp::PSUADEDesignCompExp(), RichExtrapVerification::RichExtrapVerification(), APPSOptimizer::set_apps_parameters(), Iterator::set_communicators(), COLINOptimizer::set_solver_parameters(), SNLLOptimizer::SNLLOptimizer(), and Iterator::update_from_model().

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

- DakotaIterator.hpp
- DakotaIterator.cpp

### 13.53 IteratorScheduler Class Reference

**Environment** corresponding to execution as a stand-alone application.

#### Public Member Functions

- **IteratorScheduler** (ParallelLibrary &parallel_lib, int num_servers=0, int procs_per_iterator=0, short scheduling=D-EFAULT_SCHEDULING)
  - constructor
- ~IteratorScheduler ()
  - destructor
- void init_iterator_parallelism (int max_iterator_concurrency, int min_procs_per_iterator=1, int max_procs_per_iterator=0, short default_config=PUSH_DOWN)
  - convenience function for initializing iterator communicators, setting parallel configuration attributes, and managing outputs and restart.
- void free_iterator_parallelism ()
  - convenience function for deallocating the concurrent iterator parallelism level
- int init_evaluation_concurrency (ProblemDescDB &problem_db, Iterator &the_iterator, Model &the_model, const ParallelLevel &pl)
  - convenience function for performing sufficient initialization to define the maximum evaluation concurrency
- int init_evaluation_concurrency (const String &method_string, Iterator &the_iterator, Model &the_model, const ParallelLevel &pl)
  - convenience function for performing sufficient initialization to define the maximum evaluation concurrency
- void schedule_iterators (Iterator &meta_iterator, Iterator &sub_iterator)
  - short convenience function for distributing control among master_dynamic_schedule_iterators(), serve_iterators(), and peer_static_schedule_iterators()
- void master_dynamic_schedule_iterators (Iterator &meta_iterator)
  - executed by the scheduler master to manage a dynamic schedule of iterator jobs among slave iterator servers
- void stop_iterator_servers ()
  - executed by the scheduler master to terminate slave iterator servers
- void serve_iterators (Iterator &meta_iterator, Iterator &sub_iterator)
  - executed on the slave iterator servers to perform iterator jobs assigned by the scheduler master
• void **peer**\_**static\_schedule\_iterators** (Iterator &meta\_iterator, Iterator &sub\_iterator)
  executed on iterator peers to manage a static schedule of iterator jobs
• void **iterator**\_**message**\_**lengths** (int params\_msg\_len, int results\_msg\_len)
  update params\_MsgLen and results\_MsgLen
• bool **lead**\_rank () const
  determines if current processor is rank 0 of the parent comm

**Static Public Member Functions**
• static void **init**\_**serial**\_**iterators** (ParallelLibrary &parallel\_lib)
  convenience function for serializing the concurrent iterator parallelism level
• static void **free**\_**iterator**\_parallelism (ParallelLibrary &parallel\_lib)
  convenience function for deallocating the concurrent iterator parallelism level
• static void **init**\_**iterator** (ProblemDescDB &problem\_db, Iterator &the\_iterator, Model &the\_model, const ParallelLevel &pl)
  convenience function for allocation of an iterator and (parallel) initialization of its comms
• static void **init**\_**iterator** (const String &method\_string, Iterator &the\_iterator, Model &the\_model, const ParallelLevel &pl)
  convenience function for lightweight allocation of an iterator and (parallel) initialization of its comms
• static void **run**\_**iterator** (Iterator &the\_iterator, const ParallelLevel &pl)
  Convenience function for invoking an iterator and managing parallelism. This version omits communicator reparation. Function must be public due to use by MINLPNode.
• static void **free**\_**iterator** (Iterator &the\_iterator, const ParallelLevel &pl)
  convenience function for deallocating comms after running an iterator

**Public Attributes**
• ParallelLibrary & parallel\_lib
  reference to the ParallelLibrary instance
• int numIteratorJobs
  number of iterator executions to schedule
• int numIteratorServers
  number of concurrent iterator partitions
• int procsPerIterator
  partition size request
• int iteratorCommRank
  processor rank in iteratorComm
• int iteratorCommSize
  number of processors in iteratorComm
• int iteratorServerId
  identifier for an iterator server
• bool messagePass
  flag for message passing at si level
• short iteratorScheduling
  \{DEFAULT,MASTER,PEER\}_SCHEDULING
13.53. **ITERATORSCHEDULER CLASS REFERENCE**

**Private Attributes**
- int paramsMsgLen
  - length of MPI buffer for parameter input instance(s)
- int resultsMsgLen
  - length of MPI buffer for results output instance(s)

13.53.1 **Detailed Description**

**Environment** corresponding to execution as a stand-alone application.

This environment corresponds to a stand-alone executable program, e.g., `main.cpp`. It sets up the ParallelLibrary and ProblemDescDB objects based on access to command line arguments.

13.53.2 **Constructor & Destructor Documentation**

`IteratorScheduler ( ParallelLibrary & parallel_lib, int num_servers = 0, int procs_per_iterator = 0, short scheduling = DEFAULT_SCHEDULING )`

constructor

Current constructor parameters are the input specification components, which are requests subject to override by `ParallelLibrary::init_iterator_communicators()`.

13.53.3 **Member Function Documentation**

**void init_serial_iterators ( ParallelLibrary & parallel_lib ) [static]**

convenience function for serializing the concurrent iterator parallelism level

Static version called for serialization.

References `ParallelLibrary::init_iterator_communicators()`, `ParallelLibrary::manage_outputs_restart()`, and `ParallelLibrary::world_size()`.

Referenced by `Environment::construct()`.

**void free_iterator_parallelism ( ParallelLibrary & parallel_lib ) [static]**

convenience function for deallocating the concurrent iterator parallelism level

Static version

References `ParallelLibrary::free_iterator_communicators()`.

Referenced by `MetaIterator::~MetaIterator()`.

**void init_iterator ( ProblemDescDB & problem_db, Iterator & the_iterator, Model & the_model, const ParallelLevel & pl ) [static]**

convenience function for allocation of an iterator and (parallel) initialization of its comms

This is a convenience function for encapsulating the allocation of communicators prior to running an iterator.

References `ParallelLevel::dedicated_master()`, `ProblemDescDB::get_iterator()`, `ProblemDescDB::get_ushort()`, `Model::init_comms_bcast_flag()`, `Iterator::init_communicators()`, `Iterator::is_null()`, `Iterator::iterated_model()`, `Iterator::is_null()`, `Model::stop_configurations()`, `ParallelLevel::server_communicator_rank()`, `ParallelLevel::server_communicator_size()`, `ParallelLevel::server_id()`, and `Model::stop_configurations()`.

Referenced by `MetaIterator::allocate_by_name()`, `MetaIterator::allocate_by_pointer()`, `ConcurrentMetaIterator::ConcurrentMetaIterator()`, and `Environment::construct()`.
void init_iterator ( const String & method_string, Iterator & the_iterator, Model & the_model, const ParallelLevel & pl ) [static]

convenience function for lightweight allocation of an iterator and (parallel) initialization of its comms

This is a convenience function for encapsulating the allocation of communicators prior to running an iterator.

References ParallelLevel::dedicated_master(), Model::init_comms_bcast_flag(), Iterator::init_communicators(), Iterator::is_null(), Iterator::iterated_model(), Iterator::maximum_evaluation_concurrency(), Iterator::method_string(), Model::serve_configurations(), ParallelLevel::server_communicator_rank(), ParallelLevel::server_communicator_size(), ParallelLevel::server_id(), and Model::stop_configurations().

void run_iterator ( Iterator & the_iterator, const ParallelLevel & pl ) [static]

Convenience function for invoking an iterator and managing parallelism. This version omits communicator repartitioning. Function must be public due to use by MINLPNode.

This is a convenience function for encapsulating the parallel features (run/serve) of running an iterator. This function omits allocation/deallocation of communicators to provide greater efficiency in approaches that involve multiple iterator executions but only require communicator allocation/deallocation to be performed once.

References Iterator::iterated_model(), Iterator::maximum_evaluation_concurrency(), Iterator::run(), Model::serve(), ParallelLevel::server_communicator_rank(), and Model::stop_servers().

Referenced by Environment::execute(), IteratorScheduler::peer_static_schedule_iterators(), SeqHybridMetaIterator::run_sequential_adaptive(), and IteratorScheduler::serve_iterators().

void free_iterator ( Iterator & the_iterator, const ParallelLevel & pl ) [static]

convenience function for deallocating comms after running an iterator

This is a convenience function for encapsulating the deallocation of communicators after running an iterator.

References ParallelLevel::dedicated_master(), Iterator::free_communicators(), and ParallelLevel::server_id().

Referenced by MetaIterator::deallocate(), Environment::destruct(), and ConcurrentMetaIterator::~ConcurrentMetaIterator().

void init_iterator_parallelism ( int max_iterator_concurrency, int min_procs_per_iterator = 1, int max_procs_per_iterator = 0, short default_config = PUSH_DOWN )

convenience function for initializing iterator communicators, setting parallel configuration attributes, and managing outputs and restart.

Called from derived class constructors once maxIteratorConcurrency is defined but prior to instantiating Iterators and Models.

References ParallelLevel::dedicated_master(), ParallelLibrary::init_iterator_communicators(), IteratorScheduler::iteratorCommRank, IteratorScheduler::iteratorCommSize, IteratorScheduler::iteratorScheduling, IteratorScheduler::iteratorServerId, ParallelLibrary::manage_outputs_restart(), ParallelLevel::message_pass(), IteratorScheduler::messagePass, ParallelLevel::num_servers(), IteratorScheduler::numIteratorServers, IteratorScheduler::parallelLib, IteratorScheduler::procsPerIterator, ParallelLevel::server_communicator_rank(), ParallelLevel::server_communicator_size(), ParallelLevel::server_id(), and ParallelLibrary::world_size().

Referenced by CollabHybridMetaIterator::CollabHybridMetaIterator(), ConcurrentMetaIterator::ConcurrentMetaIterator(), EmbedHybridMetaIterator::EmbedHybridMetaIterator(), and SeqHybridMetaIterator::SeqHybridMetaIterator().

int init_evaluation_concurrency ( ProblemDescDB & problem_db, Iterator & the_iterator, Model & the_model, const ParallelLevel & pl )

convenience function for performing sufficient initialization to define the maximum evaluation concurrency
This is a convenience function for computing the maximum evaluation concurrency prior to concurrent iterator partitioning.

References ParallelLibrary::bcast(), ProblemDescDB::get_iterator(), Iterator::is_null(), Iterator::maximum_evaluation_concurrency(), ProblemDescDB::parallel_library(), ParallelLevel::server_communicator_rank(), and ParallelLevel::server_communicator_size().

Referenced by ConcurrentMetaIterator::ConcurrentMetaIterator(), MetaIterator::estimate_by_name(), and MetaIterator::estimate_by_pointer().

```c
int init_evaluation_concurrency ( const String & method_string, Iterator & the_iterator, Model & the_model, const ParallelLevel & pl )
```

convenience function for computing the maximum evaluation concurrency prior to concurrent iterator partitioning.

References ParallelLibrary::bcast(), Iterator::is_null(), Iterator::maximum_evaluation_concurrency(), Model::parallel_library(), ParallelLevel::server_communicator_rank(), and ParallelLevel::server_communicator_size().

```c
void schedule_iterators ( Iterator & meta_iterator, Iterator & sub_iterator )
```

short convenience function for distributing control among master_dynamic_schedule_iterators(), serve_iterators(), and peer_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.

References IteratorScheduler::iteratorScheduling(), IteratorScheduler::iteratorServerId(), IteratorScheduler::leader_rank(), IteratorScheduler::master_dynamic_schedule_iterators(), IteratorScheduler::numIteratorServers(), IteratorScheduler::peer_static_schedule_iterators(), IteratorScheduler::serve_iterators(), and IteratorScheduler::stop_iterators().

Referenced by CollabHybridMetaIterator::core_run(), EmbedHybridMetaIterator::core_run(), ConcurrentMetaIterator::core_run(), and SeqHybridMetaIterator::run_sequential().

```c
void master_dynamic_schedule_iterators ( Iterator & meta_iterator )
```

executed by the scheduler master to manage a dynamic schedule of iterator jobs among slave iterator servers This function is adapted from ApplicationInterface::master_dynamic_schedule_evaluations().

References ParallelLibrary::free(), ParallelLibrary::irecv_si(), ParallelLibrary::isend_si(), IteratorScheduler::numIteratorJobs(), IteratorScheduler::numIteratorServers(), IteratorScheduler::parallelLib(), ParallelLevel::parallel_configuration(), ParallelLevel::parallel_time(), IteratorScheduler::resultsMsgLen(), Iterator::unpack_results_buffer(), ParallelLibrary::waitall(), and ParallelLibrary::waitsome().

Referenced by IteratorScheduler::schedule_iterators().

```c
void serve_iterators ( Iterator & meta_iterator, Iterator & sub_iterator )
```

executed on the slave iterator servers to perform iterator jobs assigned by the scheduler master This function is similar in structure to ApplicationInterface::serve_evaluations_synch().

References ParallelLibrary::bcast_i(), IteratorScheduler::iteratorCommRank(), IteratorScheduler::iteratorCommSize(), Iterator::pack_results_buffer(), ParallelLibrary::parallel_configuration(), ParallelLibrary::parallel_time(), IteratorScheduler::parallelLib(), IteratorScheduler::paramsMsgLen(), ParallelLibrary::recv_si(), IteratorScheduler::results_MsgLen(), IteratorScheduler::run_iterator(), ParallelLibrary::send_si(), ParallelConfiguration::si_parallel_level(), Iterator::unpack_parameters_buffer(), and Iterator::update_local_results().

Referenced by IteratorScheduler::schedule_iterators().
The documentation for this class was generated from the following files:
- IteratorScheduler.hpp
- IteratorScheduler.cpp

### 13.54 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
```

**Classes**

- class **Driver**
  
  A subclass of the JEGA front end driver that exposes the individual protected methods to execute the algorithm.

- class **Evaluator**
  
  An evaluator specialization that knows how to interact with Dakota.

- class **EvaluatorCreator**
  
  A specialization of the JEGA::FrontEnd::EvaluatorCreator that creates a new instance of a Evaluator.

**Public Member Functions**

- virtual void **find_optimum** ()
  
  Performs the iterations to determine the optimal set of solutions.

- virtual bool **accepts_multiple_points** () const
  
  Overridden to return true since JEGA algorithms can accept multiple initial points.

- virtual bool **returns_multiple_points** () const
  
  Overridden to return true since JEGA algorithms can return multiple final points.

- virtual void **initial_points** (const VariablesArray &pts)
  
  Overridden to assign the _initPts member variable to the passed in collection of Dakota::Variables.

- virtual const VariablesArray & **initial_points** () const
  
  Overridden to return the collection of initial points for the JEGA algorithm created and run by this JEGAOptimizer.

- **JEGAOptimizer** (ProblemDescDB &problem_db, Model &model)

  Constructs a JEGAOptimizer class object.

- **~JEGAOptimizer** ()

  Destroys a JEGAOptimizer.
13.54. JEGAOPTIMIZER CLASS REFERENCE

Protected Member Functions

- void LoadDakotaResponses (const JEGA::Utilities::Design &from, Variables &vars, Response &resp) const
  Loads the JEGA-style Design class into equivalent Dakota-style Variables and Response objects.
- void ReCreateTheParameterDatabase ()
  Destroys the current parameter database and creates a new empty one.
- void LoadTheParameterDatabase ()
  Reads information out of the known Dakota::ProblemDescDB and puts it into the current parameter database.
- void LoadAlgorithmConfig (JEGA::FrontEnd::AlgorithmConfig &aConfig)
  Completely initializes the supplied algorithm configuration.
- void LoadProblemConfig (JEGA::FrontEnd::ProblemConfig &pConfig)
  Completely initializes the supplied problem configuration.
- void LoadTheDesignVariables (JEGA::FrontEnd::ProblemConfig &pConfig)
  Adds DesignVariableInfo objects into the problem configuration object.
- void LoadTheObjectiveFunctions (JEGA::FrontEnd::ProblemConfig &pConfig)
  Adds ObjectiveFunctionInfo objects into the problem configuration object.
- void LoadTheConstraints (JEGA::FrontEnd::ProblemConfig &pConfig)
  Adds ConstraintInfo objects into the problem configuration object.
- void GetBestSolutions (const JEGA::Utilities::DesignOFSortSet &from, std::multimap<RealRealPair, JEGA::Utilities::Design *> &designSortMap)
  Returns up to numBest designs sorted by DAKOTA's fitness (L2 constraint violation, then utopia or objective),
  taking into account the algorithm type. The front of the returned map can be viewed as a single "best".
- void GetBestMOSolutions (const JEGA::Utilities::DesignOFSortSet &from, std::multimap<RealRealPair, JEGA::Utilities::Design *> &designSortMap)
  Retrieve the best Designs from a set of solutions assuming that they are generated by a multi objective algorithm.
- void GetBestSOSolutions (const JEGA::Utilities::DesignOFSortSet &from, std::multimap<RealRealPair, JEGA::Utilities::Design *> &designSortMap)
  Retrieve the best Designs from a set of solutions assuming that they are generated by a single objective algorithm.
- JEGA::DoubleMatrix ToDoubleMatrix (const VariablesArray &variables) const
  Converts the items in a VariablesArray into a DoubleMatrix whereby the items in the matrix are the design variables.

Private Attributes

- EvaluatorCreator * _theEvalCreator
  A pointer to an EvaluatorCreator used to create the evaluator used by JEGA in Dakota (a JEGAevaluator).
- JEGA::Utilities::ParameterDatabase * _theParamDB
  A pointer to the ParameterDatabase from which all parameters are retrieved by the created algorithms.
- VariablesArray _initPts
  An array of initial points to use as an initial population.

Additional Inherited Members

13.54.1 Detailed Description

A version of Dakota::Optimizer for instantiation of John Eddy’s Genetic Algorithms (JEGA).
This class encapsulates the necessary functionality for creating and properly initializing the JEGA algorithms
(MOGA and SOGA).
13.54.2 Constructor & Destructor Documentation

JEGAOptimizer ( ProblemDescDB & problem_db, 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.

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>problem_db</td>
<td>The Dakota::ProblemDescDB with information on how the algorithm controls should be set.</td>
</tr>
<tr>
<td>model</td>
<td>The Dakota::Model that will be used by this optimizer for problem information, etc.</td>
</tr>
</tbody>
</table>

References JEGAOptimizer::theEvalCreator, ProblemDescDB::get_int(), ProblemDescDB::get_short(), Iterator::iteratedModel, JEGAOptimizer::LoadTheParameterDatabase(), Iterator::maxEvalConcurrency, Iterator::methodName, Iterator::numFinalSolutions, and Iterator::probDescDB.

13.54.3 Member Function Documentation

void LoadDakotaResponses ( const JEGA::Utilities::Design & from, Dakota::Variables & vars, Dakota::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.

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>from</td>
<td>The JEGA Design class object from which to extract the variable and response information for Dakota.</td>
</tr>
<tr>
<td>vars</td>
<td>The Dakota::Variables object into which to load the design variable values of from.</td>
</tr>
<tr>
<td>resp</td>
<td>The Dakota::Response object into which to load the objective function and constraint values of from.</td>
</tr>
</tbody>
</table>

References Variables::continuous_variables(), Variables::discrete_int_variables(), Variables::discrete_real_variables(), and Response::function_values().

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.

Referenced by JEGAOptimizer::JEGAOptimizer().

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.

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aConfig</td>
<td>The algorithm configuration object to load.</td>
</tr>
</tbody>
</table>

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]**

This retrieves design variable information from the ParameterDatabase and creates DesignVariableInfo's from it.

Parameters

- pConfig - The problem configuration object to load.

References Model::continuous_lower_bounds().

**void LoadTheObjectiveFunctions ( JEGA::FrontEnd::ProblemConfig & pConfig ) [protected]**

This retrieves objective function information from the ParameterDatabase and creates ObjectiveFunctionInfo's from it.

Parameters

- pConfig - The problem configuration object to load.

References Dakota::asstring(), Dakota::copy_row_vector(), and Model::nonlinear_ineq_constraint_lower_bounds().

**void LoadTheConstraints ( JEGA::FrontEnd::ProblemConfig & pConfig ) [protected]**

This retrieves constraint function information from the ParameterDatabase and creates ConstraintInfo's from it.

Parameters

- pConfig - The problem configuration object to load.

References Dakota::asstring(), Dakota::copy_row_vector(), and Model::nonlinear_ineq_constraint_lower_bounds().

**void GetBestSolutions ( const JEGA::Utilities::DesignOFSortSet & from, std::multimap<RealRealPair, JEGA::Utilities::Design *> & designSortMap ) [protected]**

Returns up to _numBest designs sorted by DAKOTA's fitness (L2 constraint violation, then utopia or objective), taking into account the algorithm type. The front of the returned map can be viewed as a single "best".

Parameters

- from - The full set of designs returned by the solver.
- designSortMap - Map of best solutions with key pair<constraintViolation, fitness>

eventually this functionality must be moved into a separate post-processing application for MO datasets.

**void GetBestMOSolutions ( const JEGA::Utilities::DesignOFSortSet & from, std::multimap<RealRealPair, JEGA::Utilities::Design *> & designSortMap ) [protected]**

Retreive the best Designs from a set of solutions assuming that they are generated by a multi objective algorithm. eventually this functionality must be moved into a separate post-processing application for MO datasets.
void GetBestSOSolutions ( const JEGA::Utilities::DesignOFSortSet & from, std::multimap<RealRealPair, JEGA::Utilities::Design *> & designSortMap ) [protected]
Retreive the best Designs from a set of solutions assuming that they are generated by a single objective algorithm. eventually this functionality must be moved into a separate post-processing application for MO datasets.

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.

void find_optimum ( ) [virtual]
Performs the iterations to determine the optimal set of solutions.
Override of pure virtual method in Optimizer base class.
The extraction of parameter values actually occurs in this method when the JEGA::FrontEnd::Driver::ExecuteAlgorithm is called. Also the loading of the problem and algorithm configurations occurs in this method. That way, if it is called more than once and the algorithm or problem has changed, it will be accounted for.
Implements Optimizer.
References JEGAOptimizer::Driver::DestroyAlgorithm(), JEGAOptimizer::Driver::ExtractAllData(), and JEGAOptimizer::Driver::PerformIterations().

bool accepts_multiple_points ( ) const [virtual]
Overridden to return true since JEGA algorithms can accept multiple initial points.
Returns
ture, always.
Reimplemented from Iterator.

bool returns_multiple_points ( ) const [virtual]
Overridden to return true since JEGA algorithms can return multiple final points.
Returns
ture, always.
Reimplemented from Iterator.

void initial_points ( const VariablesArray & pts ) [virtual]
Overridden to assign the _initPnts member variable to the passed in collection of Dakota::Variables.
Parameters

\begin{verbatim}
  pts | The array of initial points for the JEGA algorithm created and run by this JEGAOptimizer.
\end{verbatim}

Reimplemented from \texttt{Iterator}.

\begin{verbatim}
  const VariablesArray & initial_points () const [virtual]
\end{verbatim}

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 \texttt{Iterator}.

### 13.54.4 Member Data Documentation

\begin{verbatim}
  VariablesArray _initPts [private]
\end{verbatim}

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:

- \texttt{JEGAOptimizer.hpp}
- \texttt{JEGAOptimizer.cpp}

### 13.55 LeastSq Class Reference

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

Inheritance diagram for \texttt{LeastSq}:

```
  Iterator
    Minimizer
      LeastSq
        NL2SOLLeastSq
        NLSSOLLeastSq
        SNLLLeastSq
```

#### Protected Member Functions

- \texttt{LeastSq ()}
  \textit{default constructor}
- \texttt{LeastSq (ProblemDescDB &problem_db, Model &model)
13.55. LEASTSQ CLASS REFERENCE

standard constructor

- LeastSq (unsigned short method_name, Model &model)
  alternate "on the fly" constructor

- ~LeastSq ()
  destructor

- void initialize_run ()
- void core_run ()
  core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post

- void post_run (std::ostream &s)
- void finalize_run ()
  utility function to perform common operations following post_run(); deallocation and resetting of instance pointers

- void print_results (std::ostream &s)
- virtual void minimize_residuals ()=0
  Used within the least squares branch for minimizing the sum of squares residuals. Redefines the run virtual function
  for the least squares branch.

- void get_confidence_intervals ()
  Calculate confidence intervals on estimated parameters.

Protected Attributes

- int numLeastSqTerms
  number of least squares terms

- LeastSq * prevLSqInstance
  pointer containing previous value of leastSqInstance

- bool weightFlag
  flag indicating whether weighted least squares is active

- RealVector confBoundsLower
  lower bounds for confidence intervals on calibration parameters

- RealVector confBoundsUpper
  upper bounds for confidence intervals on calibration parameters

Static Protected Attributes

- static LeastSq * leastSqInstance
  pointer to LeastSq instance used in static member functions

Private Member Functions

- void weight_model ()
  Wrap iteratedModel in a RecastModel that weights the residuals.

Static Private Member Functions

- static void primary_resp_weighter (const Variables &unweighted_vars, const Variables &weighted_vars,
  const Response &unweighted_response, Response &weighted_response)
  Recast callback function to weight least squares residuals, gradients, and Hessians.
Additional Inherited Members

13.55.1 Detailed Description

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

The LeastSq class provides common data and functionality for least squares solvers (including NL2OL, NLSSOLLeastSq, and SNLLLeastSq).

13.55.2 Constructor & Destructor Documentation

LeastSq ( ProblemDescDB & problem_db, Model & model ) [protected]

standard constructor

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

References Dakota::abort_handler(), Iterator::bestVariablesArray, Variables::copy(), Model::current_variables(), Minimizer::data_transform_model(), Iterator::iteratedModel, Iterator::methodName, Minimizer::minimizerRecasts, Minimizer::numIterPrimaryFns, LeastSq::numLeastSqTerms, Minimizer::numRowsExpData, Minimizer::numUserPrimaryFns, Minimizer::obsDataFlag, Minimizer::optimizationFlag, Minimizer::scale_model(), Minimizer::scaleFlag, LeastSq::weight_model(), and LeastSq::weightFlag.

13.55.3 Member Function Documentation

void initialize_run ( ) [protected], [virtual]

This function should be invoked (or reimplemented) by any derived implementations of initialize_run() (which would otherwise hide it).

Reimplemented from Iterator.
Reimplemented in SNLLLeastSq.

References Minimizer::initialize_run(), Iterator::iteratedModel, LeastSq::leastSqInstance, Minimizer::obsDataFlag, LeastSq::prevLSqInstance, Minimizer::scaleFlag, and Model::update_from_subordinate_model().

Referenced by SNLLLeastSq::initialize_run().

void core_run ( ) [inline], [protected], [virtual]

core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post

Virtual run function for the iterator class hierarchy. All derived classes need to redefine it.

Reimplemented from Iterator.
References LeastSq::minimize_residuals().

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

Implements portions of post_run specific to LeastSq for scaling back to native variables and functions. This function should be invoked (or reimplemented) by any derived implementations of post_run() (which would otherwise hide it).

Reimplemented from Iterator.
Reimplemented in SNLLLeastSq.

References Dakota::abort_handler(), Response::active_set_request_vector(), Iterator::bestResponseArray, Iterator::bestVariablesArray, Variables::continuous_variables(), Response::copy(), Minimizer::cvScaleMultipliers, Minimizer::cvScaleOffsets, Minimizer::cvScaleTypes, Response::function_value(), Response::function_values(), Iterator::iteratedModel, Minimizer::modify_s2n(), Minimizer::need_resp_trans_byvars(), LeastSq::numLeastSqTerms, Minimizer::numNonlinearConstraints, Minimizer::post_run(), Model::primary_response_fn_weights(), Minimizer::primary-
RespScaleFlag, Minimizer::response_modify_s2n(), Minimizer::secondaryRespScaleFlag, Model::subordinate_model(), Response::update_partial(), Minimizer::varsScaleFlag, and LeastSq::weightFlag.

```cpp
void finalize_run( ) [inline], [protected], [virtual]
```

Utility function to perform common operations following `post_run()`. Deallocation and resetting of instance pointers. Optional: perform finalization phases of run sequence, like deallocating memory and resetting instance pointers. Commonly used in sub-iterator executions. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s `finalize_run()`, typically after performing its own implementation steps.

- Reimplemented from `Iterator`.
- Reimplemented in `SNLLLeastSq`.
- References `Minimizer::finalize_run()`, `LeastSq::leastSqInstance`, and `LeastSq::prevLSqInstance`.
- Referenced by `SNLLLeastSq::finalize_run()`.

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

Redefines default iterator results printing to include nonlinear least squares results (residual terms and constraints).

- Reimplemented from `Iterator`.
- References `Iterator::activeSet`, `Minimizer::archive_allocate_best()`, `Minimizer::archive_best()`, `Iterator::bestResponseArray`, `Iterator::bestVariablesArray`, `LeastSq::confBoundsLower`, `LeastSq::confBoundsUpper`, `Model::continuous_variable_labels()`, `Dakota::data_pairs`, `Model::interface_id()`, `Iterator::iteratedModel`, `Dakota::lookup_by_val()`, `Minimizer::numContinuousVars`, `Minimizer::numFunctions`, `LeastSq::numLeastSqTerms`, `Model::primary_response_fn_weights()`, `ActiveSet::request_values()`, `Model::subordinate_model()`, and `Dakota::write_precision`.

```cpp
void get_confidence_intervals( ) [protected]
```

Calculate confidence intervals on estimated parameters.

- Calculate individual confidence intervals for each parameter. These bounds are based on a linear approximation of the nonlinear model.
- References `Iterator::activeSet`, `Iterator::bestResponseArray`, `Iterator::bestVariablesArray`, `Model::compute_response()`, `LeastSq::confBoundsLower`, `LeastSq::confBoundsUpper`, `Model::continuous_variables()`, `Model::current_response()`, `Response::function_gradients()`, `Iterator::iteratedModel`, `Minimizer::numContinuousVars`, `LeastSq::numLeastSqTerms`, `ActiveSet::request_values()`, `Minimizer::scaleFlag`, and `Minimizer::vendorNumericalGradFlag`.
- Referenced by `NL2SOLLeastSq::minimize_residuals()`, `NLSSOLLeastSq::minimize_residuals()`, and `SNLLLeastSq::post_run()`.

```cpp
void weight_model( ) [private]
```

Wrap `iteratedModel` in a `RecastModel` that weights the residuals.

- Setup Recast for weighting model the weighting transformation doesn’t resize, so use `numUserPrimaryFns`. No vars, active set or secondary mapping. All indices are one-to-one mapped (no change in counts).
- References `Model::assign_rep()`, `Iterator::iteratedModel`, `Minimizer::numContinuousVars`, `LeastSq::numLeastSqTerms`, `Minimizer::numNonlinearConstraints`, `Minimizer::numNonlinearIneqConstraints`, `Iterator::outputLevel`, `LeastSq::primary_resp_weighter()`, `Model::primary_response_fn_sense()`, `Model::primary_response_fn_weights()`, `Minimizer::secondary_resp_copier()`, and `Model::subordinate_model()`.
- Referenced by `LeastSq::LeastSq()`.
void primary_resp_weighter ( const Variables & unweighted_vars, const Variables & weighted_vars, const Response & unweighted_response, Response & weighted_response ) [static],[private]

Recast callback function to weight least squares residuals, gradients, and Hessians.
Apply weights to least squares residuals

References Dakota::NPOS, Response::active_set_derivative_vector(), Response::active_set_request_vector(), Variables::acv(), Variables::all_continuous_variable_ids(), Variables::continuous_variable_ids(), Variables::cv(), Dakota::find_index(), Response::function_gradients(), Response::function_gradients_view(), Response::function_hessian(), Response::function_hessian_view(), Response::function_values(), Response::function_values_view(), Variables::icv(), Variables::inactive_continuous_variable_ids(), Iterator::iteratedModel, LeastSq::leastSqInstance, LeastSq::numLeastSqTerms, Iterator::outputLevel, Model::primary_response_fn_weights(), and Model::subordinate_model().

Referenced by LeastSq::weight_model().
The documentation for this class was generated from the following files:

- DakotaLeastSq.hpp
- DakotaLeastSq.cpp

13.56 LibraryEnvironment Class Reference

Environment corresponding to execution as an embedded library.
Inheritance diagram for LibraryEnvironment:

```
  Environment
     ^
     | LibraryEnvironment
```

Public Member Functions

- LibraryEnvironment ()
  default constructor
- LibraryEnvironment (ProgramOptions prog_opts, bool check_bcast_construct=true, DbCallbackFunctionPtr callback=NULL, void *callback_data=NULL)
  Primary constructor: program options typically specifies an input file or input string. Optionally specify a callback function to be invoked after parsing. Set check_bcast_construct if performing late updates and later calling done_modifying_db().
- LibraryEnvironment (MPI_Comm dakota_mpi_comm, ProgramOptions prog_opts=ProgramOptions(), bool check_bcast_construct=true, DbCallbackFunctionPtr callback=NULL, void *callback_data=NULL)
  Alternate constructor accepting communicator, same options as primary.
- ~LibraryEnvironment ()
  destructor
- void insert_nodes (Dakota::DataMethod &dme, Dakota::DataModel &dmo, Dakota::DataVariables &dv, Dakota::DataInterface &di, Dakota::DataResponses &dr)
  Insert DB nodes for a {Method,Model,Variables,Interface,Responses} set.
- void done_modifying_db ()
  Check database contents, broadcast, and construct iterators.
• **bool plugin_interface** (const String &model_type, const String &interf_type, const String &an_driver, Interface *plugin_iface)
  
  Plug-in the passed interface into any interface matching the specified (possibly empty) model, interface, and driver strings; returns true if a plugin was performed.

• **InterfaceList filtered_interface_list** (const String &interf_type, const String &an_driver)
  
  Filter the available Interface instances based on matching interface type and analysis drivers (empty String matches any)

• **ModelList filtered_model_list** (const String &model_type, const String &interf_type, const String &an_driver)
  
  Filter the available Model instances based on matching model type, interface type, and analysis drivers (empty String matches any)

### Additional Inherited Members

#### 13.56.1 Detailed Description

**Environment** corresponding to execution as an embedded library.

This environment corresponds to use of Dakota as a library within another application, e.g., within library_-mode.cpp. It sets up the ParallelLibrary and ProblemDescDB objects without access to command line arguments.

#### 13.56.2 Constructor & Destructor Documentation

**LibraryEnvironment** ( ProgramOptions *prog_opts*, bool *check_bcast_construct* = *true*, DbCallbackFunctionPtr *callback* = *NULL*, void *callback_data* = *NULL*)

Primary constructor: program options typically specifies an input file or input string. Optionally specify a callback function to be invoked after parsing. Set check_bcast_construct if performing late updates and later calling done_modifying_db().

Construct library environment, optionally performing check/bcast of database and iterator construction.

References Environment::construct(), OutputManager::output_startup_message(), Environment::outputManager, and Environment::parse().

**LibraryEnvironment** ( MPI_Comm *dakota_mpi_comm*, ProgramOptions *prog_opts* = ProgramOptions(), bool *check_bcast_construct* = *true*, DbCallbackFunctionPtr *callback* = *NULL*, void *callback_data* = *NULL*)

Alternate constructor accepting communicator, same options as primary.

Construct library environment on passed MPI Comm, optionally performing check/bcast of database and iterator construction. MPI Comm is first argument so client doesn’t have to pass all args.

References Environment::construct(), OutputManager::output_startup_message(), Environment::outputManager, and Environment::parse().

#### 13.56.3 Member Function Documentation

**InterfaceList filtered_interface_list** ( const String &interf_type, const String &an_driver )

Filter the available Interface instances based on matching interface type and analysis drivers (empty String matches any)

This convenience function helps clients locate and plugin to the right Interface instance for simple cases. Pass an empty string to match any instead of a specific instance.
References Interface::analysis_drivers(), Dakota::contains(), Interface::interface_type(), ProblemDescDB::model_list(), and Environment::probDescDB.

ModelList filtered_model_list ( const String &model_type, const String &interf_type, const String &an_driver )

filter the available Model instances based on matching model type, interface type, and analysis drivers (empty String matches any)

This convenience function helps clients locate and plugin to the right Interface instance for cases where the parallel configuration is needed in constructing a parallel plugin. Pass an empty string to match any instead of a specific instance

References Interface::analysis_drivers(), Dakota::contains(), Interface::interface_type(), ProblemDescDB::model_list(), and Environment::probDescDB.

Referenced by parallel_interface_plugin(), LibraryEnvironment::plugin_interface(), run_dakota(), and run_dakota_mixed().

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

- LibraryEnvironment.hpp
- LibraryEnvironment.cpp

13.57 MatlabInterface Class Reference

Inheritance diagram for MatlabInterface:

```
Interface
   
ApplicationInterface
   
DirectApplicInterface
   
MatlabInterface
```

Public Member Functions

- MatlabInterface (const ProblemDescDB &problem_db)
  Constructor: start Matlab engine.
- ~MatlabInterface ()
  Destructor: close Matlab engine.

Protected Member Functions

- virtual int derived_map_ac (const String &ac_name)
  execute an analysis code portion of a direct evaluation invocation
- int matlab_engine_run (const Dakota::String &ac_name)
  Helper function supporting derived_map_ac. Sends data to Matlab, executes analysis, collects return data.
• int `matlab_field_prep` (mxArray *dakota_matlab, const char *field_name)
  
  *check that the dakota_matlab structure has the specified field_name and add if necessary; free structure memory in preparation for new alloc*

**Protected Attributes**

• engine * `matlabEngine`
  
  *pointer to the MATLAB engine used for direct evaluations*

### 13.57.1 Detailed Description

Specialization of `DirectApplicInterface` to link to Matlab analysis drivers. Includes convenience functions to map data to/from Matlab.

### 13.57.2 Member Function Documentation

**int derived_map_ac ( const String & `ac_name` ) [protected], [virtual]**

execute an analysis code portion of a direct evaluation invocation

Matlab specialization of derived analysis components.

Reimplemented from `DirectApplicInterface`.

References ApplicationInterface::analysisServerId, and MatlabInterface::matlab_engine_run().

**int matlab_engine_run ( const Dakota::String & `ac_name` ) [protected]**

Helper function supporting derived_map_ac. Sends data to Matlab, executes analysis, collects return data.

Direct interface to Matlab through Mathworks external API. m-file executed is specified through analysis components, extra strings through analysis components. (Original BMA 11/28/2005)

Special thanks to Lee Peterson for substantial enhancements 12/15/2007: Added output buffer for the M-ATLAB command response and error messages Made the Dakota variable persistent in the MATLAB engine workspace Added robustness to the user deleting required Dakota fields

References Dakota::abort_handler(), DirectApplicInterface::analysisComponents, DirectApplicInterface::analysisDriverIndex, Interface::currEvalId, DirectApplicInterface::directFnASV, DirectApplicInterface::directFnDVV, Dakota::FIELD_NAMES, DirectApplicInterface::fnGrads, DirectApplicInterface::fnHessians, Interface::fnLabels, DirectApplicInterface::fnVals, DirectApplicInterface::gradFlag, DirectApplicInterface::hessFlag, MatlabInterface::matlab_field_prep(), MatlabInterface::matlabEngine, DirectApplicInterface::numACV, DirectApplicInterface::numADIV, DirectApplicInterface::numADRV, Dakota::NUMBER_OF_FIELDS, DirectApplicInterface::numDerivVars, DirectApplicInterface::numFns, DirectApplicInterface::numVars, Interface::outputLevel, DirectApplicInterface::x, DirectApplicInterface::xCLabels, DirectApplicInterface::xDI, DirectApplicInterface::xDILabels, DirectApplicInterface::xDRLabels.

Referenced by MatlabInterface::derived_map_ac().

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

• MatlabInterface.hpp
• MatlabInterface.cpp
13.58 MetaIterator Class Reference

Base class for meta-iterators.

Inheritance diagram for MetaIterator:

```
    Iterator
      |
      v
MetaIterator

  CollabHybridMetaIterator  ConcurrentMetaIterator  EmbedHybridMetaIterator  SeqHybridMetaIterator
```

Protected Member Functions

- MetaIterator (ProblemDescDB &problem_db)
  - standard constructor
- MetaIterator (ProblemDescDB &problem_db, Model &model)
  - alternate constructor
- ~MetaIterator ()
  - destructor
- void post_run (std::ostream &s)
  - post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way
- int get_min_procs_per_iterator (ProblemDescDB &problem_db)
  - compute a minimum iterator partition size based on lower level overrides
- int get_max_procs_per_iterator (ProblemDescDB &problem_db, int max_eval_concurrency)
  - compute a maximum iterator partition size based on lower level concurrency
- void allocate_by_pointer (const String &method_ptr, Iterator &the_iterator, Model &the_model)
  - initialize the_iterator and the_model based on method_ptr
- void allocate_by_name (const String &method_string, const String &model_ptr, Iterator &the_iterator, Model &the_model)
  - initialize the_iterator based on method_string
- std::pair<int, int> estimate_by_pointer (const String &method_ptr, Iterator &the_iterator, Model &the_model)
  - estimate minimum and maximum processors per iterator needed for init_iterator_parallelism(); instantiates the_iterator and the_model as needed, but on minimal processor ranks (is later augmented by allocate_by_pointer())
- std::pair<int, int> estimate_by_name (const String &method_string, const String &model_ptr, Iterator &the_iterator, Model &the_model)
  - estimate minimum and maximum processors per iterator needed for init_iterator_parallelism(); instantiates the_iterator and the_model as needed, but on minimal processor ranks (is later augmented by allocate_by_name())
- void deallocate (Iterator &the_iterator, Model &the_model)
  - free communicators for the_iterator and the_model
Protected Attributes

- IteratorScheduler iterSched
  
  scheduler for concurrent execution of Iterators

- int maxIteratorConcurrency
  
  maximum number of concurrent sub-iterator executions

Additional Inherited Members

13.58.1 Detailed Description

Base class for meta-iterators.

This base class shares code for concurrent and hybrid meta-iterators, where the former supports multi-start and Pareto set iteration and the latter supports sequential, embedded, and collaborative hybrids.

13.58.2 Member Function Documentation

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

post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way

Post-run phase, which a derived iterator may optionally reimplement; when not present, post-run is likely integrated into run. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s post_run(), typically after performing its own implementation steps.

Reimplemented from Iterator.

References MetaIterator::iterSched, IteratorScheduler::lead_rank(), and Iterator::print_results().

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

- MetaIterator.hpp
- MetaIterator.cpp

13.59 Minimizer Class Reference

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

Inheritance diagram for Minimizer:
CHAPTER 13. CLASS DOCUMENTATION

**Public Member Functions**

- **void constraint_tolerance** (Real constr_tol)
  
  *set the method constraint tolerance (constraintTol)*

- **Real constraint_tolerance** () const
  
  *return the method constraint tolerance (constraintTol)*

**Protected Member Functions**

- **Minimizer ()**
  
  *default constructor*

- **Minimizer (ProblemDescDB &problem_db, Model &model)**
  
  *standard constructor*

- **Minimizer (unsigned short method_name, Model &model)**
  
  *alternate constructor for "on the fly" instantiations*

- **Minimizer (unsigned short method_name, 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 update_from_model** (const Model &model)
  
  *set inherited data attributes based on extractions from incoming model*

- **void initialize_run** ()
  
  *utility function to perform common operations prior to pre_run(); typically memory initialization; setting of instance pointers*
• **void post_run**(std::ostream &s)
  
  post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way

• **void finalize_run**()
  
  utility function to perform common operations following post_run(); deallocation and resetting of instance pointers

• **const Model & algorithm_space_model**() const

• **virtual void initialize_iterator**(int index)
  
  initialize the iterator about to be executed within a parallel iterator scheduling function (serve_iterators() or static_schedule_iterators())

• **virtual void pack_parameters_buffer**(MPITpackBuffer &send_buffer, int job_index)
  
  pack a send_buffer for assigning an iterator job to a server

• **virtual void unpack_parameters_buffer**(MPIUpackBuffer &recv_buffer)
  
  unpack a recv_buffer for accepting an iterator job from the scheduler

• **virtual void pack_results_buffer**(MPITpackBuffer &send_buffer, int job_index)
  
  pack a send_buffer for returning iterator results from a server

• **virtual void unpack_results_buffer**(MPIUpackBuffer &recv_buffer, int job_index)
  
  unpack a recv_buffer for accepting iterator results from a server

• **virtual void update_local_results**(int job_index)
  
  update local PRP results arrays with current iteration results

• **bool data_transform_model**(bool weight_flag=false)
  
  Wrap iteratedModel in a RecastModel that subtracts provided observed data from the primary response functions (variables and secondary responses are unchanged)

• **void scale_model**()
  
  Wrap iteratedModel in a RecastModel that performs variable and/or response scaling.

• **bool need_resp_trans_byvars**(const ShortArray &asv, int start_index, int num_resp)
  
  determine if response transformation is needed due to variable transformations

• **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_s2n**(const Variables &native_vars, const Response &scaled_response, Response &native_response, int start_offset, int num_responses) const
  
  map responses from scaled to native space

• **Real objective**(const RealVector &fn_vals, const BoolDeque &max_sense, const RealVector &primary_wts) const
  
  compute a composite objective value from one or more primary functions

• **Real objective**(const RealVector &fn_vals, size_t num_fns, const BoolDeque &max_sense, const RealVector &primary_wts) const
  
  compute a composite objective with specified number of source primary functions, instead of userPrimaryFns

• **void objective_gradient**(const RealVector &fn_vals, const RealMatrix &fn_grads, const BoolDeque &max_sense, const RealVector &primary_wts, RealVector &obj_grad) const
  
  compute the gradient of the composite objective function

• **void objective_gradient**(const RealVector &fn_vals, size_t num_fns, const RealMatrix &fn_grads, const BoolDeque &max_sense, const RealVector &primary_wts, RealVector &obj_grad) const
  
  compute the gradient of the composite objective function
• void objective_hessian (const RealVector &fn_vals, const RealMatrix &fn_grads, const RealSymMatrixArray &fn_hessians, const BoolDeque &max_sense, const RealVector &primary_wts, RealSymMatrix &obj_hess) const
  
  compute the Hessian of the composite objective function

• void objective_hessian (const RealVector &fn_vals, size_t num_fns, const RealMatrix &fn_grads, const RealSymMatrixArray &fn_hessians, const BoolDeque &max_sense, const RealVector &primary_wts, RealSymMatrix &obj_hess) const

  compute the Hessian of the composite objective function

• void archive_allocate_best (size_t num_points)
  allocate results arrays and labels for multipoint storage

• void archive_best (size_t index, const Variables &best_vars, const Response &best_resp)

  archive the best point into the results array

• void resize_best_vars_array (size_t newsize)
  safely resize the best variables array to newsize taking into account the envelope-letter design pattern and any recasting.

• void resize_best_resp_array (size_t newsize)
  safely resize the best response array to newsize taking into account the envelope-letter design pattern and any recasting.

### Static Protected Member Functions

• static void gnewton_set_recast (const Variables &recast_vars, const ActiveSet &recast_set, ActiveSet &sub_model_set)

  conversion of request vector values for the Gauss-Newton Hessian approximation

• static void replicate_set_recast (const Variables &recast_vars, const ActiveSet &recast_set, ActiveSet &sub_model_set)

  conversion of request vector values for Least Squares

• static void secondary_resp_copier (const Variables &input_vars, const Variables &output_vars, const Response &input_response, Response &output_response)

  copy the partial response for secondary functions when needed (data and reduction transforms)

### Protected Attributes

• 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

• Real constraintTol
  optimizer/least squares constraint tolerance

• Real bigRealBoundSize
  cutoff value for inequality constraint and continuous variable bounds
• int bigIntBoundSize
  cutoff value for discrete variable bounds
• size_t numNonlinearIneqConstraints
  number of nonlinear inequality constraints
• size_t numNonlinearEqConstraints
  number of nonlinear equality constraints
• size_t numLinearIneqConstraints
  number of linear inequality constraints
• size_t numLinearEqConstraints
  number of linear equality constraints
• int numNonlinearConstraints
  total number of nonlinear constraints
• int numLinearConstraints
  total number of linear constraints
• int numConstraints
  total number of linear and nonlinear constraints
• unsigned short minimizerRecasts
  number of RecastModels locally (in Minimizer or derived classes) layered over the initially passed in Model
• bool optimizationFlag
  flag for use where optimization and NLS must be distinguished
• size_t numUserPrimaryFns
  number of objective functions or least squares terms in the user’s model always initialize at Minimizer, even if overridden later
• size_t numIterPrimaryFns
  number of objective functions or least squares terms in iterator’s view always initialize at Minimizer, even if overridden later
• bool boundConstraintFlag
  convenience flag for denoting the presence of user-specified bound constraints. Used for method selection and error checking.
• bool speculativeFlag
  flag for speculative gradient evaluations
• String obsDataFilename
  filename from which to read observed data
• bool obsDataFlag
  flag indicating whether user-supplied data is active
• ExperimentData expData
  Container for experimental data to which to calibrate model using least squares or other formulations which minimize SSE.
• size_t numExperiments
  number of experiments
• IntVector numReplicates
  number of replicates
• size_t numRowsExpData
  number of total rows of data since we are allowing varying numbers of experiments and replicates per experiment
• bool scaleFlag
  flag for overall scaling status
• bool varsScaleFlag
  flag for variables scaling
• bool primaryRespScaleFlag
  flag for primary response scaling
• bool secondaryRespScaleFlag
  flag for secondary response scaling
• IntArray cvScaleTypes
  scale flags for continuous vars.
• RealVector cvScaleMultipliers
  scales for continuous variables
• RealVector cvScaleOffsets
  offsets for continuous variables
• IntArray responseScaleTypes
  scale flags for all responses
• RealVector responseScaleMultipliers
  scales for all responses
• RealVector responseScaleOffsets
  offsets for all responses (zero < for functions, not for nonlin con)
• IntArray linearIneqScaleTypes
  scale flags for linear ineq
• RealVector linearIneqScaleMultipliers
  scales for linear ineq constrs.
• RealVector linearIneqScaleOffsets
  offsets for linear ineq constrs.
• IntArray linearEqScaleTypes
  scale flags for linear eq.
• RealVector linearEqScaleMultipliers
  scales for linear constraints
• RealVector linearEqScaleOffsets
  offsets for linear constraints
• Minimizer * prevMinInstance
  pointer containing previous value of minimizerInstance
• bool vendorNumericalGradFlag
  convenience flag for gradient type == numerical && method source == vendor

Static Protected Attributes

• static Minimizer * minimizerInstance
  pointer to Minimizer used in static member functions
Private Member Functions

- bool data_difference_core (const Response &raw_response, Response &residual_response)
  Core of data difference, which doesn’t perform any output.
- void initialize_scaling ()
  initialize scaling types, multipliers, and offsets; perform error checking
- void compute_scaling (int object_type, int auto_type, int num_vars, RealVector &lbs, RealVector &ubs, RealVector &targets, const StringArray &scale_strings, const RealVector &scales, IntArray &scale_types, RealVector &scale_mults, RealVector &scale_offsets)
  general helper function for initializing scaling types and factors on a vector of variables, functions, constraints, etc.
- bool compute_scale_factor (const Real lower_bound, const Real upper_bound, Real *multiplier, Real *offset)
  automatically compute a single scaling factor – bounds case
- bool compute_scale_factor (const Real target, Real *multiplier)
  automatically compute a single scaling factor – target case
- void response_scaler_core (const Variables &native_vars, const Variables &scaled_vars, const Response &native_response, Response &iterator_response, size_t start_offset, size_t num_responses)
  Core of response scaling, which doesn’t perform any output.
- RealVector modify_n2s (const RealVector &native_vars, const IntArray &scale_types, const RealVector &multipliers, const RealVector &offsets) const
  general RealVector mapping from native to scaled variables vectors:
- void response_modify_n2s (const Variables &scaled_vars, const Variables &native_vars, const Response &native_response, Response &scaled_response, int start_offset, int num_responses) const
  map responses from native to scaled variable space
- RealMatrix lin_coeffs_modify_n2s (const RealMatrix &native_coeffs, const RealVector &cv_multipliers, const RealVector &lin_multipliers) const
  general linear coefficients mapping from native to scaled space
- void print_scaling (const String &info, const IntArray &scale_types, const RealVector &scale_mults, const RealVector &scale_offsets, const StringArray &labels)
  print scaling information for a particular response type in tabular form

Static Private Member Functions

- static void primary_resp_differencer (const Variables &raw_vars, const Variables &residual_vars, const Response &raw_response, Response &residual_response)
  Recast callback function to difference residuals with observed data.
- static void variables_scaler (const Variables &scaled_vars, Variables &native_vars)
  RecastModel callback for variables scaling: transform variables from scaled to native (user) space.
- static void primary_resp_scaler (const Variables &native_vars, const Variables &scaled_vars, const Response &native_response, Response &iterator_response)
  RecastModel callback for primary response scaling: transform responses (grads, Hessians) from native (user) to scaled space.
- static void secondary_resp_scaler (const Variables &native_vars, const Variables &scaled_vars, const Response &native_response, Response &scaled_response)
  RecastModel callback for secondary response scaling: transform constraints (grads, Hessians) from native (user) to scaled space.
Friends

- class SOLBase
  the SOLBase class is not derived from the iterator hierarchy but still needs access to iterator hierarchy data (to avoid attribute replication)
- class SNLLBase
  the SNLLBase class is not derived from the iterator hierarchy but still needs access to iterator hierarchy data (to avoid attribute replication)

13.59.1 Detailed Description

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

The Minimizer class provides common data and functionality for Optimizer and LeastSq.

13.59.2 Constructor & Destructor Documentation

Minimizer (ProblemDescDB & problem_db, Model & model) [protected]

standard constructor

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

References Iterator::iteratedModel, Iterator::maxIterations, Iterator::methodName, Iterator::numFinalSolutions, and Minimizer::update from model().

13.59.3 Member Function Documentation

void initialize_run() [protected], [virtual]

utility function to perform common operations prior to pre_run(); typically memory initialization; setting of instance pointers

Perform initialization phases of run sequence, like allocating memory and setting instance pointers. Commonly used in sub-iterator executions. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s initialize_run(), typically before performing its own implementation steps.

Reimplemented from Iterator.

Reimplemented in SNLLOptimizer, NLPQLPOptimizer, SNLLLeastSq, DOTOptimizer, and Optimizer.

References Model::all_continuous_variables(), Model::all_discrete_int_variables(), Model::all_discrete_real_variables(), Iterator::bestVariablesArray, Model::is_null(), Iterator::iteratedModel, Iterator::maxEvalConcurrency, Minimizer::minimizerInstance, Minimizer::minimizerRecasts, Minimizer::prevMinInstance, Model::set_communicators(), Model::set_evaluation_reference(), Iterator::subIteratorFlag, Model::subordinate_model(), and Iterator::summary-OutputFlag.

Referenced by LeastSq::initialize_run(), and Optimizer::initialize_run().

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

post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way

Post-run phase, which a derived iterator may optionally reimplement; when not present, post-run is likely integrated into run. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s post_run(), typically after performing its own implementation steps.

Reimplemented from Iterator.

Reimplemented in SNLLOptimizer, SNLLLeastSq, and Optimizer.
References Model::is\_null(), Iterator::iteratedModel, Model::print\_evaluation\_summary(), Iterator::print\_results(), Iterator::resultsDB, Iterator::summary\_OutputFlag, and ResultsManager::write\_databases().

Referenced by LeastSq::post\_run(), Optimizer::post\_run(), and SNLLLeastSq::post\_run().

```cpp
void finalize\_run( ) [inline], [protected], [virtual]
```

utility function to perform common operations following post\_run(): deallocation and resetting of instance pointers

Optional: perform finalization phases of run sequence, like deallocating memory and resetting instance pointers. Commonly used in sub-iterator executions. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s finalize\_run(), typically after performing its own implementation steps.

Reimplemented from Iterator.

Reimplemented in SNLLOptimizer, SNLLLeastSq, and Optimizer.

References Iterator::finalize\_run(), Minimizer::minimizer\_Instance, and Minimizer::prev\_Min\_Instance.

Referenced by LeastSq::finalize\_run(), and Optimizer::finalize\_run().

```cpp
const Model & algorithm\_space\_model( ) const [inline], [protected], [virtual]
```

default definition that gets redefined in selected derived Minimizers

Reimplemented from Iterator.

Reimplemented in EffGlobalMinimizer.

References Iterator::iteratedModel.

```cpp
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 Iterator.

```cpp
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 Iterator.

```cpp
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 Iterator.
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 environment master (either the dedicated master processor for self scheduling or peer 1 for static scheduling).

Reimplemented from Iterator.

bool data_transform_model ( bool weight_flag = false ) [protected]

Wrap iteratedModel in a RecastModel that subtracts provided observed data from the primary response functions (variables and secondary responses are unchanged)

Reads observation data to compute least squares residuals. Does not change size of responses, and is the first wrapper, therefore sizes are based on iteratedModel. This will set weights to \( \sigma[i]^2 \) if appropriate. weight_flag is true is there already exist user-specified weights in the calling context.

References Dakota::abort_handler(), Iterator::activeSet, Model::assign_rep(), Minimizer::expData, ProblemDescDB::get_bool(), ProblemDescDB::get_iv(), ProblemDescDB::get_size(), Iterator::iteratedModel, ExperimentData::load_scalar(), Minimizer::numContinuousVars, Minimizer::numExperiments, Minimizer::numFunctions, Minimizer::numNonlinearConstraints, Minimizer::numNonlinearIneqConstraints, Minimizer::numReplicates, Minimizer::numRowsExpData, Minimizer::numUserPrimaryFns, Minimizer::obsDataFilename, Iterator::outputLevel, Minimizer::primary_response_diffencer(), Model::primary_response_fn_sense(), Model::primary_response_fn_weights(), Iterator::probDescDB, Minimizer::replicate_set_recast(), ActiveSet::request_vector(), ExperimentData::scalar_data(), ExperimentData::scalar_sigma(), Minimizer::secondary_resp_copier(), and Model::subordinate_model().

Referenced by LeastSq::LeastSq(), and Optimizer::Optimizer().

void scale_model ( ) [protected]

Wrap iteratedModel in a RecastModel that performs variable and/or response scaling.

Wrap the iteratedModel in a scaling transformation, such that iteratedModel now contains a scaling recast model. Potentially affects variables, primary, and secondary responses

References Model::assign_rep(), Minimizer::cvScaleTypes, RecastModel::initialize(), Minimizer::initialize_scaling(), Iterator::iteratedModel, Model::model_rep(), Minimizer::numContinuousVars, Minimizer::numNonlinearConstraints, Minimizer::numNonlinearIneqConstraints, Minimizer::numUserPrimaryFns, Iterator::outputLevel, Minimizer::primary_resp_scaler(), Model::primary_response_fn_sense(), Model::primary_response_fn_weights(), Minimizer::primaryRespScaleFlag, Minimizer::responseScaleTypes, Minimizer::secondary_resp_scaler(), Minimizer::secondaryRespScaleFlag, Model::subordinate_model(), Minimizer::variables_scaler(), and Minimizer::varsScaleFlag.

Referenced by LeastSq::LeastSq(), and Optimizer::Optimizer().

void gnewton_set_recast ( const Variables & recast_vars, const ActiveSet & recast_set, ActiveSet & sub_model_set ) [static], [protected]

conversion of request vector values for the Gauss-Newton Hessian approximation

For Gauss-Newton Hessian requests, activate the 2 bit and mask the 4 bit.
References ActiveSet::request_value(), and ActiveSet::request_vector().

Referenced by Optimizer::reduce_model(), and SurrBasedLocalMinimizer::SurrBasedLocalMinimizer().

void secondary_resp_copier ( const Variables & input_vars, const Variables & output_vars, const Response & input_response, Response & output_response ) [static], [protected]

copy the partial response for secondary functions when needed (data and reduction transforms)

Constraint function map from user/native space to iterator/scaled/combined space using a RecastModel.
13.59. MINIMIZER CLASS REFERENCE

References Minimizer::minimizerInstance, Minimizer::numIterPrimaryFns, Minimizer::numNonlinearConstraints, Minimizer::numUserPrimaryFns, and Response::update().

Referenced by Minimizer::data_transform_model(), Optimizer::reduce_model(), and LeastSq::weight_model() for the following functions:

**bool need_resp_trans_byvars ( const ShortArray & asv, int start_index, int num_resp )** [protected]

determine if response transformation is needed due to variable transformations

Determine if variable transformations present and derivatives requested, which implies a response transformation is necessary

References Minimizer::varsScaleFlag.

Referenced by LeastSq::post_run(), Optimizer::post_run(), SNLLLeastSq::post_run(), and Minimizer::response_scaler().

**RealVector modify_s2n ( const RealVector & scaled_vars, const IntArray & scale_types, const RealVector & multipliers, const RealVector & offsets ) const [protected]**

general RealVector mapping from scaled to native variables (and values)

general RealVector mapping from scaled to native variables and/or vals; loosely, in greatest generality: scaled\_var = (LOG\_BASE\^scaled\_var) * multiplier + offset

Referenced by LeastSq::post_run(), Optimizer::post_run(), SNLLLeastSq::post_run(), and Minimizer::variables_scaler().

**void response_modify_s2n ( const Variables & native_vars, const Response & scaled_response, Response & native_response, int start_offset, int num_responses ) const [protected]**

map responses from scaled to native space

Unscaling response mapping: modifies response from scaled (iterator) to native (user) space. Maps num_responses starting at response offset

References Response::active_set(), Variables::acv(), Variables::all_continuous_variable_ids(), Variables::all_continuous_variables(), Variables::continuous_variable_ids(), Variables::continuous_variables(), Dakota::copy_data(), Variables::cv(), Minimizer::cvScaleMultipliers, Minimizer::cvScaleOffsets, Minimizer::cvScaleTypes, ActiveSet::derivative_vector(), Dakota::find_index(), Response::function_gradient_view(), Response::function_gradients(), Response::function_hessian_view(), Response::function_hessians(), Response::function_labels(), Response::function_value(), Response::function_values(), Variables::icv(), Variables::inactive_continuous_variable_ids(), Variables::inactive_continuous_variables(), Minimizer::numUserPrimaryFns, Iterator::outputLevel, ActiveSet::request_vector(), Minimizer::responseScaleMultipliers, Minimizer::responseScaleOffsets, Minimizer::responseScaleTypes, and Dakota::write_precision.

Referenced by LeastSq::post_run(), and Optimizer::post_run().

**Real objective ( const RealVector & fn_vals, const BoolDeque & max_sense, const RealVector & primary_wts ) const [protected]**

compute a composite objective value from one or more primary functions

The composite objective computation sums up the contributions from one of more primary functions using the primary response fn weights.

References Minimizer::numUserPrimaryFns.

Referenced by SurrBasedLocalMinimizer::approx_subprob_objective_eval(), SurrBasedMinimizer::augmented_lagrangian_merit(), EffGlobalMinimizer::expected_improvement(), SurrBasedMinimizer::lagrangian_merit(), Optimizer::objective_reduction(), SurrBasedMinimizer::penalty_merit(), COLINOptimizer::post_run(), SurrBasedMinimizer::update_filter(), and SurrBasedLocalMinimizer::update_penalty().
Real objective (const RealVector & fn vals, size_t num fns, const BoolDeque & max sense, const RealVector & primary wts) const [protected]
compute a composite objective with specified number of source primary functions, instead of userPrimaryFns.
This "composite" objective is a more general case of the previous objective(), but doesn't presume a reduction
map from user to iterated space. Used to apply weights and sense in COLIN results sorting. Leaving as a duplicate
implementation pending resolution of COLIN lookups.
References Minimizer::optimizationFlag.

void objective gradient (const RealVector & fn vals, size_t num fns, const RealMatrix & fn grads,
const BoolDeque & max sense, const RealVector & primary wts, RealVector & obj grad) const [protected]
compute the gradient of the composite objective function.
The composite objective gradient computation combines the contributions from one of more primary function
gradients, including the effect of any primary function weights. In the case of a linear mapping (MOO), only the
primary function gradients are required, but in the case of a nonlinear mapping (NLS), primary function values
are also needed. Within RecastModel::set_mapping(), the active set requests are automatically augmented to make
values available when needed, based on nonlinearRespMapping settings.
References Minimizer::numContinuousVars, and Minimizer::optimizationFlag.

void objective hessian (const RealVector & fn vals, size_t num fns, const RealMatrix & fn grads,
const RealSymMatrixArray & fn Hessians, const BoolDeque & max sense, const RealVector & primary wts,
RealSymMatrix & obj hess) const [protected]
compute the Hessian of the composite objective function.
The composite objective Hessian computation combines the contributions from one of more primary function
Hessians, including the effect of any primary function weights. In the case of a linear mapping (MOO), only the
primary function Hessians are required, but in the case of a nonlinear mapping (NLS), primary function values
and gradients are also needed in general (gradients only in the case of a Gauss-Newton approximation). Within
the default RecastModel::set_mapping(), the active set requests are automatically augmented to make values and
gradients available when needed, based on nonlinearRespMapping settings.
References Dakota::abort_handler(), Minimizer::numContinuousVars, and Minimizer::optimizationFlag.

void resize best vars array (size_t newsize) [protected]
Safely resize the best variables array to newsize taking into account the envelope-letter design pattern and any
recasting.
Uses data from the innermost model, should any Minimizer recasts be active. Called by multipoint return
solvers. Do not directly call resize on the bestVariablesArray object unless you intend to share the internal content
(letter) with other objects after assignment.
References Iterator::bestVariablesArray, Variables::copy(), Model::current_variables(), Iterator::iteratedModel,
Minimizer::minimizerRecasts, and Model::subordinate_model().
Referenced by COLINOptimizer::post_run().

void resize best resp array (size_t newsize) [protected]
Safely resize the best response array to newsize taking into account the envelope-letter design pattern and any
recasting.
13.59. MINIMIZER CLASS REFERENCE

Uses data from the innermost model, should any Minimizer recasts be active. Called by multipoint return solvers. Do not directly call resize on the bestResponseArray object unless you intend to share the internal content (letter) with other objects after assignment.

References Iterator::bestResponseArray, Response::copy(), Model::current_response(), Iterator::iteratedModel, Minimizer::minimizerRecasts, and Model::subordinate_model().

Referenced by COLINOptimizer::post_run().

void primary_resp_differencer ( const Variables & raw_vars, const Variables & residual_vars, const Response & raw_response, Response & residual_response ) [static], [private]

Recast callback function to difference residuals with observed data.

Difference the primary responses with observed data.

References Minimizer::data_difference_core(), Response::function_labels(), Response::function_values(), Minimizer::minimizerInstance, and Iterator::outputLevel.

Referenced by Minimizer::data_transform_model().

void initialize_scaling ( ) [private]

Initialize scaling types, multipliers, and offsets; perform error checking

Initialize scaling types, multipliers, and offsets. Update the iteratedModel appropriately

References Dakota::abort_handler(), Minimizer::compute_scaling(), Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), Model::continuous_variable_labels(), Model::continuous_variables(), Dakota::copy_data(), Minimizer::cvScaleMultipliers, Minimizer::cvScaleOffsets, Minimizer::cvScaleTypes, ProblemDescDB::get_rv(), ProblemDescDB::get_sa(), Iterator::iteratedModel, Minimizer::lin_coeffs_modify_n2s(), Model::linear_eq_constraint_coeffs(), Model::linear_eq_constraint_targets(), Model::linear_ineq_constraint_coeffs(), Model::linear_ineq_constraint_targets(), Model::linear_ineq_constraint_upper_bounds(), Minimizer::linearEqScaleMultipliers, Minimizer::linearEqScaleOffsets, Minimizer::linearEqScaleTypes, Model::model_rep(), Minimizer::modify_n2s(), Model::nonlinear_eq_constraint_coeffs(), Model::nonlinear_eq_constraint_targets(), Model::nonlinear_ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Minimizer::numLinearEqConstraints, Minimizer::numLinearIneqConstraints, Minimizer::numNonlinearEqConstraints, Minimizer::numNonlinearIneqConstraints, Minimizer::numUserPrimaryFns, Iterator::outputLevel, Minimizer::primaryRespScaleFlag, Minimizer::print_scaling(), Iterator::probDescDB, Model::response_labels(), Minimizer::responseScaleMultipliers, Minimizer::responseScaleOffsets, Minimizer::responseScaleTypes, Minimizer::secondaryRespScaleFlag, RecastModel::submodel_supports_derivative_estimation(), Model::subordinate_model(), Model::supports_derivative_estimation(), and Minimizer::varsScaleFlag.

Referenced by Minimizer::scale_model().

void variables_scaler ( const Variables & scaled_vars, Variables & native_vars ) [static], [private]

RecastModel callback for variables scaling: transform variables from scaled to native (user) space.

Variables map from iterator/scaled space to user/native space using a RecastModel.

References Variables::continuous_variable_labels(), Variables::continuous_variables(), Minimizer::cvScaleMultipliers, Minimizer::cvScaleOffsets, Minimizer::cvScaleTypes, Minimizer::minimizerInstance, Minimizer::modify_s2n(), and Iterator::outputLevel.

Referenced by Minimizer::scale_model().
void secondary_resp_scaler ( const Variables & native_vars, const Variables & scaled_vars, const Response & native_response, Response & iterator_response ) [static], [private]

RecastModel callback for secondary response scaling: transform constraints (grads, Hessians) from native (user) to scaled space.
Constraint function map from user/native space to iterator/scaled/combined space using a RecastModel.
References Minimizer::minimizerInstance, Minimizer::numNonlinearConstraints, Minimizer::numUserPrimaryFns, and Minimizer::response_scaler_core().
Referenced by Minimizer::scale_model().

RealVector modify_n2s ( const RealVector & native_vars, const IntArray & scale_types, const RealVector & multipliers, const RealVector & offsets ) const [private]

general RealVector mapping from native to scaled variables vectors:
general RealVector mapping from native to scaled variables; loosely, in greatest generality: scaled_var = log((native_var - offset) / multiplier )
Referenced by Minimizer::initialize_scaling().

void response_modify_n2s ( const Variables & native_vars, const Response & native_response, Response & recast_response, int start_offset, int num_responses ) const [private]

map responses from native to scaled variable space
Scaling response mapping: modifies response from a model (user/native) for use in iterators (scaled). Maps num_responses starting at response_offset
References Response::active_set(), Variables::acv(), Variables::all_continuous_variable_ids(), Variables::all_continuous_variables(), Variables::continuous_variable_ids(), Variables::continuous_variables(), Dakota::copy_data(), Variables::cv(), Minimizer::cvScaleMultipliers, Minimizer::cvScaleOffsets, Minimizer::cvScaleTypes, ActiveSet::derivative_vector(), Dakota::find_index(), Response::function_gradient_view(), Response::function_gradients(), Response::function_hessian_view(), Response::function_hessians(), Response::function_value(), Response::function_values(), Variables::icv(), Variables::inactive_continuous_variable_ids(), Variables::inactive_continuous_variables(), Minimizer::numUserPrimaryFns, Iterator::outputLevel, ActiveSet::request_vector(), Minimizer::responseScaleMultipliers, Minimizer::responseScaleOffsets, Minimizer::responseScaleTypes, and Dakota::write_precision.
Referenced by Minimizer::response_scaler_core().

RealMatrix lin_coeffs_modify_n2s ( const RealMatrix & src_coeffs, const RealVector & cv_multipliers, const RealVector & lin_multipliers ) const [private]

general linear coefficients mapping from native to scaled space
compute scaled linear constraint matrix given design variable multipliers and linear scaling multipliers. Only scales components corresponding to continuous variables so for src_coeffs of size MxN, lin_multipliers.size() <= M, cv_multipliers.size() <= N
Referenced by Minimizer::initialize_scaling().
The documentation for this class was generated from the following files:

- DakotaMinimizer.hpp
- DakotaMinimizer.cpp
13.60 MixedVarConstraints Class Reference

Derived class within the Constraints hierarchy which separates continuous and discrete variables (no domain type array merging).

Inheritance diagram for MixedVarConstraints:

```
    Constraints
       |__________
        |          |
        V          V
MixedVarConstraints
```

Public Member Functions

- MixedVarConstraints (const SharedVariablesData &svd)
  lightweight constructor
- MixedVarConstraints (const ProblemDescDB &problem_db, const SharedVariablesData &svd)
  standard constructor
- ~MixedVarConstraints ()
  destructor
- void write (std::ostream &s) const
  write a variable constraints object to an std::ostream
- void read (std::istream &s)
  read a variable constraints object from an std::istream

Protected Member Functions

- void reshape (const SizetArray &vc_totals)
  reshape the lower/upper bound arrays within the Constraints hierarchy
- void build_active_views ()
  construct active views of all variables bounds arrays
- void build_inactive_views ()
  construct inactive views of all variables bounds arrays

Additional Inherited Members

13.60.1 Detailed Description

Derived class within the Constraints hierarchy which separates continuous and discrete variables (no domain type array merging).

Derived variable constraints classes take different views of the design, uncertain, and state variable types and the continuous and discrete domain types. The MixedVarConstraints derived class separates the continuous and discrete domain types (see Variables::get_variables(problem_db) for variables type selection; variables type is passed to the Constraints constructor in Model).
13.60.2 Constructor & Destructor Documentation

MixedVarConstraints ( const ProblemDescDB & problem_db, const SharedVariablesData & svd )

standard constructor

In this class, mixed continuous/discrete variables are used. Most iterators/strategies use this approach, which is the default in Constraints::get_constraints().

References Constraints::allContinuousLowerBnds, Constraints::allContinuousUpperBnds, Constraints::allDiscreteIntLowerBnds, Constraints::allDiscreteIntUpperBnds, Constraints::allDiscreteRealLowerBnds, Constraints::allDiscreteRealUpperBnds, Constraints::build_views(), SharedVariablesData::components_totals(), Dakota::copy_data_partial(), ProblemDescDB::get_iv(), ProblemDescDB::get_rv(), Constraints::manage_linear_constraints(), Constraints::numLinearEqCons, Constraints::numLinearIneqCons, Constraints::sharedVarsData, SharedVariablesData::vc_lookup(), and SharedVariablesData::view().

13.60.3 Member Function Documentation

void reshape ( const SizetArray & vc_totals ) [protected], [virtual]

reshape the lower/upper bound arrays within the Constraints hierarchy

Resizes the derived bounds arrays.

Reimplemented from Constraints.

References Constraints::allContinuousLowerBnds, Constraints::allContinuousUpperBnds, Constraints::allDiscreteIntLowerBnds, Constraints::allDiscreteIntUpperBnds, Constraints::allDiscreteRealLowerBnds, Constraints::allDiscreteRealUpperBnds, Constraints::build_views(), and Constraints::reshape().

Referenced by MixedVarConstraints::MixedVarConstraints().

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

- MixedVarConstraints.hpp
- MixedVarConstraints.cpp

13.61 MixedVariables Class Reference

Derived class within the Variables hierarchy which separates continuous and discrete variables (no domain type array merging).

Inheritance diagram for MixedVariables:

```
Variables
```

MixedVariables

Public Member Functions

- MixedVariables (const ProblemDescDB & problem_db, const std::pair< short, short > & view)
  standard constructor
- MixedVariables (const SharedVariablesData & svd)
  lightweight constructor
- ~MixedVariables ()
  destructor
Protected Member Functions

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

- void write (std::ostream &s) const
  
  write a variables object to an std::ostream

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

- void read_tabular (std::istream &s)

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

- void reshape (const SizetArray &vc_totals)
  
  reshapes an existing Variables object based on the incoming variablesComponents

- void build_active_views ()
  
  construct active views of all variables arrays

- void build_inactive_views ()
  
  construct inactive views of all variables arrays

Additional Inherited Members

13.61.1 Detailed Description

Derived class within the Variables hierarchy which separates continuous and discrete variables (no domain type array merging).

Derived variables classes take different views of the design, uncertain, and state variable types and the continuous and discrete domain types. The MixedVariables derived class separates the continuous and discrete domain types (see Variables::get_variables(problem_db)).

13.61.2 Constructor & Destructor Documentation

MixedVariables ( const ProblemDescDB &problem_db, const std::pair< short, short > &view )

standard constructor

In this class, the distinct approach is used (design, uncertain, and state variable types and continuous and discrete domain types are distinct). Most iterators/strategies use this approach.

References Variables::allContinuousVars, Variables::allDiscreteIntVars, Variables::allDiscreteRealVars, Variables::::build_views(), SharedVariablesData::components_totals(), Dakota::copy_data_partial(), ProblemDescDB::get_iv(), ProblemDescDB::get_rv(), Variables::sharedVarsData, SharedVariablesData::vc_lookup(), and SharedVariablesData::view().

13.61.3 Member Function Documentation

void read_tabular ( std::istream & s ) [protected], [virtual]

Presumes variables object is already appropriately sized to receive!

Reimplemented from Variables.

References Variables::allContinuousVars, Variables::allDiscreteIntVars, and Variables::allDiscreteRealVars.

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

- MixedVariables.hpp
- MixedVariables.cpp
13.62 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 ()**
  return a single sub-model defined from subModel in nested and recast models and truth_model() in surrogate models; used for a directed dive through model recursions that may bypass some components.
- **virtual Model & surrogate_model ()**
  return the approximation sub-model in surrogate models
- **virtual Model & truth_model ()**
  return the truth sub-model in surrogate models
- **virtual void derived_subordinate_models (ModelList &ml, bool recurse_flag)**
  portion of subordinate_models() specific to derived model classes
- **virtual void update_from_subordinate_model (bool recurse_flag=true)**
  propagate vars/labels/bounds/targets from the bottom up
- **virtual Interface & derived_interface ()**
  return the interface employed by the derived model class, if present: SingleModel::userDefinedInterface, DataFitSurrModel::approxInterface, or NestedModel::optionalInterface
- **virtual void primary_response_fn_weights (const RealVector &wts, bool recurse_flag=true)**
  set the relative weightings for multiple objective functions or least squares terms
- **virtual void surrogate_function_indices (const IntSet &surr_fn_indices)**
  set the (currently active) surrogate function index set
• virtual void build_approximation ()
  build a new SurrogateModel approximation
• virtual bool build_approximation (const Variables &vars, const IntResponsePair &response_pr)
  build a new SurrogateModel approximation using/enforcing response at vars
• virtual void update_approximation (bool rebuild_flag)
  replace the approximation data within an existing surrogate based on data updates propagated elsewhere
• virtual void update_approximation (const Variables &vars, const IntResponsePair &response_pr, bool rebuild_flag)
  replace the anchor point data within an existing surrogate
• virtual void update_approximation (const VariablesArray &vars_array, const IntResponseMap &resp_map, bool rebuild_flag)
  replace the data points within an existing surrogate
• virtual void append_approximation (bool rebuild_flag)
  append to the existing approximation data within a surrogate based on data updates propagated elsewhere
• virtual void append_approximation (const Variables &vars, const IntResponsePair &response_pr, bool rebuild_flag)
  append a single point to an existing surrogate’s data
• virtual void append_approximation (const VariablesArray &vars_array, const IntResponseMap &resp_map, bool rebuild_flag)
  append multiple points to an existing surrogate’s data
• virtual void pop_approximation (bool save_surr_data, bool rebuild_flag=false)
  remove the previous data set addition to a surrogate (e.g., due to a previous append_approximation() call); flag manages storing of surrogate data for use in a subsequent restore_approximation() call
• virtual void restore_approximation ()
  restore a previous approximation data state within a surrogate
• virtual bool restore_available ()
  query for whether a trial increment is restorable within a surrogate
• virtual void finalize_approximation ()
  finalize an approximation by applying all previous trial increments
• virtual void store_approximation ()
  move the current approximation into storage for later combination
• virtual void combine_approximation (short corr_type)
  combine the current approximation with one previously stored
• virtual bool force_rebuild ()
  determine whether a surrogate model rebuild should be forced based on changes in the inactive data
• virtual SharedApproxData & shared_approximation ()
  retrieve the shared approximation data within the ApproximationInterface of a DataFitSurrModel
• virtual std::vector< Approximation > & approximations ()
  retrieve the set of Approximations within the ApproximationInterface of a DataFitSurrModel
• virtual const Pecos::SurrogateData & approximation_data (size_t index)
  retrieve the approximation data from a particular Approximation instance within the ApproximationInterface of a DataFitSurrModel
virtual const RealVectorArray & approximation_coeffs ()
        retrieve the approximation coefficients from each Approximation within a DataFitSurrModel
virtual void approximation_coeffs (const RealVectorArray &approx_coeffs)
        set the approximation coefficients for each Approximation within a DataFitSurrModel
virtual const RealVector & approximation_variances (const Variables &vars)
        retrieve the approximation variances from each Approximation within a DataFitSurrModel
virtual void surrogate_response_mode (short mode)
        set response computation mode used in SurrogateModels for forming currentResponse
virtual short surrogate_response_mode () const
        return response computation mode used in SurrogateModels for forming currentResponse
virtual DiscrepancyCorrection & discrepancy_correction ()
        return the DiscrepancyCorrection object used by SurrogateModels
virtual void component_parallel_mode (short mode)
        update component parallel mode for supporting parallelism in a model’s interface component, sub-model component, or neither component [componentParallelMode = 0 (none), 1 (INTERFACE/APPROX_INTERFACE/OPTIONAL_INTERFACE/INTERFACE/LF_MODEL/SURROGATE_MODEL), or 2 (SUB_MODEL/ACTUAL_MODEL/HF_MODEL/TRUE_MODEL)].
virtual String local_eval_synchronization ()
        return derived model synchronization setting
virtual int local_eval_concurrency ()
        return derived model asynchronous evaluation concurrency
virtual void serve (int max_eval_concurrency)
        Service job requests received from the master. Completes when a termination message is received from stop_servers().
virtual void stop_servers ()
        Executed by the master to terminate all server operations for a particular model when iteration on the model is complete.
virtual bool derived_master_overload () const
        Return a flag indicating the combination of multiprocessor evaluations and a dedicated master iterator scheduling. Used in synchronous compute_response functions to prevent the error of trying to run a multiprocessor job on the master.
virtual void inactive_view (short view, bool recurse_flag=true)
        update the Model’s inactive view based on higher level (nested) context
virtual const String & interface_id () const
        return the interface identifier
virtual int evaluation_id () const
        Return the value of the evaluation id counter for the Model.
virtual bool evaluation_cache () const
        Indicates the usage of an evaluation cache by the Model.
virtual void set_evaluation_reference ()
        Set the reference points for the evaluation counters within the Model.
virtual void fine_grained_evaluation_counters ()
        Request fine-grained evaluation reporting within the Model.
virtual void print_evaluation_summary (std::ostream &, bool minimal_header=false, bool relative_count=true) const
Print an evaluation summary for the Model.

- virtual void eval_tag_prefix (const String &eval_id_str)
  set the hierarchical eval ID tag prefix
- ModelList & subordinate_models (bool recurse_flag=true)
  return the sub-models in nested and surrogate models
- void compute_response ()
  Compute the Response at currentVariables (default ActiveSet).
- void compute_response (const ActiveSet &set)
  Compute the Response at currentVariables (specified ActiveSet).
- void asynch_compute_response ()
  Spawn an asynchronous job (or jobs) that computes the value of the Response at currentVariables (default ActiveSet).
- void asynch_compute_response (const ActiveSet &set)
  Spawn an asynchronous job (or jobs) that computes the value of the Response at currentVariables (specified ActiveSet).
- const IntResponseMap & synchronize ()
  Execute a blocking scheduling algorithm to collect the complete set of results from a group of asynchronous evaluations.
- const IntResponseMap & synchronize_nowait ()
  Execute a nonblocking scheduling algorithm to collect all available results from a group of asynchronous evaluations.
- void init_communicators (int max_eval_concurrency, bool recurse_flag=true)
  allocate communicator partitions for a model and store configuration in modelPCIterMap
- void init_serial ()
  for cases where init_communicators() will not be called, modify some default settings to behave properly in serial.
- void set_communicators (int max_eval_concurrency, bool recurse_flag=true)
  set active parallel configuration for the model (set modelPCIter from modelPCIterMap)
- void free_communicators (int max_eval_concurrency, bool recurse_flag=true)
  deallocate communicator partitions for a model
- MPI_Comm analysis_comm () const
  retrieve the MPI communicator on which this model is configured to conduct function evaluation analyses (provided for library clients)
- void stop_configurations ()
  called from Strategy::init_iterator() for iteratorComm rank 0 to terminate serve_configurations() on other iterator-Comm processors
- int serve_configurations ()
  called from Strategy::init_iterator() for iteratorComm rank != 0 to balance init_communicators() calls on iterator-Comm rank 0
- void estimate_message_lengths ()
  estimate messageLengths for a model
- void assign_rep (Model *model_rep, bool ref_count_incr=true)
  replaces existing letter with a new one
- size_t tv () const
  returns total number of vars
• size_t cv () const
  returns number of active continuous variables
• size_t div () const
  returns number of active discrete integer vars
• size_t drv () const
  returns number of active discrete real vars
• size_t icv () const
  returns number of inactive continuous variables
• size_t idiv () const
  returns number of inactive discrete integer vars
• size_t idrv () const
  returns number of inactive discrete real vars
• size_t acv () const
  returns total number of continuous variables
• size_t adiv () const
  returns total number of discrete integer vars
• size_t adrv () const
  returns total number of discrete real vars
• void active_variables (const Variables &vars)
  set the active variables in currentVariables
• const RealVector & continuous_variables () const
  return the active continuous variables from currentVariables
• void continuous_variables (const RealVector &c_vars)
  set the active continuous variables in currentVariables
• void continuous_variable (Real c_var, size_t i)
  set an active continuous variable in currentVariables
• const IntVector & discrete_int_variables () const
  return the active discrete integer variables from currentVariables
• void discrete_int_variables (const IntVector &d_vars)
  set the active discrete integer variables in currentVariables
• void discrete_int_variable (int d_var, size_t i)
  set an active discrete integer variable in currentVariables
• const RealVector & discrete_real_variables () const
  return the active discrete real variables from currentVariables
• void discrete_real_variables (const RealVector &d_vars)
  set the active discrete real variables in currentVariables
• void discrete_real_variable (Real d_var, size_t i)
  set an active discrete real variable in currentVariables
• UShortMultiArrayConstView continuous_variable_types () const
  return the active continuous variable types from currentVariables
• void continuous_variable_types (UShortMultiArrayConstView cv_types)
  set the active continuous variable types in currentVariables
• void continuous_variable_type (unsigned short cv_type, size_t i)
set an active continuous variable type in currentVariables

- UShortMultiArrayConstView discrete_int_variable_types () const
  return the active discrete variable types from currentVariables
- void discrete_int_variable_types (UShortMultiArrayConstView div_types)
  set the active discrete variable types in currentVariables
- void discrete_int_variable_type (unsigned short div_type, size_t i)
  set an active discrete variable type in currentVariables
- UShortMultiArrayConstView discrete_real_variable_types () const
  return the active discrete variable types from currentVariables
- void discrete_real_variable_types (UShortMultiArrayConstView drv_types)
  set the active discrete variable types in currentVariables
- void discrete_real_variable_type (unsigned short drv_type, size_t i)
  set an active discrete variable type in currentVariables
- SizetMultiArrayConstView continuous_variable_ids () const
  return the active continuous variable identifiers from currentVariables
- void continuous_variable_ids (SizetMultiArrayConstView cv_ids)
  set the active continuous variable identifiers in currentVariables
- void continuous_variable_id (size_t cv_id, size_t i)
  set an active continuous variable identifier in currentVariables
- const RealVector & inactive_continuousVariables () const
  return the inactive continuous variables in currentVariables
- void inactive_continuousVariables (const RealVector &i_c_vars)
  set the inactive continuous variables in currentVariables
- const IntVector & inactive_discrete_int_variables () const
  return the inactive discrete variables in currentVariables
- void inactive_discrete_int_variables (const IntVector &i_d_vars)
  set the inactive discrete variables in currentVariables
- const RealVector & inactive_discrete_real_variables () const
  return the inactive discrete variables in currentVariables
- void inactive_discrete_real_variables (const RealVector &i_d_vars)
  set the inactive discrete variables in currentVariables
- UShortMultiArrayConstView inactive_continuous_variable_types () const
  return the inactive continuous variable types from currentVariables
- SizetMultiArrayConstView inactive_continuous_variable_ids () const
  return the inactive continuous variable identifiers from currentVariables
- 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 (Real a_c_var, 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
CHAPTER 13. CLASS DOCUMENTATION

• void all_discrete_int_variables (const IntVector &a_d_vars)
  set all discrete variables in currentVariables
• void all_discrete_int_variable (int a_d_var, 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 (Real a_d_var, size_t i)
  set a variable within the all discrete variables in currentVariables
• UShortMultiArrayConstView all_discrete_int_variable_types () const
  return all discrete variable types from currentVariables
• UShortMultiArrayConstView all_discrete_int_variable_types () const
  return all discrete variable types from currentVariables
• UShortMultiArrayConstView all_discrete_real_variable_types () const
  return all discrete variable types from currentVariables
• SizetMultiArrayConstView all_continuous_variable_ids () const
  return all continuous variable identifiers from currentVariables
• const IntSetArray & discrete_design_set_int_values () const
  return the sets of values available for each of the discrete design set integer variables
• void discrete_design_set_int_values (const IntSetArray &isa)
  define the sets of values available for each of the discrete design set integer variables
• const RealSetArray & discrete_design_set_real_values () const
  return the sets of values available for each of the discrete design set real variables
• void discrete_design_set_real_values (const RealSetArray &rsa)
  define the sets of values available for each of the discrete design set real variables
• const IntSetArray & discrete_state_set_int_values () const
  return the sets of values available for each of the discrete state set integer variables
• void discrete_state_set_int_values (const IntSetArray &isa)
  define the sets of values available for each of the discrete state set integer variables
• const RealSetArray & discrete_state_set_real_values () const
  return the sets of values available for each of the discrete state set real variables
• void discrete_state_set_real_values (const RealSetArray &rsa)
  define the sets of values available for each of the discrete state set real variables
• const BitArray & discrete_int_sets ()
  define and return discreteIntSets
• const IntSetArray & discrete_set_int_values ()
  return the sets of values available for each of the active discrete set integer variables (aggregated in activeDiscSet-IntValues)
• const RealSetArray & discrete_set_real_values ()
  return the sets of values available for each of the active discrete set real variables (aggregated in activeDiscSet-RealValues)
• Pecos::AleatoryDistParams & aleatory_distribution_parameters ()
return aleatDistParams
• const Pecos::AleatoryDistParams & aleatory_distribution_parameters () const
  return aleatDistParams
• void aleatory_distribution_parameters (const Pecos::AleatoryDistParams &adp)
  set aleatDistParams
• Pecos::EpistemicDistParams & epistemic_distribution_parameters ()
  return epistDistParams
• const Pecos::EpistemicDistParams & epistemic_distribution_parameters () const
  return epistDistParams
• void epistemic_distribution_parameters (const Pecos::EpistemicDistParams &edp)
  set epistDistParams
• Real continuous_probability_density () const
  compute a multivariate probability density from the marginals for the active aleatory random variables (must be uncorrelated)
• std::pair<Real, Real> continuous_distribution_bounds (size_t cv_index) const
  return a set of distribution bounds for a particular index within the active aleatory random 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, size_t i)
  set a label within the all continuous labels in currentVariables

StringMultiArrayConstView all_discrete_int_variable_labels () const
  return all discrete variable labels in currentVariables

• void all_discrete_int_variable_labels (StringMultiArrayConstView a_d_v_labels)
  set all discrete variable labels in currentVariables

• void all_discrete_int_variable_label (const String &a_d_v_label, size_t i)
  set a label within the all discrete labels in currentVariables

StringMultiArrayConstView all_discrete_real_variable_labels () const
  return all discrete variable labels in currentVariables

• void all_discrete_real_variable_labels (StringMultiArrayConstView a_d_v_labels)
  set all discrete variable labels in currentVariables

• void all_discrete_real_variable_label (const String &a_d_v_label, size_t i)
  set a label within the all discrete labels in currentVariables

const StringArray & response_labels () const
  return the response labels from currentResponse

• void response_labels (const StringArray &resp_labels)
  set the response labels in currentResponse

const RealVector & continuous_lower_bounds () const
  return the active continuous lower bounds from userDefinedConstraints

• void continuous_lower_bounds (const RealVector &c_l_bnds)
  set the active continuous lower bounds in userDefinedConstraints

• void continuous_lower_bound (Real c_l_bnd, size_t i)
  set the i-th active continuous lower bound in userDefinedConstraints

const RealVector & continuous_upper_bounds () const
  return the active continuous upper bounds from userDefinedConstraints

• void continuous_upper_bounds (const RealVector &c_u_bnds)
  set the active continuous upper bounds in userDefinedConstraints

• void continuous_upper_bound (Real c_u_bnd, size_t i)
  set the i-th active continuous upper bound from userDefinedConstraints

const IntVector & discrete_int_lower_bounds () const
  return the active discrete int lower bounds from userDefinedConstraints

• void discrete_int_lower_bounds (const IntVector &d_l_bnds)
  set the active discrete int lower bounds in userDefinedConstraints

• void discrete_int_lower_bound (int d_l_bnd, size_t i)
  set the i-th active discrete int lower bound in userDefinedConstraints

const IntVector & discrete_int_upper_bounds () const
  return the active discrete int upper bounds from userDefinedConstraints

• void discrete_int_upper_bounds (const IntVector &d_u_bnds)
  set the active discrete int upper bounds in userDefinedConstraints

• void discrete_int_upper_bound (int d_u_bnd, size_t i)
  set the i-th active discrete int upper bound in userDefinedConstraints
• const RealVector & discrete_real_lower_bounds () const
  return the active discrete real lower bounds from userDefinedConstraints
• void discrete_real_lower_bounds (const RealVector & d_l_bnds)
  set the active discrete real lower bounds in userDefinedConstraints
• void discrete_real_lower_bound (Real d_l_bnd, size_t i)
  set the i-th active discrete real lower bound in userDefinedConstraints
• const RealVector & discrete_real_upper_bounds () const
  return the active discrete real upper bounds from userDefinedConstraints
• void discrete_real_upper_bounds (const RealVector & d_u_bnds)
  set the active discrete real upper bounds in userDefinedConstraints
• void discrete_real_upper_bound (Real d_u_bnd, size_t i)
  set the i-th active discrete real upper bound in userDefinedConstraints
• const RealVector & inactive_continuous_lower_bounds () const
  return the inactive continuous lower bounds in userDefinedConstraints
• void inactive_continuous_lower_bounds (const RealVector & i_c_l_bnds)
  set the inactive continuous lower bounds in userDefinedConstraints
• const RealVector & inactive_continuous_upper_bounds () const
  return the inactive continuous upper bounds in userDefinedConstraints
• void inactive_continuous_upper_bounds (const RealVector & i_c_u_bnds)
  set the inactive continuous upper bounds in userDefinedConstraints
• const IntVector & inactive_discrete_int_lower_bounds () const
  return the inactive discrete integer lower bounds in userDefinedConstraints
• void inactive_discrete_int_lower_bounds (const IntVector & i_d_l_bnds)
  set the inactive discrete integer lower bounds in userDefinedConstraints
• const IntVector & inactive_discrete_int_upper_bounds () const
  return the inactive discrete integer upper bounds in userDefinedConstraints
• void inactive_discrete_int_upper_bounds (const IntVector & i_d_u_bnds)
  set the inactive discrete integer upper bounds in userDefinedConstraints
• const RealVector & inactive_discrete_real_lower_bounds () const
  return the inactive discrete real lower bounds in userDefinedConstraints
• void inactive_discrete_real_lower_bounds (const RealVector & i_d_l_bnds)
  set the inactive discrete real lower bounds in userDefinedConstraints
• const RealVector & inactive_discrete_real_upper_bounds () const
  return the inactive discrete real upper bounds in userDefinedConstraints
• void inactive_discrete_real_upper_bounds (const RealVector & i_d_u_bnds)
  set the inactive discrete real 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 (Real a_c_l_bnd, size_t i)
  set a lower bound within continuous lower bounds in 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 (Real a_c_u_bnd, size_t i)
  set a lower bound within continuous upper bounds in userDefinedConstraints
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 (Real a_c_u_bnd, size_t i)
  set an upper bound within all continuous upper bounds in 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 (int a_d_l_bnd, size_t i)
  set a lower bound within all discrete lower bounds in userDefinedConstraints

• const IntVector & all_discrete_int_upper_bounds () const
  return all discrete upper bounds in userDefinedConstraints

• void all_discrete_int_upper_bounds (const IntVector &a_d_u_bnds)
  set all discrete upper bounds in userDefinedConstraints

• void all_discrete_int_upper_bound (int a_d_u_bnd, size_t i)
  set an upper bound within all discrete upper bounds in userDefinedConstraints

• const RealVector & all_discrete_real_lower_bounds () const
  return all discrete lower bounds in userDefinedConstraints

• void all_discrete_real_lower_bounds (const RealVector &a_d_l_bnds)
  set all discrete lower bounds in userDefinedConstraints

• void all_discrete_real_lower_bound (Real a_d_l_bnd, size_t i)
  set a lower bound within all discrete lower bounds in userDefinedConstraints

• const RealVector & all_discrete_real_upper_bounds () const
  return all discrete upper bounds in userDefinedConstraints

• void all_discrete_real_upper_bounds (const RealVector &a_d_u_bnds)
  set all discrete upper bounds in userDefinedConstraints

• void all_discrete_real_upper_bound (Real a_d_u_bnd, size_t i)
  set an upper bound within all discrete upper bounds in userDefinedConstraints

• size_t num_linear_ineq_constraints () const
  return the number of linear inequality constraints

• size_t num_linear_eq_constraints () const
  return the number of linear equality constraints

• const RealMatrix & linear_ineq_constraint_coeff() (const RealMatrix &lin_ineq_coeffs)
  return the linear inequality constraint coefficients

• void linear_ineq_constraint_coeff() (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 Variables & current_variables () const  
  return the current variables (currentVariables) as const reference (preferred)
• Variables & current_variables ()  
  return the current variables (currentVariables) in mutable form (special cases)
• const Constraints & user_defined_constraints () const  
  return the user-defined constraints (userDefinedConstraints)
• const Response & current_response () const  
  return the current response (currentResponse)
• ProblemDescDB & problem_description_db () const  
  return the problem description database (probDescDB)
• ParallelLibrary & parallel_library () const  
  return the parallel library (parallelLib)
• const String & model_type () const  
  return the model type (modelType)
• const String & surrogate_type () const  
  return the surrogate type (surrogateType)
• const String & model_id () const
return the model identifier (modelId)

- `size_t num_functions () const`
  return number of functions in currentResponse

- `const String & gradient_type () const`
  return the gradient evaluation type (gradientType)

- `const String & method_source () const`
  return the numerical gradient evaluation method source (methodSource)

- `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 (fdGradStepSize)

- `const String & fd_gradient_step_type () const`
  return the finite difference gradient step type (fdGradStepType)

- `const IntSet & gradient_id_analytic () const`
  return the mixed gradient analytic IDs (gradIdAnalytic)

- `const IntSet & gradient_id_numerical () const`
  return the mixed gradient numerical IDs (gradIdNumerical)

- `const String & hessian_type () const`
  return the Hessian evaluation type (hessianType)

- `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 (fdHessByGradStepSize)

- `const RealVector & fd_hessian_by_fn_step_size () const`
  return function-based finite difference Hessian step size (fdHessByFnStepSize)

- `const String & fd_hessian_step_type () const`
  return the finite difference Hessian step type (fdHessStepType)

- `const IntSet & hessian_id_analytic () const`
  return the mixed Hessian analytic IDs (hessIdAnalytic)

- `const IntSet & hessian_id_numerical () const`
  return the mixed Hessian analytic IDs (hessIdNumerical)

- `const IntSet & hessian_id_quasi () const`
  return the mixed Hessian analytic IDs (hessIdQuasi)

- `void primary_response_fn_sense (const BoolDeque &sense)`
  set the optimization sense for multiple objective functions

- `const BoolDeque & primary_response_fn_sense () const`
  get the optimization sense for multiple objective functions

- `const RealVector & primary_response_fn_weights () const`
  get the relative weightings for multiple objective functions or least squares terms
- **bool** `derivative_estimation()`  
  *indicates potential usage of estimate_derivatives() based on gradientType/hessianType*

- **void** `supports_derivative_estimation(bool sed_flag)`  
  *set whether this model should perform or pass on derivative estimation*

- **void** `init_comms_bcast_flag(bool icb_flag)`  
  *set initCommsBcastFlag*

- **int** `evaluation_capacity() const`  
  *return the evaluation capacity for use in iterator logic*

- **int** `derivative_concurrency() const`  
  *return the gradient concurrency for use in parallel configuration logic*

- **bool** `async_flag() const`  
  *return the asynchronous evaluation flag (asyncEvalFlag)*

- **void** `async_flag(const bool flag)`  
  *set the asynchronous evaluation flag (asyncEvalFlag)*

- **short** `output_level() const`  
  *return the outputLevel*

- **void** `output_level(const short level)`  
  *set the outputLevel*

- **const IntArray &** `message_lengths() const`  
  *return the array of MPI packed message buffer lengths (messageLengths)*

- **void** `parallel_configuration_iterator(const ParConfigLIter &pc_iter)`  
  *set modelPCIter*

- **const ParConfigLIter &** `parallel_configuration_iterator() const`  
  *return modelPCIter*

- **void** `auto_graphics(const bool flag)`  
  *set modelAutoGraphicsFlag to activate posting of graphics data within compute_response/synchronize functions (automatic graphics posting in the model as opposed to graphics posting at the strategy level).*

- **bool** `is_null() const`  
  *function to check modelRep (does this envelope contain a letter)*

- **Model * model_rep() const**  
  *returns modelRep for access to derived class member functions that are not mapped to the top Model level*

- **Real** `initialize_h(Real x_j, Real lb_j, Real ub_j, Real step_size, String step_type)`  
  *function to determine initial finite difference h (before step length adjustment) based on type of step desired*

- **Real** `FDstep1(Real x0_j, Real lb_j, Real ub_j, Real h_mag)`  
  *function returning finite-difference step size (affected by bounds)*

- **Real** `FDstep2(Real x0_j, Real lb_j, Real ub_j, Real h)`  
  *function returning second central-difference step size (affected by bounds)*

### Public Attributes

- **bool** `shortStep`  
  *flags finite-difference step size adjusted by bounds*
Protected Member Functions

- **Model** (BaseConstructor, ProblemDescDB &problem_db)
  constructor initializing the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

- **Model** (NoDBBaseConstructor, ParallelLibrary &parallel_lib, const SharedVariablesData &svd, const ActiveSet &set, short output_level)
  constructor initializing base class for derived model class instances constructed on the fly

- **Model** (RecastBaseConstructor, ProblemDescDB &problem_db, ParallelLibrary &parallel_lib)
  constructor initializing base class for recast model class instances constructed on the fly

- **virtual void derived_compute_response** (const ActiveSet &set)
  portion of compute_response() specific to derived model classes

- **virtual void derived_asynch_compute_response** (const ActiveSet &set)
  portion of asynch_compute_response() specific to derived model classes

- **virtual const IntResponseMap & derived_synchronize** ()
  portion of synchronize() specific to derived model classes

- **virtual const IntResponseMap & derived_synchronize_nowait** ()
  portion of synchronize_nowait() specific to derived model classes

- **virtual void derived_init_communicators** (int max_eval_concurrency, bool recurse_flag=true)
  portion of init_communicators() specific to derived model classes

- **virtual void derived_init_serial** ()
  portion of init_serial() specific to derived model classes

- **virtual void derived_set_communicators** (int max_eval_concurrency, bool recurse_flag=true)
  portion of set_communicators() specific to derived model classes

- **virtual void derived_free_communicators** (int max_eval_concurrency, bool recurse_flag=true)
  portion of free_communicators() specific to derived model classes

Protected Attributes

- **Variables currentVariables**
  the set of current variables used by the model for performing function evaluations

- **size_t numDerivVars**
  the number of active continuous variables used in computing most response derivatives (i.e., in places such as quasi-Hessians and response corrections where only the active continuous variables are supported)

- **Response currentResponse**
  the set of current responses that holds the results of model function evaluations

- **size_t numFns**
  the number of functions in currentResponse

- **Constraints userDefinedConstraints**
  Explicit constraints on variables are maintained in the Constraints class hierarchy. Currently, this includes linear constraints and bounds, but could be extended in the future to include other explicit constraints which (1) have their form specified by the user, and (2) are not catalogued in Response since their form and coefficients are published to an iterator at startup.

- **String modelType**
  type of model: single, nested, or surrogate
• String surrogateType
  type of surrogate model: local\_*, multipoint\_*, global\_*, or hierarchical

• String 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

• RealVector fdGradStepSize
  relative finite difference step size for numerical gradients

• String fdGradStepType
  type of finite difference step to use for numerical gradient: relative - step length is relative to x absolute - step length is what is specified bounds - step length is relative to range of x

• RealVector fdHessByGradStepSize
  relative finite difference step size for numerical Hessians estimated using first-order differences of gradients

• RealVector fdHessByFnStepSize
  relative finite difference step size for numerical Hessians estimated using second-order differences of function values

• String fdHessStepType
  type of finite difference step to use for numerical Hessian: relative - step length is relative to x absolute - step length is what is specified bounds - step length is relative to range of x

• bool ignoreBounds
  option to ignore bounds when computing finite diffs

• bool centralHess
  option to use old 2nd-order finite diffs for Hessians

• bool supportsEstimDerivs
  whether model should perform or forward derivative estimation

• String quasiHessType
  quasi-Hessian type: bfgs, damped\_bfgs, sr1

• IntSet gradIdAnalytic
  analytic id’s for mixed gradients

• IntSet gradIdNumerical
  numerical id’s for mixed gradients

• IntSet hessIdAnalytic
  analytic id’s for mixed Hessians

• IntSet hessIdNumerical
  numerical id’s for mixed Hessians

• IntSet hessIdQuasi
  quasi id’s for mixed Hessians

• IntArray messageLengths
  length of packed MPI buffers containing vars, vars/set, response, and PRPair

• ProblemDescDB & probDescDB
class member reference to the problem description database

- ParallelLibrary & parallelLib
class member reference to the parallel library

- ParConfgClIter modelPCIter
the ParallelConfiguration node used by this model instance

- short componentParallelMode
the component parallelism mode: 0 (none), 1 (INTERFACE/LF_MODE), or 2 (SUB_MODE/HF_MODE/TRUTH_MODE)

- bool asynchEvalFlag
flags asynch evaluations (local or distributed)

- int evaluationCapacity
capacity for concurrent evaluations supported by the Model

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

- ModelList modelList
used to collect sub-models for subordinate_models()

- IntSetArray discreteDesignSetIntValues
array of IntSet's, each containing the set of allowable integer values corresponding to a discrete design integer set variable

- RealSetArray discreteDesignSetRealValues
array of RealSet's, each containing the set of allowable real values corresponding to a discrete design real set variable

- IntSetArray discreteStateSetIntValues
array of IntSet's, each containing the set of allowable integer values corresponding to a discrete state integer set variable

- RealSetArray discreteStateSetRealValues
array of RealSet's, each containing the set of allowable real values corresponding to a discrete state real set variable

- Pecos::AleatoryDistParams aleatDistParams
container for aleatory random variable distribution parameters

- Pecos::EpistemicDistParams epistDistParams
container for epistemic random variable distribution parameters

- BoolDeque primaryRespFnSense
array of flags (one per primary function) for switching the sense to maximize the primary function (default is minimize)

- RealVector primaryRespFnWts
primary response function weightings (either weights for multiobjective optimization or weighted least squares)

- bool hierarchicalTagging
whether to perform hierarchical evalID tagging of params/results
Private Member Functions

- **Model * get_model** (ProblemDescDB &problem_db)
  
  Used by the envelope to instantiate the correct letter class.

- **int estimate_derivatives** (const ShortArray &map_asv, const ShortArray &fd_grad_asv, const ShortArray &fd_hess_asv, const ShortArray &quasi_hess_asv, const ActiveSet &original_set, const bool asynch_flag)
  
  evaluate numerical gradients using finite differences. This routine is selected with "method_source dakota" (the default method_source) in the numerical gradient specification.

- **void synchronize_derivatives** (const Variables &vars, const IntResponseMap &fd_responses, Response &new_response, const ShortArray &fd_grad_asv, const ShortArray &fd_hess_asv, const ShortArray &quasi_hess_asv, const ActiveSet &original_set)
  
  combine results from an array of finite difference response objects (fd_grad_responses) into a single response (new_response)

- **void update_response** (const Variables &vars, Response &new_response, const ShortArray &fd_grad_asv, const ShortArray &fd_hess_asv, const ShortArray &quasi_hess_asv, const ActiveSet &original_set, Response &initial_map_response, const RealMatrix &new_fn_grads, const RealSymMatrixArray &new_fn_hessians)
  
  overlay results to update a response object

- **void update_quasi_hessians** (const Variables &vars, Response &new_response, const ActiveSet &original_set)
  
  perform quasi-Newton Hessian updates

- **Real finite_difference_lower_bound** (UShortMultiArrayConstView cv_types, const RealVector &global_c_l_bnds, size_t cv_index)
  
  return the lower bound for a finite difference offset, drawn from global or distribution bounds

- **Real finite_difference_upper_bound** (UShortMultiArrayConstView cv_types, const RealVector &global_c_u_bnds, size_t cv_index)
  
  return the upper bound for a finite difference offset, drawn from global or distribution bounds

- **bool manage_asv** (const ShortArray &asv_in, ShortArray &map_asv_out, ShortArray &fd_grad_asv_out, ShortArray &fd_hess_asv_out, ShortArray &quasi_hess_asv_out)
  
  Coordinates usage of estimate_derivatives() calls based on asv_in.

Private Attributes

- **String modelId**
  
  model identifier string from the input file

- **int modelEvalCntr**
  
  evaluation counter for top-level compute_response() and asynch_compute_response() calls. Differs from lower level counters in case of numerical derivative estimation (several lower level evaluations are assimilated into a single higher level evaluation)

- **bool estDerivsFlag**
  
  flags presence of estimated derivatives within a set of calls to asynch_compute_response()

- **std::map< int, ParConfigLIter > modelPCIterMap**
  
  map<> used for tracking modelPCIter instances using concurrency level as the lookup key

- **bool initCommsBcastFlag**
  
  flag for determining need to bcast the max concurrency from init_communicators(); set from Strategy::init_iterator()

- **bool modelAutoGraphicsFlag**
flag for posting of graphics data within compute_response (automatic graphics posting in the model as opposed to graphics posting at the strategy level)

- VariablesList varsList
  - history of vars populated in asynchronous compute_response() and used in synchronize().
- std::list< ShortArray > asvList
  - if estimate_derivatives() is used, transfers ASVs from asynchronous compute_response() to synchronize()
- std::list< ActiveSet > setList
  - if estimate_derivatives() is used, transfers ActiveSets from asynchronous compute_response() to synchronize()
- BoolList initialMapList
  - transfers initial_map flag values from estimate_derivatives() to synchronize_derivatives()
- BoolList dbCaptureList
  - transfers db_capture flag values from estimate_derivatives() to synchronize_derivatives()
- ResponseList dbResponseList
  - transfers database captures from estimate_derivatives() to synchronize_derivatives()
- RealList deltaList
  - transfers deltas from estimate_derivatives() to synchronize_derivatives()
- IntIntMap numFDEvalsMap
  - tracks the number of evaluations used within estimate_derivatives(). Used in synchronize() as a key for combining finite difference responses into numerical gradients.
- IntIntMap rawEvalIdMap
  - maps from the raw evaluation ids returned by derived_synchronize() and derived_synchronize_nowait() to the corresponding modelEvalCntr id. Used for rekeying responseMap.
- RealVectorArray xPrev
  - previous parameter vectors used in computing s for quasi-Newton updates
- RealMatrix fnGradsPrev
  - previous gradient vectors used in computing y for quasi-Newton updates
- RealSymMatrixArray quasiHessians
  - quasi-Newton Hessian approximations
- SizetArray numQuasiUpdates
  - number of quasi-Newton Hessian updates applied
- IntResponseMap responseMap
  - used to return a map of responses for asynchronous evaluations in final concatenated form. The similar map in Interface contains raw responses.
- IntResponseMap graphicsRespMap
  - used to cache the data returned from derived_synchronize_nowait() prior to sequential input into the graphics
- IntSetArray activeDiscSetIntValues
  - aggregation of the admissible value sets for all active discrete set integer variables
- RealSetArray activeDiscSetRealValues
  - aggregation of the admissible value sets for all active discrete set real variables
- BitArray discreteIntSets
  - key for identifying discrete integer set variables within the active discrete integer variables
- Model * modelRep
  - pointer to the letter (initialized only for the envelope)
- int referenceCount
  - number of objects sharing modelRep
13.62. MODEL CLASS REFERENCE

Friends

- bool operator==(const Model &m1, const Model &m2)
  
  equality operator (detect same letter instance)

- bool operator!=(const Model &m1, const Model &m2)
  
  inequality operator (detect different letter instances)

13.62.1 Detailed Description

Base class for the model class hierarchy.

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

13.62.2 Constructor & Destructor Documentation

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.

Model ( ProblemDescDB & problem_db )

standard constructor for envelope

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

References Dakota::abort_handler(), Model::get_model(), and Model::modelRep.

Model ( const Model & model )

copy constructor

Copy constructor manages sharing of modelRep and incrementing of referenceCount.

References Model::modelRep, and Model::referenceCount.

~Model ( ) [virtual]

destructor

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

References Model::modelRep, and Model::referenceCount.
CHAPTER 13. CLASS DOCUMENTATION

Model (BaseConstructor, ProblemDescDB & problem_db) [protected]

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

This constructor builds the base class data for all inherited models. get_model() instantiates a derived class and the derived class selects this base class constructor in its initialization list (to avoid the recursion of the base class constructor calling get_model() again). Since the letter IS the representation, its representation pointer is set to NULL (an uninitialized pointer causes problems in ~Model).

References Dakota::abort_handler(), Model::currentResponse, Model::fdGradStepSize, Model::fdHessByGradStepSize, Model::fdHessByGradStepSize, ProblemDescDB::get_sa(), Model::gradIdNumerical, Model::gradientType, Model::hessianType, Model::hessIdNumerical, Response::num_functions(), Constraints::num_nonlinear_eq_constraints(), Constraints::num_nonlinear_ineq_constraints(), Model::primaryRespFnSense, Dakota::strbegins(), Dakota::strtolower(), and Model::userDefinedConstraints.

Model (RecastBaseConstructor, ProblemDescDB & problem_db, ParallelLibrary & parallel_lib) [protected]

constructor initializing base class for recast model class instances constructed on the fly

This constructor also builds the base class data for inherited models. However, it is used for recast models which are instantiated on the fly. Therefore it only initializes a small subset of attributes. Note that parallel_lib is managed separately from problem_db since parallel_lib is needed even in cases where problem_db is an empty envelope.

13.62.3 Member Function Documentation

Model operator= (const Model & model)

assignment operator


References Model::modelRep, and Model::referenceCount.

Iterator & subordinate_iterator () [virtual]

return the sub-iterator in nested and surrogate models

return by reference requires use of dummy objects, but is important to allow use of assign_rep() since this operation must be performed on the original envelope object.

Reimplemented in RecastModel, DataFitSurrModel, and NestedModel.

References Dakota::dummy_iterator, Model::modelRep, and Model::subordinate_iterator().

Referenced by NonDExpansion::compute_expansion(), NonDExpansion::compute_print_converged_results(), NonDExpansion::compute_print_iteration_results(), NonDExpansion::finalize_sets(), NonDGlobalReliability::get_best_sample(), NonDPolynomialChaos::increment_order(), NonDExpansion::increment_sets(), NonDPolynomialChaos::increment_specification_sequence(), NonDExpansion::increment_specification_sequence(), DOTOptimizer::initialize(), CONMINOptimizer::initialize(), NCSUOptimizer::initialize(), NLPQLPOptimizer::initialize(), NonDExpansion::initialize_expansion(), NonDExpansion::initialize_sets(), NonDStochCollocation::initialize_u_space_model(), NonDExpansion::initialize_u_space_model(), SurrBasedLocalMinimizer::minimize_surrogates(), SurrBasedGlobalMinimizer::minimize_surrogates(), NonDLocalInterval::NonDLocalInterval(), NonDLocalReliability::NonDLocalReliability(), NonDGlobalReliability::optimize_gaussian_process(), NonDExpansion::refine_expansion(), SOLBase::SOLBase(), Model::subordinate_iterator(), RecastModel::subordinate_iterator(), and NonDStochCollocation::update_expansion().
Model & subordinate_model ( ) [virtual]

return a single sub-model defined from subModel in nested and recast models and truth_model() in surrogate models; used for a directed dive through model recursions that may bypass some components.

return by 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 RecastModel, NestedModel, and SurrogateModel.

References Dakota::dummy_model, Model::modelRep, and Model::subordinate_model().

Reference by Minimizer: data_transform_model(), NonDGloabalReliability: expected_feasibility(), NonDGloabalReliability: expected_improvement(), SurrogateModel: force_rebuild(), NonDExpansion: initialize_expansion(), Minimizer:: initialize_run(), Minimizer:: initialize_scaling(), NonDExpansion: initialize_u_space_model(), NonDGloabalReliability:: optimize_gaussian_process(), LeastSq:: post_run(), COLINOptimizer:: post_run(), Optimizer:: primary_resp_reducer(), LeastSq:: primary_resp_weighter(), LeastSq:: print_results(), Optimizer:: print_results(), Minimizer:: resize_best_resp_array(), Minimizer:: resize_best_vars_array(), Minimizer:: scale_model(), Model:: subordinate_model(), DataFitSurrModel:: update_global(), and LeastSq:: weight_model().

Model & surrogate_model ( ) [virtual]

return the approximation sub-model in surrogate models

return by reference requires use of dummy objects, but is important to allow use of assign_rep() since this operation must be performed on the original envelope object.

Reimplemented in RecastModel, DataFitSurrModel, and HierarchSurrModel.

References Dakota::dummy_model, Model:: modelRep, and Model::surrogate_model().

Referenced by NonDAadaptiveSampling:: calc_score_delta_y(), NonDAadaptiveSampling:: calc_score_topo_alm_hybrid(), NonDAadaptiveSampling:: calc_score_topo_avg_persistence(), NonDAadaptiveSampling:: calc_score_topo_bottleneck(), SurrBasedLocalMinimizer:: find_center_approx(), SurrBasedLocalMinimizer:: minimize_surrogates(), SurrBasedGlobalMinimizer:: minimize_surrogates(), NonDAadaptiveSampling:: output_round_data(), SurrBasedLocalMinimizer:: SurrBasedLocalMinimizer(), Model:: surrogate_model(), and RecastModel:: surrogate_model().

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 RecastModel, DataFitSurrModel, and HierarchSurrModel.

References Dakota:: dummy_model, Model:: modelRep, and Model:: truth_model().

Referenced by SurrogateModel:: force_rebuild(), SurrBasedMinimizer:: initialize_graphics(), SurrBasedLocalMinimizer:: minimize_surrogates(), SurrBasedGlobalMinimizer:: minimize_surrogates(), NonDLocalReliability:: NonDLocalReliability(), SurrBasedMinimizer::print_results(), SurrogateModel:: subordinate_model(), SurrBasedGlobalMinimizer:: SurrBasedGlobalMinimizer(), SurrBasedLocalMinimizer:: SurrBasedLocalMinimizer(), Model:: truth_model(), and RecastModel:: truth_model().

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 RecastModel, and DataFitSurrModel.
References Model::modelRep, and Model::update_from_subordinate_model().

Referenced by NonDLocalReliability::initialize_class_data(), NonDExpansion::initialize_expansion(), Least-Sq::initialize_run(), Optimizer::initialize_run(), EffGlobalMinimizer::minimize_surrogates_on_model(), NonDGlobalReliability::optimize_gaussian_process(), NonDGlobalInterval::quantify_uncertainty(), NonDLocalInterval::quantify_uncertainty(), Model::update_from_subordinate_model(), DataFitSurrModel::update_from_subordinate_model(), and RecastModel::update_from_subordinate_model().

**Interface & derived_interface ( ) [virtual]**

return the interface employed by the derived model class, if present: SingleModel::userDefinedInterface, DataFitSurrModel::approxInterface, or NestedModel::optionalInterface

return by reference requires use of dummy objects, but is important to allow use of assign_rep() since this operation must be performed on the original envelope object.

Reimplemented in RecastModel, DataFitSurrModel, NestedModel, and SingleModel.

References Model::derived_interface(), Dakota::dummy_interface, and Model::modelRep.

Referenced by Model::derived_interface(), RecastModel::derived_interface(), and SurrBasedGlobalMinimizer::minimize_surrogates().

**String local_eval_synchronization ( ) [virtual]**

return derived model synchronization setting

SingleModels and HierarchSurrModels redefine this virtual function. A default value of "synchronous" prevents async local operations for:

- NestedModels: a subiterator can support message passing parallelism, but not async local.
- DataFitSurrModels: while async evals on approximations will work due to some added bookkeeping, avoiding them is preferable.

Reimplemented in RecastModel, and SingleModel.

References Model::local_eval_synchronization(), and Model::modelRep.

Referenced by Model::init_serial(), Model::local_eval_synchronization(), RecastModel::local_eval_synchronization(), and Model::set_communicators().

**int local_eval_concurrency ( ) [virtual]**

return derived model asynchronous evaluation concurrency

SingleModels and HierarchSurrModels redefine this virtual function.

Reimplemented in RecastModel, and SingleModel.

References Model::local_eval_concurrency(), and Model::modelRep.

Referenced by Model::local_eval_concurrency(), RecastModel::local_eval_concurrency(), and Model::set_communicators().

**const String & interface_id ( ) const [virtual]**

return the interface identifier

return by reference requires use of dummy objects, but is important to allow use of assign_rep() since this operation must be performed on the original envelope object.

Reimplemented in RecastModel, DataFitSurrModel, NestedModel, and SingleModel.

References Dakota::dummy_interface, Interface::interface_id(), Model::interface_id(), and Model::modelRep.
Referenced by DataFitSurrModel::build_global(), DataFitSurrModel::DataFitSurrModel(), Model::estimate_derivatives(), Model::estimate_message_lengths(), SurrBasedLocalMinimizer::find_center_approx(), Model::interface_id(), RecastModel::interface_id(), Optimizer::local_objective_recast_retrieve(), SNLLastSq::post_run(), SurrBasedMinimizer::print_results(), LeastSq::print_results(), Optimizer::print_results(), SeqHybridMetaIterator::run_sequential(), DiscrepancyCorrection::search_db(), Analyzer::update_best(), ConcurrentMetaIterator::update_local_results(), SeqHybridMetaIterator::update_local_results(), and NonDLocalReliability::update_mpp_search_data().

**bool evaluation_cache ( ) const [virtual]**

Indicates the usage of an evaluation cache by the Model.

Only Models including ApplicationInterfaces support an evaluation cache: surrogate, nested, and recast mappings are not stored in the cache. Possible exceptions: HierarchSurrModel, NestedModel::optionalInterface.

Reimplemented in SingleModel.

References Model::evaluation_cache(), and Model::modelRep.

Referenced by DataFitSurrModel::DataFitSurrModel(), and Model::evaluation_cache().

**void eval_tag_prefix ( const String & eval_id_str ) [virtual]**

set the hierarchical eval ID tag prefix

Derived classes containing additional models or interfaces should implement this function to pass along to their sub Models/Interfaces.

Reimplemented in RecastModel, DataFitSurrModel, HierarchSurrModel, NestedModel, and SingleModel.

References Model::eval_tag_prefix(), and Model::modelRep.

Referenced by HierarchSurrModel::build_approximation(), HierarchSurrModel::derived_asynch_compute_response(), DataFitSurrModel::derived_asynch_compute_response(), HierarchSurrModel::derived_compute_response(), DataFitSurrModel::derived_compute_response(), Model::eval_tag_prefix(), Iterator::eval_tag_prefix(), and RecastModel::eval_tag_prefix().

**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_rep() since this operation must be performed on the original envelope object. They may, however, be used for letter-based operations (including assign_rep() on letter contents such as an interface).

References Model::derived_subordinate_models(), Model::modelList, Model::modelRep, and Model::subordinate_models().

Referenced by DOTOptimizer::initialize(), CONMINOptimizer::initialize(), NCSUOptimizer::initialize(), NLPLQLOptimizer::initialize(), NonDLocalInterval::NonDLocalInterval(), NonDLocalReliability::NonDLocalReliability(), SOLBase::SOLBase(), Model::subordinate_models(), and SurrBasedLocalMinimizer::SurrBasedLocalMinimizer().

**void init_communicators ( int max_eval_concurrency, bool recurse_flag = true )**

allocate communicator partitions for a model and store configuration in modelPCIterMap

The init_communicators() and derived_init_communicators() functions are structured to avoid performing the messageLengths estimation more than once. init_communicators() (not virtual) performs the estimation and then forwards the results to derived_init_communicators (virtual) which uses the data in different contexts.

References ParallelLibrary::bcast_i(), Model::derived_init_communicators(), Model::estimate_message_lengths(), ParallelLibrary::increment_parallel_configuration(), Model::init_communicators(), Model::initCommsBcastFlag, Model::messageLengths, Model::modelPCIter, Model::modelPCIterMap, Model::modelRep, ParallelLibrary::parallel_configuration_iterator(), Model::parallelLib, and ParallelLibrary::si_parallel_level_defined().
Referenced by HierarchSurrModel::derived_init_communicators(), RecastModel::derived_init_communicators(), NonDGPImpSampling::free_communicators(), EfficientSubspaceMethod::init_communicators(), EffGlobalMinimizer::init_communicators(), NonDGlobalReliability::init_communicators(), NonDLocalInterval::init_communicators(), NonDBayesCalibration::init_communicators(), NonDExpansion::init_communicators(), NonDGlobalInterval::init_communicators(), SurrBasedLocalMinimizer::init_communicators(), NonDAdaptImpSampling::init_communicators(), SurrBasedGlobalMinimizer::init_communicators(), NonDPolynomialChaos::init_communicators(), NonDAdaptiveSampling::init_communicators(), NonDGPImpSampling::init_communicators(), NonDLocalReliability::init_communicators(), Iterator::init_communicators(), Model::init_communicators(), EfficientSubspaceMethod::reduced_space_uq(), and Model::serve_configurations.

```cpp
void init_serial()
```

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

The init_serial() and derived_init_serial() functions are structured to separate base class (common) operations from derived class (specialized) operations.

References Model::asynchEvalFlag, Model::derived_init_serial(), Model::init_serial(), Model::local_eval_synchronization(), and Model::modelRep.

Referenced by NestedModel::derived_init_serial(), HierarchSurrModel::derived_init_serial(), DataFitSurrModel::derived_init_serial(), RecastModel::derived_init_serial(), and Model::init_serial().

```cpp
void estimate_message_lengths()
```

estimate messageLengths for a model

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

References Response::active_set_derivative_vector(), Response::copy(), Model::currentResponse, Model::currentVariables, Model::estimate_message_lengths(), Model::interface_id(), Model::messageLengths, Model::modelRep, Model::numFns, Model::parallelLib, MPIPackBuffer::reset(), MPIPackBuffer::size(), and ParallelLibrary::world_size().

Referenced by Model::estimate_message_lengths(), Model::init_communicators(), and ConcurrentMetaIterator::initialize().

```cpp
void assign_rep (Model * model_rep, bool ref_count_incr = true)
```

replaces existing letter with a new one

Similar to the assignment operator, the assign_rep() function decrements referenceCount for the old modelRep and assigns the new modelRep. It is different in that it is used for publishing derived class letters to existing envelopes, as opposed to sharing representations among multiple envelopes (in particular, assign_rep is passed a letter object and оператор= is passed an envelope object). Letter assignment supports two models as governed by ref_count_incr:

- ref_count_incr = true (default): the incoming letter belongs to another envelope. In this case, increment the reference count in the normal manner so that deallocation of the letter is handled properly.

- ref_count_incr = false: the incoming letter is instantiated on the fly and has no envelope. This case is modeled after get_model(): a letter is dynamically allocated using new and passed into assign_rep, the letter's reference count is not incremented, and the letter is not remotely deleted (its memory management is passed over to the envelope).
References Dakota::abort_handler(), Model::model_rep(), Model::modelRep, and Model::referenceCount.

Referenced by Minimizer::data_transform_model(), EffGlobalMinimizer::EffGlobalMinimizer(), NonDAdaptiveSampling::NonDAdaptiveSampling(), NonDBayesCalibration::NonDBayesCalibration(), NonDGlobalInterval::NonDGlobalInterval(), NonDGlobalReliability::NonDGlobalReliability(), NonDGPImpSampling::NonDGPImpSampling(), NonDLocalInterval::NonDLocalInterval(), NonDLocalReliability::NonDLocalReliability(), NonDPolynomialChaos::NonDPolynomialChaos(), NonDStochCollocation::NonDStochCollocation(), Optimizer::reduce_model(), EfficientSubspaceMethod::reduced_space_uq(), Minimizer::scale_model(), SurrBasedLocalMinimizer::SurrBasedLocalMinimizer(), NonD::transform_model(), and LeastSq::weight_model().

```cpp
int derivative_concurrency() const
```

return the gradient concurrency for use in parallel configuration logic

This function assumes derivatives with respect to the active continuous variables. Therefore, concurrency with respect to the inactive continuous variables is not captured.

References Dakota::contains(), Model::derivative_concurrency(), Model::gradIdAnalytic, Model::gradientType, Model::hessianType, Model::hessIdNumerical, Model::intervalType, Model::methodSource, Model::modelRep, and Model::numDerivVars.

Referenced by Model::derivative_concurrency(), HierarchSurrModel::derived_free_communicators(), HierarchSurrModel::derived_init_communicators(), DataFitSurrModel::derived_init_communicators(), HierarchSurrModel::derived_set_communicators(), NonDExpansion::initialize_u_space_model(), Analyzer::num_samples(), HierarchSurrModel::serve(), and Iterator::update_from_model().

```cpp
Real initialize_h( Real x_j, Real lb_j, Real ub_j, Real stepsize, String step_type )
```

function to determine initial finite difference h (before step length adjustment) based on type of step desired

Auxiliary function to determine initial finite difference h (before step length adjustment) based on type of step desired.

Referenced by Model::estimate_derivatives().

```cpp
Real FDstep1( Real x0_j, Real lb_j, Real ub_j, Real h_mag )
```

function returning finite-difference step size (affected by bounds)

Auxiliary function to compute forward or first central-difference step size.

References Model::ignoreBounds, and Model::shortStep.

Referenced by Model::estimate_derivatives().

```cpp
Real FDstep2( Real x0_j, Real lb_j, Real ub_j, Real h )
```

function returning second central-difference step size (affected by bounds)

Auxiliary function to second central-difference step size, honoring bounds.

References Model::ignoreBounds, and Model::shortStep.

Referenced by Model::estimate_derivatives().

```cpp
Model * get_model( ProblemDescDB & problem_db ) [private]
```

Used by the envelope to instantiate the correct letter class.

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

References ProblemDescDB::get_string(), Model::model_type(), and Model::modelType.

Referenced by Model::Model().
evaluate numerical gradients using finite differences. This routine is selected with "method_source_dakota" (the
default method_source) in the numerical gradient specification.

Estimate derivatives by computing finite difference gradients, finite difference Hessians, and/or quasi-Newton
Hessians. The total number of finite difference evaluations is returned for use by synchronize() to track response
arrays, and it could be used to improve management of max_function_evaluations within the iterators. ! new logic

References Model::all_continuous_lower_bounds(), Model::all_continuous_upper_bounds(), Model::all_continuous-
_variable_ids(), Model::all_continuous_variable_types(), Variables::all_continuous_variables(), Model::centralHess,
Model::central_lower_bounds(), Model::continuous_lower_bounds(), Variables::continuous_variable_ids(), Model::
continuous_variable_ids(), Model::continuous_variable_types(), Variables::continuous_variables(), Response::copy(),
Dakota::copy_data(), Model::currentResponse, Model::currentVariables, Dakota::data_pairs, Model::dbCapture-
List, Model::dbResponseList, Model::deltaList, ActiveSet::derivative_vector(), Model::derived_asynch_compute-
_response(), Model::derived_compute_response(), Model::fdGradStepSize, Model::fdGradStepType, Model::fd-
HessByFnStepSize, Model::fdHessByGradStepSize, Model::fdHessStepType, Model::FDstep1(), Model::FDstep2(),
Dakota::find_index(), Model::finite_difference_lower_bound(), Model::finite_difference_upper_bound(), Response-
::function_gradients(), Response::function_values(), Model::ignoreBounds, Model::inactive_continuous_lower-
_bounds(), Model::inactive_continuous_upper_bounds(), Variables::inactive_continuous_variable_ids(), Model::inactive-
_continuous_variable_ids(), Model::inactive_continuous_variable_types(), Variables::inactive_continuous_variables(),
Model::initialize_h(), Model::initialMapList, Model::interface_id(), Model::intervalType, Dakota::lookup_by_val(),
Model::numFns, Model::outputLevel, ActiveSet::request_vector(), Model::shortStep, and Model::update_response().

Referenced by Model::asynch_compute_response(), and Model::compute_response().

void synchronize_derivatives ( const Variables & vars, const IntResponseMap & fd_responses, Response
& new_response, const ShortArray & fd_grad_asv, const ShortArray & fd_hess_asv, const ShortArray & quasi_hess_asv, const ActiveSet & original_set ) [private]

combine results from an array of finite difference response objects (fd_responses) into a single response
(new_response)

Merge an array of fd_responses into a single new_response. This function is used both by synchronous
compute_response() for the case of asynchronous estimate_derivatives() and by synchronize() for the case where one or more asynch_compute_response() calls has employed asynchronous estimate_derivatives(). !

References Response::active_set(), Model::acv(), Variables::all_continuous_variable_ids(), Model::centralHess,
Variables::continuous_variable_ids(), Response::copy(), Model::currentResponse, Model::currentVariables, Model::
cv(), Model::dbCaptureList, Model::dbResponseList, Model::deltaList, ActiveSet::derivative_vector(), Dakota-
::find_index(), Response::function_gradients(), Response::function_values(), Model::icv(), Variables::inactive_continuous-
_variable_ids(), Model::initialMapList, Model::interface_id(), Model::intervalType, Model::numFns, ActiveSet::request_values(), Response-
::reset_inactive(), and Model::update_response().

Referenced by Model::compute_response(), and Model::synchronize().

void update_response ( const Variables & vars, Response & new_response, const ShortArray
& fd_grad_asv, const ShortArray & fd_hess_asv, const ShortArray & quasi_hess_asv, const ActiveSet & original_set, Response & initial_map_response, const RealMatrix & new_fn_grads, const RealSymMatrixArray & new_fn_hessians ) [private]

overlay results to update a response object

Overlay the initial_map_response with numerically estimated new_fn_grads and new_fn_hessians to populate
new_response as governed by asv vectors. Quasi-Newton secant Hessian updates are also performed here, since
this is where the gradient data needed for the updates is first consolidated. Convenience function used by estimate_derivatives() for the synchronous case and by synchronize_derivatives() for the asynchronous case.

References Response::active_set_request_vector(), Variables::continuous_variable_ids(), Response::copy(), Model::currentResponse, Model::currentVariables, ActiveSet::derivative_vector(), Response::function_gradients(), Response::function_hessians(), Response::function_values(), Model::hessianType, Model::hessIdQuasi, Response::is_null(), Model::numFnS, Model::outputLevel, Model::quasiHessians, ActiveSet::request_vector(), Response::reset_inactive(), Model::supportsEstimDerivs, Model::surrogate_response_mode(), and Model::update_quasi_hessians().

Referenced by Model::estimate_derivatives(), and Model::synchronize_derivatives().

```cpp
void update_quasi_hessians ( const Variables & vars, Response & new_response, const ActiveSet & original_set ) [private]
```
perform quasi-Newton Hessian updates

quasi-Newton updates are performed for approximating response function Hessians using BFGS or SR1 formulations. These Hessians are supported only for the active continuous variables, and a check is performed on the DVV prior to invoking the function.

References Dakota::contains(), Variables::continuous_variables(), Dakota::copy_data(), Model::fnGradsPrev, Response::function_gradients(), Model::hessianType, Model::hessIdQuasi, Model::modelType, Model::numDerivVars, Model::numFnS, Model::numQuasiUpdates, Model::outputLevel, Model::quasiHessians, Model::quasiHessType, ActiveSet::request_vector(), and Model::xPrev.

Referenced by Model::update_response().

```cpp
bool manage_asv ( const ShortArray & asv_in, ShortArray & map_asv_out, ShortArray & fd_grad_asv_out, ShortArray & fd_hess_asv_out, ShortArray & quasi_hess_asv_out ) [private]
```
Coordinates usage of estimate_derivatives() calls based on asv_in.

Splits asv_in total request into map_asv_out, fd_grad_asv_out, fd_hess_asv_out, and quasi_hess_asv_out as governed by the responses specification. If the returned use_est_deriv is true, then these asv outputs are used by estimate_derivatives() for the initial map, finite difference gradient evals, finite difference Hessian evals, and quasi-Hessian updates, respectively. If the returned use_est_deriv is false, then only map_asv_out is used.

References Dakota::abort_handler(), Dakota::contains(), Model::gradIdAnalytic, Model::gradIdNumerical, Model::gradientType, Model::hessianType, Model::hessIdAnalytic, Model::hessIdNumerical, Model::hessIdQuasi, Model::intervalType, Model::methodSource, Model::supportsEstimDerivs, and Model::surrogate_response_mode().

Referenced by Model::asynch_compute_response(), and Model::compute_response().

### 13.62.4 Member Data Documentation

**RealVector fdGradStepSize**  [protected]
relative finite difference step size for numerical gradients

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

Referenced by DataFitSurrModel::DataFitSurrModel(), Model::estimate_derivatives(), Model::fd_gradient_step_size(), RecastModel::initialize_data_from_submodel(), and Model::Model().

**RealVector fdHessByGradStepSize**  [protected]
relative finite difference step size for numerical Hessians estimated using first-order differences of gradients

For vendor numerical Hessian algorithms, a scalar value is used.

Referenced by DataFitSurrModel::DataFitSurrModel(), Model::estimate_derivatives(), Model::fd_hessian_by_grad_step_size(), RecastModel::initialize_data_from_submodel(), and Model::Model().
**RealVector fdHessByFnStepSize**  [protected]

relative finite difference step size for numerical Hessians estimated using second-order differences of function values.

For vendor numerical Hessian algorithms, a scalar value is used.

Referenced by DataFitSurrModel::DataFitSurrModel(), Model::estimate_derivatives(), Model::fd_hessian_by_fn_step_size(), RecastModel::initialize_data_from_submodel(), and Model::Model().

**ProblemDescDB& probDescDB**  [protected]

class member reference to the problem description database.

Iterator and Model cannot use a shallow copy of ProblemDescDB due to circular destruction dependency (reference counts can’t get to 0), since ProblemDescDB contains {iterator, model} List.

Referenced by Model::problem_description_db(), and DataFitSurrModel::update_from_subordinate_model().

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

- DakotaModel.hpp
- DakotaModel.cpp

### 13.63 MPIManager Class Reference

Class MPIManager to manage Dakota’s MPI world, which may be a subset of MPI_COMM_WORLD.

**Public Member Functions**

- **MPIManager ()**
  
  Default constructor; Dakota will not call MPI_Init.

- **MPIManager (int &argc, char **&argv)**
  
  Command-line constructor; parses MPI arguments during call to MPI_Init.

- **MPIManager (MPI_Comm dakota_mpi_comm)**
  
  Construct on specified MPI_Comm.

- **~MPIManager ()**
  
  Destructor: calls finalize if Dakota owns MPI.

- **MPI_Comm dakota_mpi_comm () const**
  
  Get the MPI_Comm on which Dakota is running.

- **int world_rank () const**
  
  Get the rank of this process in Dakota’s MPI_Comm.

- **int world_size () const**
  
  Get the size of the MPI_Comm on which Dakota is running.

- **bool mpirun_flag () const**
  
  True when Dakota is running in MPI mode.

**Static Public Member Functions**

- **static bool detect_parallel_launch (int &argc, char **&argv)**
  
  Detect parallel launch of Dakota using mpirun/mpiexec/poe/etc. based on command line arguments and environment variables.
13.64. MPIPACKBUFFER CLASS REFERENCE

Private Attributes

- **MPI_Comm dakotaMPIComm**
  
  *MPI_Comm on which DAKOTA is running.*

- **int worldRank**
  
  *rank in MPI_Comm in which DAKOTA is running*

- **int worldSize**
  
  *size of MPI_Comm in which DAKOTA is running*

- **bool mpirunFlag**
  
  *flag for a parallel mpirun/yod launch*

- **bool ownMPIFlag**
  
  *flag for ownership of MPI_Init/MPI_Finalize*

13.63.1 Detailed Description

Class **MPIManager** to manage Dakota’s MPI world, which may be a subset of MPI_COMM_WORLD.

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

- MPIManager.hpp
- MPIManager.cpp

13.64 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 \texttt{pack} (const \texttt{u_long *}data, const int num=1)  
  \textit{Pack one or more unsigned long's.}
• void \texttt{pack} (const \texttt{short *}data, const int num=1)  
  \textit{Pack one or more short's.}
• void \texttt{pack} (const \texttt{u_short *}data, const int num=1)  
  \textit{Pack one or more unsigned short's.}
• void \texttt{pack} (const \texttt{char *}data, const int num=1)  
  \textit{Pack one or more char's.}
• void \texttt{pack} (const \texttt{u_char *}data, const int num=1)  
  \textit{Pack one or more unsigned char's.}
• void \texttt{pack} (const \texttt{double *}data, const int num=1)  
  \textit{Pack one or more double's.}
• void \texttt{pack} (const \texttt{float *}data, const int num=1)  
  \textit{Pack one or more float's.}
• void \texttt{pack} (const \texttt{bool *}data, const int num=1)  
  \textit{Pack one or more bool's.}
• void \texttt{pack} (const int &data)  
  \textit{Pack a int.}
• void \texttt{pack} (const \texttt{u_int} &data)  
  \textit{Pack a unsigned int.}
• void \texttt{pack} (const long &data)  
  \textit{Pack a long.}
• void \texttt{pack} (const \texttt{u_long} &data)  
  \textit{Pack a unsigned long.}
• void \texttt{pack} (const short &data)  
  \textit{Pack a short.}
• void \texttt{pack} (const \texttt{u_short} &data)  
  \textit{Pack a unsigned short.}
• void \texttt{pack} (const char &data)  
  \textit{Pack a char.}
• void \texttt{pack} (const \texttt{u_char} &data)  
  \textit{Pack a unsigned char.}
• void \texttt{pack} (const double &data)  
  \textit{Pack a double.}
• void \texttt{pack} (const float &data)  
  \textit{Pack a float.}
• void \texttt{pack} (const bool &data)  
  \textit{Pack a bool.}

\textbf{Protected Member Functions}
• void \texttt{resize} (const int newsize)  
  \textit{Resizes the internal buffer.}
13.65. **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 ()**

---

### Protected Attributes

- **char * Buffer**
  
  *The internal buffer for packing.*

- **int Index**
  
  *The index into the current buffer.*

- **int Size**
  
  *The total size that has been allocated for the buffer.*

---

13.64.1 **Detailed Description**

Class for packing MPI message buffers.

A class that provides a facility for packing message buffers using the MPIPack facility. The **MPIPackBuffer** class dynamically resizes the internal buffer to contain enough memory to pack the entire object. When deleted, the **MPIPackBuffer** object deletes this internal buffer. This class is based on the Dakota Version 3.0 version of utilib::PackBuffer from utilib/src/io/PackBuf.[cpp,h]

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

- **MPIPackBuffer.hpp**
- **MPIPackBuffer.cpp**

---

13.64 **MPIUnpackBuffer Class Reference**

Class for unpacking MPI message buffers.

### Public Member Functions

- **void setup (char *buf, int size, bool flag=false)**
  
  *Method that does the setup for the constructors.*

- **MPIUnpackBuffer ()**
  
  *Default constructor.*

- **MPIUnpackBuffer (int size_)**
  
  *Constructor that specifies the size of the buffer.*

- **MPIUnpackBuffer (char *buf, int size, bool flag=false)**
  
  *Constructor that sets the internal buffer to the given array.*

- **~MPIUnpackBuffer ()**
  
  *Destructor.*

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

- **const char * buf ()**
  
  *Returns a pointer to the internal buffer.*

- **int size ()**
  
  *Returns the length of the buffer.*

- **int curr ()**
  
  *Returns the number of bytes that have been unpacked from the buffer.*

- **void reset ()**
Resets the index of the internal buffer.

- **void unpack (int *data, const int num=1)**
  Unpack one or more int's.
- **void unpack (u_int *data, const int num=1)**
  Unpack one or more unsigned int's.
- **void unpack (long *data, const int num=1)**
  Unpack one or more long's.
- **void unpack (u_long *data, const int num=1)**
  Unpack one or more unsigned long's.
- **void unpack (short *data, const int num=1)**
  Unpack one or more short's.
- **void unpack (u_short *data, const int num=1)**
  Unpack one or more unsigned short's.
- **void unpack (char *data, const int num=1)**
  Unpack one or more char's.
- **void unpack (u_char *data, const int num=1)**
  Unpack one or more unsigned char's.
- **void unpack (double *data, const int num=1)**
  Unpack one or more double's.
- **void unpack (float *data, const int num=1)**
  Unpack one or more float's.
- **void unpack (bool *data, const int num=1)**
  Unpack one or more bool's.
- **void unpack (int &data)**
  Unpack a int.
- **void unpack (u_int &data)**
  Unpack a unsigned int.
- **void unpack (long &data)**
  Unpack a long.
- **void unpack (u_long &data)**
  Unpack a unsigned long.
- **void unpack (short &data)**
  Unpack a short.
- **void unpack (u_short &data)**
  Unpack a unsigned short.
- **void unpack (char &data)**
  Unpack a char.
- **void unpack (u_char &data)**
  Unpack a unsigned char.
- **void unpack (double &data)**
  Unpack a double.
- **void unpack (float &data)**
  Unpack a float.
- **void unpack (bool &data)**
  Unpack a bool.
Protected Attributes

- char * Buffer
  The internal buffer for unpacking.
- int Index
  The index into the current buffer.
- int Size
  The total size that has been allocated for the buffer.
- bool ownFlag
  If TRUE, then this class owns the internal buffer.

13.65.1 Detailed Description

Class for unpacking MPI message buffers.

A class that provides a facility for unpacking message buffers using the MPI_Unpack facility. This class is based on the Dakota_Version_3.0 version of utilib::UnPackBuffer from utilib/src/io/PackBuf.[cpp,h]

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

- MPIPackBuffer.hpp
- MPIPackBuffer.cpp

13.66 NCSUOptimizer Class Reference

Wrapper class for the NCSU DIRECT optimization library.

Inheritance diagram for NCSUOptimizer:

```
        _______
       /      \
  Iterator /  Minimizer
       /    \
  Optimizer
       /    \
NCSUOptimizer
```

Public Member Functions

- NCSUOptimizer (ProblemDescDB &problem_db, 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 (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 ()**

  Used within the optimizer branch for computing the optimal solution. 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 idata[ ], int *iisize, double ddata[ ], int *idsize, char cdata[ ], int *icsize)**

  ‘fep’ in Griffin-modified NCSUDirect: computes the value of the objective function (potentially at multiple points, passed by function pointer to NCSUDirect). Include unscaling from DIRECT.

**Private Attributes**

• **short setUpType**

  controls iteration mode: SETUP_MODEL (normal usage) or SETUP_USERFUNC (user-supplied functions mode for "on the fly" instantiations). see enum in NCSUOptimizer.cpp NonDGlobalReliability currently uses the model mode. GaussProcApproximation currently uses the user functions mode.

• **Real minBoxSize**

  holds the minimum boxsize

• **Real volBoxSize**

  hold the minimum volume boxsize

• **Real solutionTarget**

  holds the solution target minimum to drive towards

• **RealVector lowerBounds**

  holds variable lower bounds passed in for "user functions" mode.

• **RealVector upperBounds**

  holds variable upper bounds passed in for "user functions" mode.

• **double(* userObjectiveEval ) (const RealVector &x)**

  holds function pointer for objective function evaluator passed in for "user functions" mode.
Static Private Attributes

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

Additional Inherited Members

13.66.1 Detailed Description

Wrapper class for the NCSU DIRECT optimization library.

The NCSUOptimizer class provides a wrapper for a Fortran 77 implementation of the DIRECT algorithm developed at North Carolina State University. It uses a function pointer approach for which passed functions must be either global functions or static member functions. Any attribute used within static member functions must be either local to that function or accessed through a static pointer.

The user input mappings are as follows:

13.66.2 Constructor & Destructor Documentation

NCSUOptimizer ( ProblemDescDB & problem_db, Model & model )

standard constructor

This is the standard constructor with method specification support.

References NCSUOptimizer::check_inputs(), and NCSUOptimizer::initialize().

NCSUOptimizer ( Model & model, const int & max_iter, const int & max_eval, double min_box_size = -1., double vol_box_size = -1., double solution_target = -DBL_MAX )

alternate constructor for instantiations "on the fly"

This is an alternate constructor for instantiations on the fly using a Model but no ProblemDescDB.

References NCSUOptimizer::check_inputs(), NCSUOptimizer::initialize(), Iterator::maxFunctionEvals, and Iterator::maxIterations.

NCSUOptimizer ( Model & model )

alternate constructor for Iterator instantiations by name

This is an alternate constructor for Iterator instantiations by name using a Model but no ProblemDescDB.

References NCSUOptimizer::check_inputs(), and NCSUOptimizer::initialize().

NCSUOptimizer ( const RealVector & var_l_bnds, const RealVector & var_u_bnds, const int & max_iter, const int & max_eval, double(*)(const RealVector &x) user_obj_eval, double min_box_size = -1., double vol_box_size = -1., double solution_target = -DBL_MAX )

alternate constructor for instantiations "on the fly"

This is an alternate constructor for performing an optimization using the passed in objective function pointer.

References NCSUOptimizer::check_inputs(), Iterator::maxFunctionEvals, and Iterator::maxIterations.
13.66.3 Member Function Documentation

```cpp
int objective_eval ( int * n, double * c[ ], double * l[ ], double * u[ ], int * point[ ], int * maxI, int * start, int * maxfunc, double * fvec[ ], int * isize, double * adata[ ], int * isize, char * cdata[ ], int * isize ) [static], [private]

'fep' in Griffin-modified NCSUDirect: computes the value of the objective function (potentially at multiple points, passed by function pointer to NCSUDirect). Include unscaling from DIRECT.

Modified batch evaluator that accepts multiple points and returns corresponding vector of functions in fvec. Must be used with modified DIRECT src (DIRbatch.f).

References Model::asynch compute response(), Model::asynch flag(), Model::compute response(), Model::continuous variables(), Model::current response(), Response::function value(), Iterator::iteratedModel, NCSU-Optimizer::ncsudirectInstance, Model::primary response fn sense(), NCSUOptimizer::setUpType, Model::synchronize(), and NCSUOptimizer::userObjectiveEval.

Referenced by NCSUOptimizer::find optimum().

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

- NCSUOptimizer.hpp
- NCSUOptimizer.cpp
```

13.67 NestedModel Class Reference

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

Inheritance diagram for NestedModel:

```
  Model
     \   /
      \ /     NestedModel
```

Public Member Functions

- **NestedModel** (ProblemDescDB &problem_db)
  
  constructor

- ```cpp
    ~NestedModel()
  ```
  
  destructor

Protected Member Functions

- ```cpp
    void derived compute response (const ActiveSet &set)
  ```

  portion of compute response() specific to NestedModel

- ```cpp
    void derived asynch compute response (const ActiveSet &set)
  ```

  portion of asynch compute response() specific to NestedModel

- ```cpp
    Iterator & subordinate iterator ()
  ```

  return subIterator

- ```cpp
    Model & subordinate model ()
  ```

  return subModel
void derived_subordinate_models (ModelList &ml, bool recurse_flag)
    return subModel

Interface & derived_interface ()
    return optionalInterface

void surrogate_response_mode (short mode)
    pass a bypass request on to the subModel for any lower-level surrogates

void component_parallel_mode (short mode)
    update component parallel mode for supporting parallelism in optionalInterface and subModel

bool derived_master_overload () const
    flag which prevents overloading the master with a multiprocessor evaluation (forwarded to optionalInterface)

void derived_init_communicators (int max_eval_concurrency, bool recurse_flag=true)
    set up optionalInterface and subModel for parallel operations

void derived_init_serial ()
    set up optionalInterface and subModel for serial operations.

void derived_set_communicators (int max_eval_concurrency, bool recurse_flag=true)
    set active parallel configuration within subModel

void derived_free_communicators (int max_eval_concurrency, bool recurse_flag=true)
    deallocate communicator partitions for the NestedModel (forwarded to optionalInterface and subModel)

void serve (int max_eval_concurrency)
    Service optionalInterface and subModel job requests received from the master. Completes when a termination
    message is received from stop_servers().

void stop_servers ()
    Executed by the master to terminate server operations for subModel and optionalInterface when iteration on the
    NestedModel is complete.

const String & interface_id () const
    return the optionalInterface identifier

int evaluation_id () const
    Return the current evaluation id for the NestedModel.

void set_evaluation_reference ()
    set the evaluation counter reference points for the NestedModel (request forwarded to optionalInterface and sub-
    Model)

void fine_grained_evaluation_counters ()
    request fine-grained evaluation reporting within optionalInterface and subModel

void print_evaluation_summary (std::ostream &s, bool minimal_header=false, bool relative_count=true) const
    print the evaluation summary for the NestedModel (request forwarded to optionalInterface and subModel)

virtual void eval_tag_prefix (const String &eval_id_str)
    set the hierarchical eval ID tag prefix
Private Member Functions

- void resolve_real_variable_mapping (const String &map1, const String &map2, size_t curr_index, short &inactive_sm_view)
  for a named real mapping, resolve primary index and secondary target
- void resolve_integer_variable_mapping (const String &map1, const String &map2, size_t curr_index, short &inactive_sm_view)
  for a named integer mapping, resolve primary index and secondary target
- size_t sm_acv_index_map (size_t pacvm_index, short sacvm_target)
  offset pacvm_index based on sacvm_target to create mapped_index
- size_t sm_adiv_index_map (size_t padvm_index, short sadivm_target)
  offset padvm_index based on sadivm_target to create mapped_index
- size_t sm_adrv_index_map (size_t padrvm_index, short sadrvm_target)
  offset padrvm_index based on sadrvm_target to create mapped_index
- size_t cv_index_map (size_t cv_index)
  offset cv_index to create index into aggregated primary/secondary arrays
- size_t div_index_map (size_t div_index)
  offset div_index to create index into aggregated primary/secondary arrays
- size_t drv_index_map (size_t drv_index)
  offset drv_index to create index into aggregated primary/secondary arrays
- size_t ccv_index_map (size_t ccv_index)
  offset active complement ccv_index to create index into all continuous arrays
- size_t cdiv_index_map (size_t cdiv_index)
  offset active complement cdiv_index to create index into all discrete int arrays
- size_t cdrv_index_map (size_t cdrv_index)
  offset active complement cdrv_index to create index into all discrete real arrays
- void real_variable_mapping (const Real &r_var, size_t mapped_index, short svm_target)
  insert r_var into appropriate recipient
- void integer_variable_mapping (const int &i_var, size_t mapped_index, short svm_target)
  insert i_var into appropriate recipient
- void set_mapping (const ActiveSet &mapped_set, ActiveSet &interface_set, bool &opt_interface_map, ActiveSet &sub_iterator_set, bool &sub_iterator_map)
  define the evaluation requirements for the optional interface (interface_set) and the subIterator (sub_iterator_set)
  from the total model evaluation requirements (mapped_set)
- void response_mapping (const Response &interface_response, const Response &sub_iterator_response, Response &mapped_response)
  combine the response from the optional interface evaluation with the response from the sub-iteration using the primaryCoeffs/secondaryCoeffs mappings to create the total response for the model
- void update_inactive_view (short new_view, short &view)
  update inactive variables view for subIterator based on new_view
- void update_inactive_view (unsigned short type, short &view)
  update inactive variables view for subIterator based on type
- void update_sub_model ()
  update subModel with current variable values/bounds/labels
Private Attributes

- int nestedModelEvalCntr
  number of calls to `derived_compute_response() / derived_asynch_compute_response()`
- Iterator subIterator
  the sub-iterator that is executed on every evaluation of this model
- Model subModel
  the sub-model used in sub-iterator evaluations
- size_t numSubIterFns
  number of sub-iterator response functions prior to mapping
- size_t numSubIterMappedIneqCon
  number of top-level inequality constraints mapped from the sub-iteration results
- size_t numSubIterMappedEqCon
  number of top-level equality constraints mapped from the sub-iteration results
- Interface optionalInterface
  the optional interface contributes nonnested response data to the total model response
- String optInterfacePointer
  the optional interface pointer from the nested model specification
- Response optInterfaceResponse
  the response object resulting from optional interface evaluations
- size_t numOptInterfPrimary
  number of primary response functions (objective/least squares/generic functions) resulting from optional interface evaluations
- size_t numOptInterfIneqCon
  number of inequality constraints resulting from optional interface evaluations
- size_t numOptInterfEqCon
  number of equality constraints resulting from the optional interface evaluations
- SizetArray active1ACVarMapIndices
  "primary" variable mappings for inserting active continuous `currentVariables` within all continuous `subModel` variables. If there are no secondary mappings defined, then the insertions replace the `subModel` variable values.
- SizetArray active1ADIVarMapIndices
  "primary" variable mappings for inserting active discrete int `currentVariables` within all discrete int `subModel` variables. No secondary mappings are defined for discrete int variables, so the insertions replace the `subModel` variable values.
- SizetArray active1ADRVarMapIndices
  "primary" variable mappings for inserting active discrete real `currentVariables` within all discrete real `subModel` variables. No secondary mappings are defined for discrete real variables, so the insertions replace the `subModel` variable values.
- ShortArray active2ACVarMapTargets
  "secondary" variable mappings for inserting active continuous `currentVariables` into sub-parameters (e.g., distribution parameters for uncertain variables or bounds for continuous design/state variables) within all continuous `subModel` variables.
- ShortArray active2ADIVarMapTargets
  "secondary" variable mappings for inserting active discrete int `currentVariables` into sub-parameters (e.g., bounds for discrete design/state variables) within all discrete int `subModel` variables.
• ShortArray active2ADRVarMapTargets
  "secondary" variable mappings for inserting active discrete real currentVariables into sub-parameters (e.g., bounds for discrete design/state variables) within all discrete real subModel variables.

• SizetArray complement1ACVarMapIndices
  "primary" variable mappings for inserting the complement of the active continuous currentVariables within all continuous subModel variables

• SizetArray complement1ADIVarMapIndices
  "primary" variable mappings for inserting the complement of the active discrete int currentVariables within all discrete int subModel variables

• SizetArray complement1ADRVarMapIndices
  "primary" variable mappings for inserting the complement of the active discrete real currentVariables within all discrete real subModel variables

• BoolDeque extraCVarsData
  flags for updating subModel continuous bounds and labels, one for each active continuous variable in currentVariables

• BoolDeque extraDIVarsData
  flags for updating subModel discrete int bounds and labels, one for each active discrete int variable in currentVariables

• BoolDeque extraDRVarsData
  flags for updating subModel discrete real bounds and labels, one for each active discrete real variable in currentVariables

• RealMatrix primaryRespCoeffs
  "primary" response mapping matrix applied to the sub-iterator response functions. For OUU, the matrix is applied to UQ statistics to create contributions to the top-level objective functions/least squares/generic response terms.

• RealMatrix secondaryRespCoeffs
  "secondary" response mapping matrix applied to the sub-iterator response functions. For OUU, the matrix is applied to UQ statistics to create contributions to the top-level inequality and equality constraints.

• String evalTagPrefix
  cached evalTag Prefix from parents to use at compute_response time

Additional Inherited Members

13.67.1 Detailed Description

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

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

13.67.2 Member Function Documentation

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

portion of compute_response() specific to NestedModel

Update subModel's inactive variables with active variables from currentVariables, compute the optional interface and sub-iterator responses, and map these to the total model response.
Reimplemented from Model.
References Response::active_set(), NestedModel::component_parallel_mode(), Model::currentResponse, Model::currentVariables, Interface::eval_tag_prefix(), Iterator::eval_tag_prefix(), NestedModel::evalTagPrefix, Model::hierarchicalTagging, Interface::map(), NestedModel::nestedModelEvalCntr, NestedModel::optInterfaceResponse, NestedModel::optionalInterface, NestedModel::response_mapping(), Iterator::response_results(), Iterator::response_results_active_set(), Iterator::run(), NestedModel::set_mapping(), NestedModel::subIterator, and NestedModel::update_sub_model().

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

portion of asynch_compute_response() specific to NestedModel
Not currently supported by NestedModels (need to add concurrent iterator support). As a result, derived_synchronize() and derived_synchronize_nowait() are inactive as well).
Reimplemented from Model.
References Dakota::abort_handler(), Response::active_set(), Model::currentResponse, Model::currentVariables, Interface::map(), NestedModel::nestedModelEvalCntr, NestedModel::optInterfaceResponse, NestedModel::optionalInterface, and NestedModel::set_mapping().

bool derived_master_overload ( ) const [inline], [protected], [virtual]
flag which prevents overloading the master with a multiprocessor evaluation (forwarded to optionalInterface)
Derived master overload for subModel is handled separately in subModel.compute_response() within subIterator.run().
Reimplemented from Model.
References Interface::iterator_eval_dedicated_master(), Interface::multi_proc_eval(), NestedModel::optInterfacePointer, and NestedModel::optionalInterface.

void derived_init_communicators ( int max_eval_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_eval_concurrency is the outer level iterator concurrency, not the subIterator concurrency that subModel will see, and recomputing the message_lengths on the subModel is probably not a bad idea either. Therefore, recompute everything on subModel using init_communicators().
Reimplemented from Model.
References Interface::init_communicators(), Iterator::init_communicators(), Model::messageLengths, NestedModel::optInterfacePointer, NestedModel::optionalInterface, and NestedModel::subIterator.

int evaluation_id ( ) const [inline], [protected], [virtual]
Return the current evaluation id for the NestedModel.
return the top level nested evaluation count. To get the lower level eval count, the subModel must be explicitly queried. This is consistent with the eval counter definitions in surrogate models.
Reimplemented from Model.
References NestedModel::nestedModelEvalCntr.
size_t cv_index_map ( size_t cv_index ) [private]

offset cv_index to create index into aggregated primary/secondary arrays
maps index within active continuous variables to index within aggregated active continuous/discrete-int/discrete-real variables.
References Model::currentVariables, Variables::variables_components_totals(), and Variables::view().
Referenced by NestedModel::NestedModel(), and NestedModel::update_sub_model().

size_t div_index_map ( size_t div_index ) [private]

offset div_index to create index into aggregated primary/secondary arrays
maps index within active discrete int variables to index within aggregated active continuous/discrete-int/discrete-real variables.
References Model::currentVariables, Variables::cv(), Variables::variables_components_totals(), and Variables::view().
Referenced by NestedModel::NestedModel(), and NestedModel::update_sub_model().

size_t drv_index_map ( size_t drv_index ) [private]

offset drv_index to create index into aggregated primary/secondary arrays
maps index within active discrete real variables to index within aggregated active continuous/discrete-int/discrete-real variables.
References Model::currentVariables, Variables::cv(), Variables::div(), Variables::variables_components_totals(), and Variables::view().
Referenced by NestedModel::NestedModel(), and NestedModel::update_sub_model().

size_t ccv_index_map ( size_t ccv_index ) [private]

offset active complement ccv_index to create index into all continuous arrays
maps index within complement of active continuous variables to index within all continuous variables.
References Dakota::abort_handler(), Model::currentVariables, Variables::variables_components_totals(), and Variables::view().
Referenced by NestedModel::NestedModel(), and NestedModel::update_sub_model().

size_t cdiv_index_map ( size_t cdiv_index ) [private]

offset active complement cdiv_index to create index into all discrete int arrays
maps index within complement of active discrete int variables to index within all discrete int variables.
References Dakota::abort_handler(), Model::currentVariables, Variables::variables_components_totals(), and Variables::view().
Referenced by NestedModel::NestedModel(), and NestedModel::update_sub_model().

size_t cdrv_index_map ( size_t cdrv_index ) [private]

offset active complement cdrv_index to create index into all discrete real arrays
maps index within complement of active discrete real variables to index within all discrete real variables.
References Dakota::abort_handler(), Model::currentVariables, Variables::variables_components_totals(), and Variables::view().
Referenced by NestedModel::NestedModel(), and NestedModel::update_sub_model().
void response_mapping ( const Response & opt_interface_response, const Response & sub_iterator_response, Response & mapped_response ) [private]

combine the response from the optional interface evaluation with the response from the sub-iteration using the primaryCoeffs/secondaryCoeffs mappings to create the total response for the model

In the OUU case,

optionalInterface fns = {f}, {g} (deterministic primary functions, constraints)
subIterator fns = {S} (UQ response statistics)

Problem formulation for mapped functions:
minimize {f} + [W]{S}
subject to {g_l} <= {g} <= {g_u}
{a_l} <= [A]{S} <= {a_u}
{g} == {g_t}
{A}{S} == {a_t}

where [W] is the primary_mapping_matrix user input (primaryRespCoeffs class attribute), [A] is the secondary_mapping_matrix user input (secondaryRespCoeffs class attribute), {{g_l},{a_l}} are the top level inequality constraint lower bounds, {{g_u},{a_u}} are the top level inequality constraint upper bounds, and {{g_t},{a_t}} are the top level equality constraint targets.

NOTE: optionalInterface/subIterator primary fns (obj/lsq/generic fns) overlap but optionalInterface/subIterator secondary fns (ineq/eq constraints) do not. The [W] matrix can be specified so as to allow

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

• some combined and some purely stochastic primary functions: [W] filled and [W].num_rows() > {f}.length()

• separate deterministic and stochastic primary functions: [W].num_rows() > {f}.length() and [W] contains {f}.length() rows of zeros.

If the need arises, could change constraint definition to allow overlap as well: {g_l} <= {g} + [A]{S} <= {g_u}

with [A] usage the same as for [W] above.

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

References Dakota::abort_handler(), Response::active_set_derivative_vector(), Response::active_set_request_derivative_vector(), Dakota::copy_data(), Response::function_gradient(), Response::function_gradient_view(), Response::function_gradients(), Response::function_hessian(), Response::function_hessian_view(), Response::function_hessians(), Response::function_value(), Response::function_values(), Response::function_values_view(), Response::num_functions(), NestedModel::numOptInterfEqCon, NestedModel::numOptInterfIneqCon, NestedModel::numOptInterfPrimary, NestedModel::numSubIterFns, NestedModel::numSubIterMappedEqCon, NestedModel::numSubIterMappedIneqCon, NestedModel::optInterfEqCon, NestedModel::optInterfIneqCon, NestedModel::optInterfacePointer, NestedModel::primaryRespCoeffs, Response::reset_inactive(), and NestedModel::secondaryRespCoeffs.

Referenced by NestedModel::derived_compute_response().

13.67.3 Member Data Documentation

Model subModel [private]

the sub-model used in sub-iterator evaluations

There are no restrictions on subModel, so arbitrary nestings are possible. This is commonly used to support surrogate-based optimization under uncertainty by having NestedModels contain SurrogateModels and vice versa.
Referenced by NestedModel::component_parallel_mode(), NestedModel::derived_init_serial(), NestedModel::derived_subordinate_models(), NestedModel::fine_grained_evaluation_counters(), NestedModel::integer_variable_mapping(), NestedModel::init(), NestedModel::print_evaluation_summary(), NestedModel::real_variable_mapping(), NestedModel::resolve_integer_variable_mapping(), NestedModel::resolve_real_variable_mapping(), NestedModel::serve(), NestedModel::set_mapping(), NestedModel::sm_acv_index_map(), NestedModel::sm_adiv_index_map(), NestedModel::subordinate_model(), NestedModel::surrogate_response_mode(), NestedModel::update_inactive_view(), and NestedModel::update_sub_model().

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

- NestedModel.hpp
- NestedModel.cpp

### 13.68 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 ProgramOptions &prog_opts)
  
  *parses the input file and populates the problem description database using NIDR.*

- **void derived_broadcast** ()
  
  *perform any data processing that must be coordinated with DB buffer broadcasting (performed prior to broadcasting the DB buffer on rank 0 and after receiving the DB buffer on other processor ranks)*

- **void derived_post_process** ()
  
  *perform any additional data post-processing*

- **KWH** (iface_Real)

- **KWH** (iface_Rlit)

- **KWH** (iface_false)

- **KWH** (iface_ilit)

- **KWH** (iface_pint)

- **KWH** (iface_lit)

- **KWH** (iface_start)

- **KWH** (iface_stop)

- **KWH** (iface_str)

- **KWH** (iface_str2D)
- KWH (iface_strL)
- KWH (iface_true)
- KWH (iface_type)
- KWH (method_Li)
- KWH (method_Real)
- KWH (method_Real01)
- KWH (method_RealDL)
- KWH (method_RealL_lit)
- KWH (method_Realp)
- KWH (method_Realz)
- KWH (method_Ri)
- KWH (method_false)
- KWH (method_szarray)
- KWH (method_ilit2)
- KWH (method_ilit2p)
- KWH (method_int)
- KWH (method_ivec)
- KWH (method_lit)
- KWH (method_litic)
- KWH (method_liti)
- KWH (method_litp)
- KWH (method_litr)
- KWH (method_litz)
- KWH (method_nuint)
- KWH (method_num_resplevs)
- KWH (method_piecewise)
- KWH (method_pint)
- KWH (method_pintz)
- KWH (method_resplevs)
- KWH (method_resplevs01)
- KWH (method_shint)
- KWH (method_sizet)
- KWH (method_slit2)
- KWH (method_start)
- KWH (method_stop)
- KWH (method_str)
- KWH (method_strL)
- KWH (method_true)
- KWH (method_tr_final)
- KWH (method_type)
- KWH (method_usharray)
- KWH (method_ushint)
- KWH (method_utype)
- KWH (method_utype_lit)
- KWH (model_Real)
- KWH (model_RealDL)
• \texttt{KWH (model\_false)}
• \texttt{KWH (model\_int)}
• \texttt{KWH (model\_intsetm1)}
• \texttt{KWH (model\_lit)}
• \texttt{KWH (model\_order)}
• \texttt{KWH (model\_pint)}
• \texttt{KWH (model\_shint)}
• \texttt{KWH (model\_start)}
• \texttt{KWH (model\_stop)}
• \texttt{KWH (model\_str)}
• \texttt{KWH (model\_strL)}
• \texttt{KWH (model\_true)}
• \texttt{KWH (model\_type)}
• \texttt{KWH (resp\_RealDL)}
• \texttt{KWH (resp\_RealL)}
• \texttt{KWH (resp\_false)}
• \texttt{KWH (resp\_intset)}
• \texttt{KWH (resp\_ivec)}
• \texttt{KWH (resp\_lit)}
• \texttt{KWH (resp\_sizet)}
• \texttt{KWH (resp\_start)}
• \texttt{KWH (resp\_stop)}
• \texttt{KWH (resp\_str)}
• \texttt{KWH (resp\_strL)}
• \texttt{KWH (resp\_true)}
• \texttt{KWH (env\_int)}
• \texttt{KWH (env\_start)}
• \texttt{KWH (env\_str)}
• \texttt{KWH (env\_true)}
• \texttt{KWH (var\_RealLb)}
• \texttt{KWH (var\_RealUb)}
• \texttt{KWH (var\_IntLb)}
• \texttt{KWH (var\_caulbl)}
• \texttt{KWH (var\_daulbl)}
• \texttt{KWH (var\_ceulbl)}
• \texttt{KWH (var\_deulbl)}
• \texttt{KWH (var\_deurlbl)}
• \texttt{KWH (var\_pintz)}
• \texttt{KWH (var\_start)}
• \texttt{KWH (var\_stop)}
• \texttt{KWH (var\_str)}
• \texttt{KWH (var\_strL)}
• \texttt{KWH (var\_true)}
• \texttt{KWH (var\_newiarray)}
• \texttt{KWH (var\_newivec)}
13.68. NIDRPROBLEMDESCDB CLASS REFERENCE

- **KWH (var_newrvec)**
- **KWH (var_ivec)**
- **KWH (var_rvec)**
- **KWH (var_type)**

### Static Public Member Functions

- static void `botch` (const char ∗fmt,...)
- static void `check_variables` (std::list< DataVariables > ∗)
- static void `check_responses` (std::list< DataResponses > ∗)
- static void `make_variable_defaults` (std::list< DataVariables > ∗)
- static void `make_response_defaults` (std::list< DataResponses > ∗)
- static void `squawk` (const char ∗fmt,...)
- static void `warn` (const char ∗fmt,...)

### Static Public Attributes

- static NIDRProblemDescDB ∗pDBInstance
  
  *pointer to the active object instance used within the static kwhandler functions in order to avoid the need for static data*
- static int nerr = 0

### Static Private Member Functions

- static void `check_variables_node` (void ∗)

### Private Attributes

- std::list< void ∗ > VIL

### Additional Inherited Members

13.68.1 Detailed Description

The derived input file database utilizing the new IDR parser.

The NIDRProblemDescDB class is derived from ProblemDescDB for use by the NIDR parser in processing DAKOTA input file data. For information on modifying the NIDR input parsing procedures, refer to Dakota/docs/Dev_Spec_Change.dox. For more on the parsing technology, see “Specifying and Reading Program Input with NIDR” by David M. Gay (report SAND2008-2261P, which is available in PDF form as http://www.sandia.gov/~dmgay/nidr08.pdf). Source for the routines declared herein is NIDRProblemDescDB.cpp, in which most routines are so short that a description seems unnecessary.

13.68.2 Member Function Documentation

```cpp
void derived_parse_inputs ( const ProgramOptions & prog_opts ) [virtual]
```

parses the input file and populates the problem description database using NIDR.

Parse the input file using the Input Deck Reader (IDR) parsing system. IDR populates the IDRProblemDescDB object with the input file data.

Reimplemented from ProblemDescDB.
References Dakota::abort_handler(), ProblemDescDB::dataInterfaceList, ProblemDescDB::dataMethodList, DataMethodRep::dlDetails, DataMethodRep::dlLib, ProgramOptions::input_file(), ProgramOptions::input_string(), ProblemDescDB::parallel_library(), ProgramOptions::parser_options(), and NIDRProblemDescDB::pDDBInstance. The documentation for this class was generated from the following files:

- NIDRProblemDescDB.hpp
- NIDRProblemDescDB.cpp

### 13.69 NL2Res Struct Reference

Auxiliary information passed to calcr and calcj via ur.

#### Public Attributes

- **Real * r**
  
  residual \( r = r(x) \)

- **Real * J**
  
  Jacobian \( J = J(x) \)

- **Real * x**
  
  corresponding parameter vector

- **int nf**
  
  function invocation count for \( r(x) \)

#### 13.69.1 Detailed Description

Auxiliary information passed to calcr and calcj via ur.

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

- NL2SOLLeastSq.cpp

### 13.70 NL2SOLLeastSq Class Reference

Wrapper class for the NL2SOL nonlinear least squares library.

Inheritance diagram for NL2SOLLeastSq:

```
  Iterator
  Minimizer
  LeastSq
  NL2SOLLeastSq
```
Public Member Functions

- **NL2SOLLeastSq (ProblemDescDB &problem_db, Model &model)**
  standard constructor
- **NL2SOLLeastSq (Model &model)**
  alternate constructor
- **~NL2SOLLeastSq ()**
  destructor
- **void minimize_residuals ()**
  Used within the least squares branch for minimizing the sum of squares residuals. Redefines the run virtual function for the least squares branch.

Static Private Member Functions

- **static void calcr (int *np, int *pp, Real *x, int *nfp, Real *r, int *ui, void *ur, Vf vf)**
  evaluator function for residual vector
- **static void calcj (int *np, int *pp, Real *x, int *nfp, Real *J, int *ui, void *ur, Vf vf)**
  evaluator function for residual Jacobian

Private Attributes

- **int auxprt**
  auxiliary printing bits (see Dakota Ref Manual): sum of
  - 1 = x0prt (print initial guess)
  - 2 = solprt (print final solution)
  - 4 = statpr (print solution statistics)
  - 8 = parprt (print nondefault parameters)
  - 16 = dradpr (print bound constraint drops/adds)
  use default = 31 (everything), quiet uses 3, silent uses 0.
- **int outlev**
  frequency of output summary lines in number of iterations
  (debug/verbose/normal/quiet use default = 1, silent uses 0)
- **Real dltfdj**
  finite-diff step size for computing Jacobian approximation
  (fd.gradient_step_size)
- **Real delta0**
  finite-diff step size for gradient differences for H
  (a component of some covariance approximations, if desired)
  (fd.hessian_step_size)
- **Real dltfdc**
  finite-diff step size for function differences for H
  (fd.hessian_step_size)
- **int mxfcal**
  function-evaluation limit (max_function_evaluations)
- **int mxiter**
  iteration limit (max_iterations)
- **Real rfctol**
  relative fn convergence tolerance (convergence_tolerance)
- **Real afctol**
  absolute fn convergence tolerance (absolute_conv_tol)
- **Real xctol**
x-convergence tolerance (x_conv_tol)

- Real sctol

singular convergence tolerance (singular_conv_tol)

- Real lmaxs

radius for singular-convergence test (singular_radius)

- Real xftol

false-convergence tolerance (false_conv_tol)

- int covreq
  kind of covariance required (\c covariance):
  \begin{align*}
  < 1 \text{ or } -1 & \implies \sigma^{-2} H^{-1} J H^{-1} J^T J H^{-1} J^T J \text{ or } \sigma^{-2} H^{-1} J H^{-1} J^T J H^{-1} J^T J \\
  < 2 \text{ or } -2 & \implies \sigma^{-2} H^{-1} J H^{-1} J^T J \text{ or } \sigma^{-2} H^{-1} J H^{-1} J^T J H^{-1} J^T J \\
  < 3 \text{ or } -3 & \implies \sigma^{-2} (J J^T J) J H^{-1} J^T J \text{ or } \sigma^{-2} (J J^T J) J H^{-1} J^T J H^{-1} J^T J
  \end{align*}

- int rdreq
  whether to compute the regression diagnostic vector
  \((\text{regression_diagnostics})\

- Real fprec
  expected response function precision (function_precision)

- Real lmax0
  initial trust-region radius (initial_trust_radius)

Static Private Attributes

- static NL2SOLLeastSq * nl2solInstance
  pointer to the active object instance used within the static evaluator functions

Additional Inherited Members

13.70.1 Detailed Description

Wrapper class for the NL2SOL nonlinear least squares library.

The NL2SOLLeastSq class provides a wrapper for NL2SOL (TOMS Algorithm 573), in the updated form of Port Library routines dn[fg][b ] from Bell Labs; see http://www.netlib.org/port/readme. The Fortran from Port has been turned into C by f2c. NL2SOL uses a function pointer approach for which passed functions must be either global functions or static member functions.

13.70.2 Member Function Documentation

void minimize_residuals ( ) [virtual]

Used within the least squares branch for minimizing the sum of squares residuals. Redefines the run virtual function for the least squares branch.


Implements LeastSq.
13.71 NLPQLPOptimizer Class Reference

Wrapper class for the NLPQLP optimization library, Version 2.0.

Inheritance diagram for NLPQLPOptimizer:

```
  Iterator
   |
   v
Minimizer
   |
   v
Optimizer
   |
   v
NLPQLPOptimizer
```

Public Member Functions

- **NLPQLPOptimizer (ProblemDescDB &problem_db, Model &model)**
  
  *standard constructor*

- **NLPQLPOptimizer (Model &model)**
  
  *alternate constructor*

- **~NLPQLPOptimizer ()**
  
  *destructor*

- **void find_optimum ()**
  
  *Used within the optimizer branch for computing the optimal solution. Redefines the run virtual function for the optimizer branch.*

Protected Member Functions

- **void initialize_run ()**
  
  *performs run-time set up*
Private Member Functions

- void initialize ()
  
  Shared constructor code.
- void allocate_workspace ()
  
  Allocates workspace for the optimizer.
- void deallocate_workspace ()
  
  Releases workspace memory.
- void allocate_constraints ()
  
  Allocates constraint mappings.

Private Attributes

- int L
  
  L: Number of parallel systems, i.e. function calls during line search at predetermined iterates. HINT: If only less than 10 parallel function evaluations are possible, it is recommended to apply the serial version by setting L=1.
- int numEqConstraints
  
  numEqConstraints: Number of equality constraints.
- int MMAX
  
  MMAX: Row dimension of array DG containing Jacobian of constraints. MMAX must be at least one and greater or equal to M.
- int N
  
  N: Number of optimization variables.
- int NMAX
  
  NMAX: Row dimension of C. NMAX must be at least two and greater than N.
- int MNN2
  
  MNN2: Must be equal to M+N+N+2.
- double * X
  
  X(NMAX,L): Initially, the first column of X has to contain starting values for the optimal solution. On return, X is replaced by the current iterate. In the driving program the row dimension of X has to be equal to NMAX. X is used internally to store L different arguments for which function values should be computed simultaneously.
- double * F
  
  F(L): On return, F(1) contains the final objective function value. F is used also to store L different objective function values to be computed from L iterates stored in X.
- double * G
  
  G(MMAX,L): On return, the first column of G contains the constraint function values at the final iterate X. In the driving program the row dimension of G has to be equal to MMAX. G is used internally to store L different set of constraint function values to be computed from L iterates stored in X.
- double * DF
  
  DF(NMAX): DF contains the current gradient of the objective function. In case of numerical differentiation and a distributed system (L>1), it is recommended to apply parallel evaluations of F to compute DF.
- double * DG
  
  DG(MMAX,NMAX): DG contains the gradients of the active constraints (ACTIVE(J)=.true.) at a current iterate X. The remaining rows are filled with previously computed gradients. In the driving program the row dimension of DG has to be equal to MMAX.
- double * U
U(MNN2) : U contains the multipliers with respect to the actual iterate stored in the first column of X. The first M locations contain the multipliers of the M nonlinear constraints, the subsequent N locations the multipliers of the lower bounds, and the final N locations the multipliers of the upper bounds. At an optimal solution, all multipliers with respect to inequality constraints should be nonnegative.

- double * C

C(NMAX,NMAX) : On return, C contains the last computed approximation of the Hessian matrix of the Lagrangian function stored in form of an LDL decomposition. C contains the lower triangular factor of an LDL factorization of the final quasi-Newton matrix (without diagonal elements, which are always one). In the driving program, the row dimension of C has to be equal to NMAX.

- double * D

D(NMAX) : The elements of the diagonal matrix of the LDL decomposition of the quasi-Newton matrix are stored in the one-dimensional array D.

- double ACC

ACC : The user has to specify the desired final accuracy (e.g. 1.0D-7). The termination accuracy should not be smaller than the accuracy by which gradients are computed.

- double ACCQP

ACCQP : The tolerance is needed for the QP solver to perform several tests, for example whether optimality conditions are satisfied or whether a number is considered as zero or not. If ACCQP is less or equal to zero, then the machine precision is computed by NLPQLP and subsequently multiplied by 1.0D+4.

- double STPMIN

STPMIN : Minimum steplength in case of L=1. Recommended is any value in the order of the accuracy by which functions are computed. The value is needed to compute a steplength reduction factor by STPMIN**(1/L-1). If STPMIN<0, then STPMIN=ACC is used.

- int MAXFUN

MAXFUN : The integer variable defines an upper bound for the number of function calls during the line search (e.g. 20). MAXFUN is only needed in case of L=1, and must not be greater than 50.

- int MAXIT

MAXIT : Maximum number of outer iterations, where one iteration corresponds to one formulation and solution of the quadratic programming subproblem, or, alternatively, one evaluation of gradients (e.g. 100).

- int MAX_NM

MAX_NM : Stack size for storing merit function values at previous iterations for non-monotone line search (e.g. 10). In case of MAX_NM=0, monotone line search is performed.

- double TOL_NM

TOL_NM : Relative bound for increase of merit function value, if line search is not successful during the very first step. Must be non-negative (e.g. 0.1).

- int IPRINT

IPRINT : Specification of the desired output level. IPRINT = 0 : No output of the program. IPRINT = 1 : Only a final convergence analysis is given. IPRINT = 2 : One line of intermediate results is printed in each iteration. IPRINT = 3 : More detailed information is printed in each iteration step, e.g. variable, constraint and multiplier values. IPRINT = 4 : In addition to IPRINT=3, merit function and steplength values are displayed during the line search.

- int MODE

MODE : The parameter specifies the desired version of NLPQLP. MODE = 0 : Normal execution (reverse communication!). MODE = 1 : The user wants to provide an initial guess for the multipliers in U and for the Hessian of the Lagrangian function in C and D in form of an LDL decomposition.

- int IOUT

IOUT : Integer indicating the desired output unit number, i.e. all write-statements start with 'WRITE(IOUT,... '.

U(MNN2) : U contains the multipliers with respect to the actual iterate stored in the first column of X. The first M locations contain the multipliers of the M nonlinear constraints, the subsequent N locations the multipliers of the lower bounds, and the final N locations the multipliers of the upper bounds. At an optimal solution, all multipliers with respect to inequality constraints should be nonnegative.
* int **IFAIL**

  *IFAIL*: The parameter shows the reason for terminating a solution process. Initially **IFAIL** must be set to zero. On return **IFAIL** could contain the following values: 
  **IFAIL** = -2 : Compute gradient values w.r.t. the variables stored in first column of X, and store them in DF and DG. Only derivatives for active constraints **ACTIVE(J)**= .TRUE. need to be computed. Then call NLPQLP again, see below. 
  **IFAIL** = -1 : Compute objective fn and all constraint values subject the variables found in the first L columns of X, and store them in F and G. Then call NLPQLP again, see below. 
  **IFAIL** = 0 : The optimality conditions are satisfied. 
  **IFAIL** = 1 : The algorithm has been stopped after MAXIT iterations. 
  **IFAIL** = 2 : The algorithm computed an uphill search direction. 
  **IFAIL** = 3 : Underflow occurred when determining a new approximation matrix for the Hessian of the Lagrangian. 
  **IFAIL** = 4 : The line search could not be terminated successfully. 
  **IFAIL** = 5 : Length of a working array is too short. More detailed error information is obtained with 'IPRINT'>0'. 
  **IFAIL** = 6 : There are false dimensions, for example M>MMAX, N>=NMAX, or MNN2< >M+N+N+2. 
  **IFAIL** = 7 : The search direction is close to zero, but the current iterate is still infeasible. 
  **IFAIL** = 8 : The starting point violates a lower or upper bound. 
  **IFAIL** = 9 : Wrong input parameter, i.e., **MODE**, **LDL** decomposition in D and C (in case of **MODE**=1), **IPRINT**, **IOUT** 
  **IFAIL** = 10 : Internal inconsistency of the quadratic subproblem, division by zero. 
  **IFAIL** > 100 : The solution of the quadratic programming subproblem has been terminated with an error message and **IFAIL** is set to **IFQL**+100, where **IFQL** denotes the index of an inconsistent constraint.

* double *WA

  WA(LWA) : WA is a real working array of length LWA.

* int **LWA**

  LWA : LWA value extracted from NLPQLP20.f.

* int **KWA**

  KWA(LKWA) : The user has to provide working space for an integer array.

* int **LKWA**

  LKWA : LKWA should be at least N+10.

* int **ACTIVE**

  **ACTIVE**(LACTIV) : The logical array shows a user the constraints, which NLPQLP considers to be active at the last computed iterate, i.e. G(J,X) is active, if and only if **ACTIVE**(J)=.TRUE., J=1,...,M.

* int **LACTIVE**

  LACTIVE : The length LACTIVE of the logical array should be at least 2+M+10.

* int **LQL**

  LQL : If **LQL** = .TRUE., the quadratic programming subproblem is to be solved with a full positive definite quasi-Newton matrix. Otherwise, a Cholesky decomposition is performed and updated, so that the subproblem matrix contains only an upper triangular factor.

* int **numNlpqlConstr**

  total number of constraints seen by NLPQL

* SizetList **nonlinIneqConMappingIndices**

  a list of indices for referencing the DAKOTA nonlinear inequality constraints used in computing the corresponding NLPQL constraints.

* RealList **nonlinIneqConMappingMultipliers**

  a list of multipliers for mapping the DAKOTA nonlinear inequality constraints to the corresponding NLPQL constraints.

* RealList **nonlinIneqConMappingOffsets**

  a list of offsets for mapping the DAKOTA nonlinear inequality constraints to the corresponding NLPQL constraints.

* SizetList **linIneqConMappingIndices**

  a list of indices for referencing the DAKOTA linear inequality constraints used in computing the corresponding NLPQL constraints.
13.72. **NLSSOLLEASTSQ CLASS REFERENCE**

- **RealList linIneqConMappingMultipliers**
  
  a list of multipliers for mapping the DAKOTA linear inequality constraints to the corresponding NLPQL constraints.

- **RealList linIneqConMappingOffsets**
  
  a list of offsets for mapping the DAKOTA linear inequality constraints to the corresponding NLPQL constraints.

**Additional Inherited Members**

### 13.71.1 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

\[
\begin{align*}
  \text{minimize} & \quad F(X) \\
  \text{subject to} & \quad G(J,X) = 0 , \quad J=1,...,ME \\
  & \quad G(J,X) \geq 0 , \quad J=ME+1,...,M \\
  & \quad XL \leq X \leq XU
\end{align*}
\]

and is an extension of the code NLPQLD. NLPQLP is specifically tuned to run under distributed systems. A new input parameter L is introduced for the number of parallel computers, that is the number of function calls to be executed simultaneously. In case of L=1, NLPQLP is identical to NLPQLD. Otherwise the line search is modified to allow L parallel function calls in advance. Moreover the user has the opportunity to used distributed function calls for evaluating gradients.

The algorithm is a modification of the method of Wilson, Han, and Powell. In each iteration step, a linearly constrained quadratic programming problem is formulated by approximating the Lagrangian function quadratically and by linearizing the constraints. Subsequently, a one-dimensional line search is performed with respect to an augmented Lagrangian merit function to obtain a new iterate. Also the modified line search algorithm guarantees convergence under the same assumptions as before.

For the new version, a non-monotone line search is implemented which allows to increase the merit function in case of instabilities, for example caused by round-off errors, errors in gradient approximations, etc.

The subroutine contains the option to predetermine initial guesses for the multipliers or the Hessian of the Lagrangian function and is called by reverse communication.

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

- NLPQLPOptimizer.hpp
- NLPQLPOptimizer.cpp

### 13.72 NLSSOLLEASTSQ Class Reference

Wrapper class for the NLSSOL nonlinear least squares library.

Inheritance diagram for NLSSOLLEASTSQ:
CHAPTER 13. CLASS DOCUMENTATION

Public Member Functions

- **NLSSOLLeastSq (ProblemDescDB &problem_db, Model &model)**
  standard constructor
- **NLSSOLLeastSq (Model &model)**
  alternate constructor
- **~NLSSOLLeastSq ()**
  destructor
- **void minimize_residuals ()**
  Used within the least squares branch for minimizing the sum of squares residuals. Redefines the run virtual function for the least squares branch.

Static Private Member Functions

- **static void least_sq_eval (int &mode, int &m, int &n, int &nrowfj, double *x, double *f, double *gradf, int &nstate)**
  Evaluator for NLSSOL: computes the values and first derivatives of the least squares terms (passed by function pointer to NLSSOL).

Static Private Attributes

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

Additional Inherited Members

13.72.1 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`,
13.73. NODBBASECONSTRUCTOR STRUCT REFERENCE

The parameters "convergence_tolerance", "output verbosity", "verify_level", "function precision", and "linesearch_tolerance" are mapped into NLSSOL’s "Major Iteration Limit", "Optimality Tolerance", "Major Print Level" (verbose: Major Print Level = 20; quiet: Major Print Level = 10), "Verify Level", "Function Precision", and "Linesearch Tolerance" parameters, respectively, using NLSSOL’s npoptn() subroutine (as wrapped by npoptn2() from the npoptn_wrapper.f file). Refer to [Gill, P.E., Murray, W., Saunders, M.A., and Wright, M.H., 1986] for information on NLSSOL’s optional input parameters and the npoptn() subroutine.

13.72.2 Constructor & Destructor Documentation

NLSSOLLeastSq (ProblemDescDB & problem_db, Model & model)

standard constructor

This is the primary constructor. It accepts a Model reference.

References Minimizer::constraintTol, Model::fd_gradient_step_size(), ProblemDescDB::get_int(), ProblemDescDB::get_real(), Model::gradient_type(), Iterator::iteratedModel, Iterator::maxIterations, Iterator::outputLevel, Model::probDescDB, SOLBase::set_options(), Minimizer::speculativeFlag, and Minimizer::vendorNumericalGradFlag.

NLSSOLLeastSq (Model & model)

alternate constructor

This is an alternate constructor which accepts a Model but does not have a supporting method specification from the ProblemDescDB.

References Minimizer::constraintTol, Model::fd_gradient_step_size(), Model::gradient_type(), Iterator::iteratedModel, Iterator::maxIterations, Iterator::outputLevel, SOLBase::set_options(), Minimizer::speculativeFlag, and Minimizer::vendorNumericalGradFlag.

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

* NLSSOLLeastSq.hpp
* NLSSOLLeastSq.cpp

13.73 NoDBBaseConstructor Struct Reference

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

Public Member Functions

* NoDBBaseConstructor (int=0)

C++ structs can have constructors.

13.73.1 Detailed Description

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

NoDBBaseConstructor is used to overload the constructor used for on-the-fly instantiations in which ProblemDescDB queries cannot be used. Putting this struct here avoids circular dependencies.

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

* dakota_global_def.hpp
13.74 NomadOptimizer Class Reference

Wrapper class for NOMAD Optimizer.
Inheritance diagram for NomadOptimizer:

Classes
- class Evaluator
  NOMAD-based Evaluator class.

Public Member Functions
- NomadOptimizer (ProblemDescDB &problem_db, Model &model)
  Constructor.
- NomadOptimizer (Model &model)
  alternate constructor for Iterator instantiations without DB
- ~NomadOptimizer ()
  Destructor.
- void find_optimum ()
  Calls the NOMAD solver.

Private Member Functions
- void load_parameters (Model &model)
  Convenience function for Parameter loading.

Private Attributes
- int numTotalVars
  Total across all types of variables.
- int numNomadNonlinearIneqConstraints
  Number of nonlinear inequality constraints after put into the format required by Nomad.
- int randomSeed
  Parameters passes to Nomad.
- int maxBlackBoxEvals
- int maxIterations
13.74. NOMADOPTIMIZER CLASS REFERENCE

- std::string outputFormat
- std::string historyFile
- bool displayAll
- Real epsilon
- Real vns
- NOMAD::Point initialPoint
  
  Pointer to Nomad initial point.
- NOMAD::Point upperBound
  
  Pointer to Nomad upper bounds.
- NOMAD::Point lowerBound
  
  Pointer to Nomad lower bounds.
- std::vector<int> constraintMapIndices
  
  map from Dakota constraint number to Nomad constraint number
- std::vector<double> constraintMapMultipliers
  
  multipliers for constraint transformations
- std::vector<double> constraintMapOffsets
  
  offsets for constraint transformations

Additional Inherited Members

13.74.1 Detailed Description

Wrapper class for NOMAD Optimizer.

NOMAD (is a Nonlinear Optimization by Mesh Adaptive Direct search) is a simulation-based optimization package designed to efficiently explore a design space using Mesh Adaptive Search.

Mesh Adaptive Direct Search uses Meshes, discretizations of the domain space of variables. It generates multiple meshes, and as its name implies, it also adapts the refinement of the meshes in order to find the best solution of a problem.

The objective of each iteration is to find points in a mesh that improves the current solution. If a better solution is not found, the next iteration is done over a finer mesh.

Each iteration is composed of two steps: Search and Poll. The Search step finds any point in the mesh in an attempt to find an improvement; while the Poll step generates trial mesh points surrounding the current best current solution.

The NomadOptimizer is a wrapper for the NOMAD library. It features the following attributes: max_function_evaluations, display_format, display_all_evaluations, function_precision, max_iterations.

13.74.2 Constructor & Destructor Documentation

NomadOptimizer ( ProblemDescDB & problem_db, Model & model )

Constructor.

NOMAD Optimizer Constructor
Parameters

| model | DAKOTA Model object |

References ProblemDescDB::get_bool(), ProblemDescDB::get_int(), ProblemDescDB::get_real(), ProblemDescDB::get_string(), NomadOptimizer::load_parameters(), Iterator::probDescDB, and NomadOptimizer::randomSeed.

### 13.74.3 Member Function Documentation

```cpp
def void load_parameters ( Model & model ) [private]
```

Convenience function for Parameter loading.

This function takes the Parameters provided by the user in the DAKOTA model.

Parameters

| model | NOMAD Model object |

References Dakota::NPOS, Dakota::abort_handler(), Minimizer::bigIntBoundSize, Minimizer::bigRealBoundSize, NomadOptimizer::constraintMapIndices, NomadOptimizer::constraintMapMultipliers, NomadOptimizer::constraintMapOffsets, Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), Model::continuous_variables(), Model::discrete_int_lower_bounds(), Model::discrete_int_sets(), Model::discrete_int_upper_bounds(), Model::discrete_int_variables(), Model::discrete_real_lower_bounds(), Model::discrete_real_upper_bounds(), Model::discrete_real_variables(), Model::discrete_set_int_values(), Model::discrete_set_real_values(), NomadOptimizer::initialPoint, Iterator::iteratedModel, NomadOptimizer::lowerBound, Model::nonlinear_eq_constraint_targets(), Model::nonlinear_ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Minimizer::numContinuousVars, Minimizer::numDiscreteIntVars, Minimizer::numDiscreteRealVars, NomadOptimizer::numNomadNonlinearIneqConstraints, Minimizer::numNonlinearEqConstraints, Minimizer::numNonlinearIneqConstraints, NomadOptimizer::numTotalVars, Dakota::set_value_to_index(), and NomadOptimizer::upperBound.

Referenced by NomadOptimizer::NomadOptimizer().

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

- NomadOptimizer.hpp
- NomadOptimizer.cpp

### 13.75 NonD Class Reference

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

Inheritance diagram for NonD:
Public Member Functions

- void `initialize_random_variables` (short u_space_type)
  
  *initialize natafTransform based on distribution data from iteratedModel*

- void `initialize_random_variables` (const Pecos::ProbabilityTransformation &transform)
  
  *alternate form: initialize natafTransform based on incoming data*

- void `requested_levels` (const RealVectorArray &req_resp_levels, const RealVectorArray &req_prob_levels, const RealVectorArray &req_rel_levels, const RealVectorArray &req_gen_rel_levels, short resp_lev_tgt, short resp_lev_tgt_reduce, bool cdf_flag)
  
  *set requestedRespLevels, requestedProbLevels, requestedRelLevels, requestedGenRelLevels, respLevelTarget, and cdfFlag (used in combination with alternate ctors)*

- void `distribution_parameter_derivatives` (bool dist_param_derivs)
  
  *set distParamDerivs*

- bool `pdf_output` () const
  
  *get pdfOutput*

- void `pdf_output` (bool output)
  
  *set pdfOutput*

- Pecos::ProbabilityTransformation & `variable_transformation` ()
  
  *return natafTransform*
Protected Member Functions

- **NonD** (ProblemDescDB &problem_db, Model &model)
  constructor
- **NonD** (unsigned short method_name, Model &model)
  alternate constructor for sample generation and evaluation "on the fly"
- **NonD** (unsigned short method_name, const RealVector &lower_bnds, const RealVector &upper_bnds)
  alternate constructor for sample generation "on the fly"
- **~NonD** ()
  destructor
- **void initialize_run()**
  utility function to perform common operations prior to pre_run(); typically memory initialization; setting of instance pointers
- **void core_run()**
- **void finalize_run()**
  utility function to perform common operations following post_run(); deallocation and resetting of instance pointers
- **const Response & response_results()**
  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
  Mapping of the core_run() virtual function for the NonD branch.
- **virtual void initialize_response_covariance()**
  initializes respCovariance
- **virtual void initialize_final_statistics()**
  initializes finalStatistics for storing NonD final results
- **virtual void update_final_statistics()**
  update finalStatistics::functionValues
- **int generate_system_seed()**
  create a system-generated unique seed (when a seed is unspecified)
- **void initialize_random_variable_transformation()**
  instantiate natafTransform
- **void initialize_random_variable_types(short u_space_type)**
  initializes ranVarTypesX and ranVarTypesU within natafTransform
- **void initialize_random_variable_parameters()**
  initializes ranVarMeansX, ranVarStdDevsX, ranVarLowerBndsX, ranVarUpperBndsX, and ranVarAddtlParamsX within natafTransform
- **void initialize_random_variable_correlations()**
  propagate iteratedModel correlations to natafTransform
- **void verify_correlation_support()**
  verify that correlation warping supported by Der Kiureghian & Liu for given variable types
- **void initialize_final_statistics_gradients()**
  initializes finalStatistics::functionGradVal
- **void update_aleatory_final_statistics()**
update finalStatistics::functionValues from momentStats and computed\{Prob,Rel,GenRel,Resp\} Levels

- **void update_system_final_statistics** ()
  - update system metrics from component metrics within finalStatistics

- **void update_system_final_statistics_gradients** ()
  - update finalStatistics::functionGradients

- **void initialize_distribution_mappings** ()
  - size computed\{Resp,Prob,Rel,GenRel\} Levels

- **void print_distribution_mappings** (std::ostream &s) const
  - prints the z/p/beta/beta+ mappings reflected in \{requested,computed\}\{Resp,Prob,Rel,GenRel\} Levels

- **void print_system_mappings** (std::ostream &s) const
  - print system series/parallel mappings for response levels

- **void transform_model** (Model &x_model, Model &u_model, bool global_bounds=false, Real bound=10.)
  - recast x_model from x-space to u-space to create u_model

- **void construct_lhs** (Iterator &u_space_sampler, Model &u_model, unsigned short sample_type, int num_samples, int seed, const String &rng, bool vary_pattern, short sampling_vars_mode=ACTIVE)
  - assign a NonDLHSSampling instance within u_space_sampler

- **void archiveAllocate_mappings** ()
  - allocate results array storage for distribution mappings

- **void archive_from_resp** (size_t fn_index)
  - archive the mappings from specified response levels for specified fn

- **void archive_to_resp** (size_t fn_index)
  - archive the mappings to computed response levels for specified fn

**Static Protected Member Functions**

- **static void vars_u_to_x_mapping** (const Variables &u_vars, Variables &x_vars)
  - static function for RecastModels used for forward mapping of u-space variables from NonD Iterators to x-space variables for Model evaluations

- **static void vars_x_to_u_mapping** (const Variables &x_vars, Variables &u_vars)
  - static function for RecastModels used for inverse mapping of x-space variables from data import to u-space variables for NonD Iterators

- **static void set_u_to_x_mapping** (const Variables &u_vars, const ActiveSet &u_set, ActiveSet &x_set)
  - static function for RecastModels used to map u-space ActiveSets from NonD Iterators to x-space ActiveSets for Model evaluations

- **static void resp_x_to_u_mapping** (const Variables &x_vars, const Variables &u_vars, const Response &x_response, Response &u_response)
  - static function for RecastModels used to map x-space responses from Model evaluations to u-space responses for return to NonD Iterator.

**Protected Attributes**

- **NonD * prevNondInstance**
  - pointer containing previous value of nondInstance

- **Pecos::ProbabilityTransformation natafTransform**
Nonlinear variable transformation that encapsulates the required data for performing transformations from \( X \Rightarrow Z \Rightarrow U \) and back.

- `size_t numContDesVars`  
  number of continuous design variables (modeled using uniform distribution for All view modes)
- `size_t numDiscIntDesVars`  
  number of discrete integer design variables (modeled using discrete histogram distributions for All view modes)
- `size_t numDiscRealDesVars`  
  number of discrete real design variables (modeled using discrete histogram distributions for All view modes)
- `size_t numDesignVars`  
  total number of design variables
- `size_t numContStateVars`  
  number of continuous state variables (modeled using uniform distribution for All view modes)
- `size_t numDiscIntStateVars`  
  number of discrete integer state variables (modeled using discrete histogram distributions for All view modes)
- `size_t numDiscRealStateVars`  
  number of discrete real state variables (modeled using discrete histogram distributions for All view modes)
- `size_t numStateVars`  
  total number of state variables
- `size_t numNormalVars`  
  number of normal uncertain variables (native space)
- `size_t numLognormalVars`  
  number of lognormal uncertain variables (native space)
- `size_t numUniformVars`  
  number of uniform uncertain variables (native space)
- `size_t numLoguniformVars`  
  number of loguniform uncertain variables (native space)
- `size_t numTriangularVars`  
  number of triangular uncertain variables (native space)
- `size_t numExponentialVars`  
  number of exponential uncertain variables (native space)
- `size_t numBetaVars`  
  number of beta uncertain variables (native space)
- `size_t numGammaVars`  
  number of gamma uncertain variables (native space)
- `size_t numGumbelVars`  
  number of gumbel uncertain variables (native space)
- `size_t numFrechetVars`  
  number of frechet uncertain variables (native space)
- `size_t numWeibullVars`  
  number of weibull uncertain variables (native space)
- `size_t numHistogramBinVars`  
  number of histogram bin uncertain variables (native space)
- `size_t numPoissonVars`
number of Poisson uncertain variables (native space)
• size_t numBinomialVars
  number of binomial uncertain variables (native space)
• size_t numNegBinomialVars
  number of negative binomial uncertain variables (native space)
• size_t numGeometricVars
  number of geometric uncertain variables (native space)
• size_t numHyperGeomVars
  number of hypergeometric uncertain variables (native space)
• size_t numHistogramPtVars
  number of histogram point uncertain variables (native space)
• size_t numContIntervalVars
  number of continuous interval uncertain variables (native space)
• size_t numDiscIntervalVars
  number of discrete interval uncertain variables (native space)
• size_t numDiscSetIntUncVars
  number of discrete integer set uncertain variables (native space)
• size_t numDiscSetRealUncVars
  number of discrete real set uncertain variables (native space)
• size_t numContAleatUncVars
  total number of aleatory uncertain variables (native space)
• size_t numDiscIntAleatUncVars
  total number of aleatory uncertain variables (native space)
• size_t numDiscRealAleatUncVars
  total number of aleatory uncertain variables (native space)
• size_t numAleatoryUncVars
  total number of aleatory uncertain variables (native space)
• size_t numContEpistUncVars
  total number of epistemic uncertain variables (native space)
• size_t numDiscIntEpistUncVars
  total number of epistemic uncertain variables (native space)
• size_t numDiscRealEpistUncVars
  total number of epistemic uncertain variables (native space)
• size_t numEpistemicUncVars
  total number of epistemic uncertain variables (native space)
• size_t numUncertainVars
  total number of uncertain variables (native space)
• bool epistemicStats
  flag for computing interval-type metrics instead of integrated metrics If any epistemic variables are active in a
  metric evaluation, then this flag is set.
• RealMatrix momentStats
  moments of response functions (mean, std deviation, skewness, and kurtosis calculated in compute_moments()),
  indexed as (moment,fn)
• RealVectorArray requestedRespLevels
  requested response levels for all response functions

• RealVectorArray computedProbLevels
  output probability levels for all response functions resulting from requestedRespLevels

• RealVectorArray computedRelLevels
  output reliability levels for all response functions resulting from requestedRespLevels

• RealVectorArray computedGenRelLevels
  output generalized reliability levels for all response functions resulting from requestedRespLevels

• short respLevelTarget
  indicates mapping of $z > p$ (PROBABILITIES), $z > \beta$ (RELIABILITIES), or $z > \beta \ast$ (GEN_RELIABILITIES)

• short respLevelTargetReduce
  indicates component or system series/parallel failure metrics

• RealVectorArray requestedProbLevels
  requested probability levels for all response functions

• RealVectorArray requestedRelLevels
  requested reliability levels for all response functions

• RealVectorArray requestedGenRelLevels
  requested generalized reliability levels for all response functions

• RealVectorArray computedRespLevels
  output response levels for all response functions resulting from requestedProbLevels, requestedRelLevels, or requestedGenRelLevels

• size_t totalLevelRequests
  total number of levels specified within requestedRespLevels, requestedProbLevels, and requestedRelLevels

• bool cdfFlag
  flag for type of probabilities/reliabilities used in mappings: cumulative/CDF (true) or complementary/CCDF (false)

• bool pdfOutput
  flag for managing output of response probability density functions (PDFs)

• Response finalStatistics
  final statistics from the uncertainty propagation used in strategies: response means, standard deviations, and probabilities of failure

Static Protected Attributes

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

Private Member Functions

• void distribute_levels (RealVectorArray &levels, bool ascending=true)
  convenience function for distributing a vector of levels among multiple response functions if a short-hand specification is employed.

• void distribution_mappings_file (size_t fn_index) const
  Write distribution mappings to a file for a single response.

• void print_distribution_map (size_t fn_index, std::ostream &s) const
Print distribution mapping for a single response function to ostream.

• unsigned short pecos_to_dakota_variable_type (unsigned short pecos_var_type)
convert from Pecos To Dakota variable enumeration type for continuous aleatory uncertain variables used in variable transformations

Private Attributes

• bool distParamDerivs
flags calculation of derivatives with respect to distribution parameters s within resp x_to_u_mapping() using the chain rule df/dx dx/ds. The default is to calculate derivatives with respect to standard random variables u using the chain rule df/dx dx/du.

13.75.1 Detailed Description
Base class for all nondeterministic iterators (the DAKOTA/UQ branch).
The base class for nondeterministic iterators consolidates uncertain variable data and probabilistic utilities for inherited classes.

13.75.2 Member Function Documentation

void initialize_random_variables ( short u_space_type )
initialize natafTransform based on distribution data from iteratedModel
Build ProbabilityTransformation::ranVar arrays containing the uncertain variable distribution types and their corresponding means/standard deviations. This function is used when the Model variables are in x-space.
References NonD::initialize_random_variable_correlations(), NonD::initialize_random_variable_parameters(), NonD::initialize_random_variable_transformation(), NonD::initialize_random_variable_types(), NonD::natafTransform, and NonD::verify_correlation_support().
Referenced by NonDExpansion::compute_statistics(), NonDGlobalReliability::importance_sampling(), NonDLocalReliability::initialize_class_data(), NonDExpansion::initialize_expansion(), NonDAadaptImpSampling::NonDAdaptImpSampling(), and NonDGlobalReliability::optimize_gaussian_process().

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.
References NonD::initialize_random_variable_transformation(), NonD::natafTransform, NonD::numContDesVars, and NonD::numContStateVars.

void initialize_run ( ) [inline], [protected], [virtual]
utility function to perform common operations prior to pre_run(); typically memory initialization; setting of instance pointers
Perform initialization phases of run sequence, like allocating memory and setting instance pointers. Commonly used in sub-iterator executions. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s initialize_run(), typically before performing its own implementation steps.
Reimplemented from Analyzer.
References Analyzer::initialize_run(), NonD::nondInstance, and NonD::prevNondInstance.
void core_run( ) [inline], [protected], [virtual]
Performs a forward uncertainty propagation of parameter distributions into response statistics.
Reimplemented from Iterator.
References Analyzer::bestVarsRespMap, and NonD::quantify_uncertainty().

void finalize_run( ) [inline], [protected], [virtual]
utility function to perform common operations following post_run(); deallocation and resetting of instance pointers
Optional: perform finalization phases of run sequence, like deallocating memory and resetting instance pointers.
Commonly used in sub-iterator executions. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s finalize_run(), typically after performing its own implementation steps.
Reimplemented from Iterator.
References Iterator::finalize_run(), NonD::nondInstance, and NonD::prevNondInstance.

void initialize_final_statistics( ) [protected], [virtual]
initializes finalStatistics for storing NonD final results
Default definition of virtual function (used by sampling, reliability, and stochastic expansion methods) defines
the set of statistical results to include means, standard deviations, and level mappings.
Reimplemented in NonDInterval.
References Dakota::abort_handler(), NonD::cdfFlag, Model::cv(), ActiveSet::derivative_vector(), NonD::epistemicStats, NonD::finalStatistics, Response::function_labels(), Model::inactive_continuous_variable_ids(), Iterator::iteratedModel, Analyzer::numFunctions, NonD::requestedGenRelLevels, NonD::requestedProbLevels, NonD::requestedRelLevels, NonD::requestedRespLevels, NonD::respLevelTarget, NonD::respLevelTargetReduce, and NonD::totalLevelRequests.
Referenced by NonDExpansion::NonDExpansion(), NonDIntegration::NonDIntegration(), NonDReliability::NonDReliability(), NonDSampling::NonDSampling(), and NonD::requested_levels().

void initialize_random_variable_types( short u_space_type ) [protected]
initializes ranVarTypesX and ranVarTypesU within natafTransform
Build ProbabilityTransformation::ranVar arrays containing the uncertain variable distribution types and their corresponding means/standard deviations. This function is used when the Model variables are in x-space.
References Dakota::abort_handler(), Model::aleatory_distribution_parameters(), Model::cv(), Iterator::iteratedModel, NonD::natafTransform, NonD::numBetaVars, NonD::numContDesVars, NonD::numContIntervalVars, NonD::numContStateVars, NonD::numExponentialVars, NonD::numFrechetVars, NonD::numGammaVars, NonD::numGumbelVars, NonD::numHistogramBinVars, NonD::numLognormalVars, NonD::numLoguniformVars, NonD::numNormalVars, NonD::numTriangularVars, NonD::numUniformVars, and NonD::numWeibullVars.
Referenced by NonDExpansion::initialize(), NonD::initialize_random_variables(), NonDBayesCalibration::NonDBayesCalibration(), NonDIntegration::NonDIntegration(), and NonDReliability::NonDReliability().

void initialize_random_variable_parameters( ) [protected]
initializes ranVarMeansX, ranVarStdDevX, ranVarLowerBndsX, 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.
References Dakota::abort_handler(), Model::aleatory_distribution_parameters(), Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), Model::cv(), Iterator::iteratedModel, NonD::natafTransform, NonD::numBetaVars, NonD::numContDesVars, NonD::numContIntervalVars, NonD::numContStateVars, NonD::numExponentialVars, NonD::numFrechetVars, NonD::numGammaVars, NonD::numGumbelVars, NonD::numHistogramBinVars, NonD::numLognormalVars, NonD::numLoguniformVars, NonD::numNormalVars, NonD::numTriangularVars, NonD::numUniformVars, and NonD::numWeibullVars.

Referenced by NonDExpansion::initialize_expansion(), NonD::initialize_random_variables(), NonDLocalReliability::mean_value(), NonDLocalReliability::mpp_search(), NonDGlobalReliability::quantify_uncertainty(), and NonDBayesCalibration::quantify_uncertainty().

```cpp
void print_distribution_mappings ( std::ostream & s ) const [protected]
```
prints the z/p/beta/beta∗ mappings reflected in {requested,computed} {Resp,Prob,Rel,GenRel} Levels
print distribution mappings, including to file per response.
References NonD::distribution_mappings_file(), Analyzer::numFunctions, Iterator::outputLevel, NonD::print_distribution_map(), NonD::requestedGenRelLevels, NonD::requestedProbLevels, NonD::requestedRelLevels, NonD::requestedRespLevels, and Dakota::write_precision.
Referenced by NonDExpansion::print_results(), NonDAdaptImpSampling::print_results(), NonDGPImpSampling::print_results(), NonDAdaptiveSampling::print_results(), NonDPOFDarts::print_results(), and NonDSampling::print_statistics().

```cpp
void vars_u_to_x_mapping ( const Variables & u_vars, Variables & x_vars ) [static], [protected]
```
static function for RecastModels used for forward mapping of u-space variables from NonD Iterators to x-space variables for Model evaluations.
Map the variables from iterator space (u) to simulation space (x).
References Variables::continuous_variables(), Variables::continuous_variables_view(), NonD::natafTransform, and NonD::nondInstance.
Referenced by NonD::transform_model().

```cpp
void vars_x_to_u_mapping ( const Variables & x_vars, Variables & u_vars ) [static], [protected]
```
static function for RecastModels used for inverse mapping of x-space variables from data import to u-space variables for NonD Iterators.
Map the variables from simulation space (x) to iterator space (u).
References Variables::continuous_variables(), Variables::continuous_variables_view(), NonD::natafTransform, and NonD::nondInstance.
Referenced by NonD::transform_model().

```cpp
void set_u_to_x_mapping ( const Variables & u_vars, const ActiveSet & u_set, ActiveSet & x_set ) [static], [protected]
```
static function for RecastModels used to map u-space ActiveSets from NonD Iterators to x-space ActiveSets for Model evaluations.
Define the DVV for x-space derivative evaluations by augmenting the iterator requests to account for correlations.
References Dakota::NPOS, Variables::all_continuous_variable_ids(), Dakota::contains(), Variables::continuous_variable_ids(), ActiveSet::derivative_vector(), Dakota::find_index(), Variables::inactive_continuous_variable_ids(), NonD::natafTransform, and NonD::nondInstance.
Referenced by NonD::transform_model().
void print_distribution_map(size_t fn_index, std::ostream &s) const [private]

Print distribution mapping for a single response function to ostream.

Print the distribution mapping for a single response function to the passed output stream.

References NonD::cdfFlag, NonD::computedGenRelLevels, NonD::computedProbLevels, NonD::computedRelLevels, NonD::computedRespLevels, Iterator::iteratedModel, NonD::requestedGenRelLevels, NonD::requestedProbLevels, NonD::requestedRelLevels, NonD::requestedRespLevels, NonD::respLevelTarget, Model::response_labels(), and Dakota::write_precision.

Referenced by NonD::distribution_mappings_file(), and NonD::print_distribution_mappings().

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

- DakotaNonD.hpp
- DakotaNonD.cpp

13.76 NonDAdaptImpSampling Class Reference

Class for the Adaptive Importance Sampling methods within DAKOTA.

Inheritance diagram for NonDAdaptImpSampling:

```
NonDAdaptImpSampling
|    |
|    | NonDSampling
|    | NonD
|    | Iterator
|    |
```

Public Member Functions

- NonDAdaptImpSampling (ProblemDescDB &problem_db, Model &model)
  - standard constructor
- NonDAdaptImpSampling (Model &model, unsigned short sample_type, int samples, int seed, const String &rng, bool vary_pattern, unsigned short is_type, bool cdf_flag, bool x_space_model, bool use_model_-
  bounds)
  - alternate constructor for on-the-fly instantiations
- ~NonDAdaptImpSampling ()
  - destructor
- void init_communicators ()
  - initialize the communicators associated with iteratedModel as well as any subordinate iterators or subordinate models
- void free_communicators ()
  - free the communicators associated with iteratedModel as well as any subordinate iterators or subordinate models
- **void quantify_uncertainty** ()
  performs adaptive importance sampling and computes probability of failure

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

- **void initialize** (const RealVectorArray &full_points, bool x_space_data, size_t resp_index, Real initial_prob, Real failure_threshold)
  initializes data needed for importance sampling: an initial set of points around which to sample, a failure threshold, an initial probability to refine, and flags to control transformations

- **void initialize** (const RealMatrix &full_points, bool x_space_data, size_t resp_index, Real initial_prob, Real failure_threshold)
  initializes data needed for importance sampling: an initial set of points around which to sample, a failure threshold, an initial probability to refine, and flags to control transformations

- **void initialize** (const RealVector &full_point, bool x_space_data, size_t resp_index, Real initial_prob, Real failure_threshold)
  initializes data needed for importance sampling: an initial point around which to sample, a failure threshold, an initial probability to refine, and flags to control transformations

- **Real final_probability** ()
  returns the final probability calculated by the importance sampling

### Private Member Functions

- **void select_rep_points** (const RealVectorArray &var_samples_u, const RealVector &fn_samples)
  select representative points from a set of samples

- **void converge_statistics** (bool cov_flag)
  iteratively generate samples and select representative points until probability and (optionally) coefficient of variation converge

- **void generate_samples** (RealVectorArray &var_samples_u)
  generate a set of samples based on multimodal sampling density

- **void evaluate_samples** (const RealVectorArray &var_samples_u, RealVector &fn_samples)
  evaluate the model at the sample points and store the responses

- **void calculate_statistics** (const RealVectorArray &var_samples_u, const RealVector &fn_samples, size_t total_samples, Real &sum_prob, Real &prob, bool compute_cov, Real &sum_var, Real &cov)
  calculate the probability of exceeding the failure threshold and the coefficient of variation (if requested)

- **Real distance** (const RealVector &a, const RealVector &b)
  compute Euclidean distance between points a and b

- **Real recentered_density** (const RealVector &sample_point)
  compute density between a representative point and a sample point, assuming standard normal

### Private Attributes

- **Model uSpaceModel**
  importance sampling is performed in standardized probability space. This u-space model is either passed in (alternate constructor for helper AIS) or constructed using transform_model() (standard constructor for stand-alone AIS)

- **unsigned short importanceSamplingType**
  integration type (is, ais, mmais) provided by input specification
• bool initLHS
  flag to identify if initial points are generated from an LHS sample
• bool useModelBounds
  flag to control if the sampler should respect the model bounds
• bool invertProb
  flag for inversion of probability values using 1.-p
• int refineSamples
  size of sample batch within each refinement iteration
• size_t respFnIndex
  the active response function index in the model to be sampled
• RealVector designPoint
  design subset for which uncertain subset is being sampled
• RealVectorArray initPointsU
  the original set of u-space samples passed in initialize()
• RealVectorArray repPointsU
  the set of representative points in u-space around which to sample
• RealVector repWeights
  the weight associated with each representative point
• Real probEstimate
  the probability estimate that is iteratively refined by importance sampling
• Real failThresh
  the failure threshold (z-bar) for the problem.

Additional Inherited Members

13.76.1 Detailed Description

Class for the Adaptive Importance Sampling methods within DAKOTA.

The NonDAdaptImpSampling implements the multi-modal adaptive importance sampling used for reliability calculations. (eventually we will want to broaden this). Need to add more detail to this description.

13.76.2 Constructor & Destructor Documentation

NonDAdaptImpSampling ( ProblemDescDB & problem_db, Model & model )

standard constructor

This is the primary constructor. It accepts a Model reference. It will perform refinement for all response QOI and all probability levels.

References NonD::initialize_random_variables(), Iterator::iteratedModel, NonDSampling::numSamples, NonDAdaptImpSampling::refineSamples, NonDSampling::statsFlag, NonD::transform_model(), NonDAdaptImpSampling::useModelBounds, and NonDAdaptImpSampling::uSpaceModel.
NonDAdaptImpSampling ( Model & model, unsigned short sample_type, int refine_samples, int refine_seed, const String & rng, bool vary_pattern, unsigned short is_type, bool cdf_flag, bool x_space_model, bool use_model_bounds )

alternate constructor for on-the-fly instantiations

This is an alternate constructor for instantiations on the fly using a Model but no ProblemDescDB. It will perform refinement for one response QOI and one probability level (passed in initialize()).

References NonD::cdfFlag, Iterator::maxEvalConcurrency, NonDAdaptImpSampling::refineSamples, NonD::transform_model(), NonDAdaptImpSampling::useModelBounds, and NonDAdaptImpSampling::uSpaceModel.

13.76.3 Member Function Documentation

void initialize ( const RealVectorArray & acv_points, bool x_space_data, size_t resp_index, Real initial_prob, Real failure_threshold )

initializes data needed for importance sampling: an initial set of points around which to sample, a failure threshold, an initial probability to refine, and flags to control transformations

Initializes data using a vector array of starting points.

References NonDAdaptImpSampling::designPoint, NonDAdaptImpSampling::failThresh, NonDAdaptImpSampling::initPointsU, NonDAdaptImpSampling::invertProb, NonD::natafTransform, NonD::numContDesVars, NonD::numUncertainVars, NonDAdaptImpSampling::probEstimate, and NonDAdaptImpSampling::respFnIndex.

Referenced by NonDExpansion::compute_statistics(), NonDGlobalReliability::importance_sampling(), and NonDAdaptImpSampling::quantify_uncertainty().

void initialize ( const RealMatrix & acv_points, bool x_space_data, size_t resp_index, Real initial_prob, Real failure_threshold )

initializes data needed for importance sampling: an initial set of points around which to sample, a failure threshold, an initial probability to refine, and flags to control transformations

Initializes data using a matrix of starting points.

References NonDAdaptImpSampling::designPoint, NonDAdaptImpSampling::failThresh, NonDAdaptImpSampling::initPointsU, NonDAdaptImpSampling::invertProb, NonD::natafTransform, NonD::numContDesVars, Analyzer::numContinuousVars, NonD::numUncertainVars, NonDAdaptImpSampling::probEstimate, and NonDAdaptImpSampling::respFnIndex.

void initialize ( const RealVector & acv_point, bool x_space_data, size_t resp_index, Real initial_prob, Real failure_threshold )

initializes data needed for importance sampling: an initial point around which to sample, a failure threshold, an initial probability to refine, and flags to control transformations

Initializes data using only one starting point.

References NonDAdaptImpSampling::designPoint, NonDAdaptImpSampling::failThresh, NonDAdaptImpSampling::initPointsU, NonDAdaptImpSampling::invertProb, NonD::natafTransform, NonD::numContDesVars, NonD::numUncertainVars, NonDAdaptImpSampling::probEstimate, and NonDAdaptImpSampling::respFnIndex.

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

- NonDAdaptImpSampling.hpp
- NonDAdaptImpSampling.cpp
13.77  **NonDAdaptiveSampling Class Reference**

Class for testing various Adaptively sampling methods using geometric, statistical, and topological information of the surrogate.

Inheritance diagram for NonDAdaptiveSampling:

```
  NonDAdaptiveSampling
  |                      
  v                      
Analyzer
  |                      
  v                      
NonD
  |                      
  v                      
NonDSampling
  |                      
  v                      
NonDAdaptiveSampling
```

**Public Member Functions**

- **NonDAdaptiveSampling** *(ProblemDescDB &problem_db, Model &model)*  
  *standard constructor*
- **~NonDAdaptiveSampling** ()  
  *alternate constructor for sample generation and evaluation “on the fly” has not been implemented*

**Protected Member Functions**

- void **init_communicators** ()  
  *initialize the communicators associated with iteratedModel as well as any subordinate iterators or subordinate models*
- void **free_communicators** ()  
  *free the communicators associated with iteratedModel as well as any subordinate iterators or subordinate models*
- void **quantify_uncertainty** ()  
  *perform the GP importance sampling and return probability of failure.*
- Real **final_probability** ()  
  *returns the probability calculated by the importance sampling*
- void **print_results** (std::ostream &s)  
  *print the final statistics*

**Private Member Functions**

- void **calc_score_alm** ()  
  *Function to compute the ALM scores for the candidate points ALM score is the variance computed by the surrogate at the point.*
- void **calc_score_delta_x** ()
Function to compute the Distance scores for the candidate points. Distance score is the shortest distance between the candidate and an existing training point.

- void calc_score_delta_y ()

Function to compute the Gradient scores for the candidate points. Gradient score is the function value difference between a candidate’s surrogate response and its nearest evaluated true response from the training set.

- void calc_score_topo_bottleneck ()

Function to compute the Bottleneck scores for the candidate points. Bottleneck score is computed by determining the bottleneck distance between the persistence diagrams of two approximate Morse-Smale complices. The complices used include one built from only the training data, and another built from the training data and the single candidate.

- void calc_score_topo_avg_persistence (int respFnCount)

Function to compute the Average Change in Persistence scores for the candidate points. Avg Persistence score is computed as the average change in persistence each point undergoes between two approximate Morse-Smale complices. The complices used include one built from only the training data, and another built from the training data and the single candidate.

- void calc_score_topo_highest_persistence (int respFnCount)

Function to compute the Highest Persistence scores for the candidate points. Highest Persistence score is calculated as a ranking of a set of candidates by constructing an approximate Morse-Smale complex over the entire set of candidates, using their surrogate responses, and the training data, using their true responses, and ranking points based on the most topological significance as measured by their persistence values. In the case where there are no topologically significant points, the point will be chosen randomly TODO: It may be wiser to fall back to a scheme that ranks points based on proximity to extrema, or the most significant extremum?

- void calc_score_topo_alm_hybrid (int respFnCount)

Function to compute the Hybrid scores for the candidate points. Hybrid score is computed the same as Average Persistence score except that instead of computing one score, three scores are computing not only a mean surface, but a mean +/- std. dev. surfaces and then averaging the three separate scores. The hope is that you strike a balance between selecting points in topologically important areas and areas of high uncertainty.

- Real calc_score_alm (int respFnCount, RealVector &test_point)

Same as the other function of the same name, only this allows the user to specify the location of the candidate.

- Real calc_score_delta_x (int respFnCount, RealVector &test_point)

Same as the other function of the same name, only this allows the user to specify the location of the candidate.

- Real calc_score_delta_y (int respFnCount, RealVector &test_point)

Same as the other function of the same name, only this allows the user to specify the location of the candidate.

- Real calc_score_topo_bottleneck (int respFnCount, RealVector &test_point)

Same as the other function of the same name, only this allows the user to specify the location of the candidate.

- Real calc_score_topo_avg_persistence (int respFnCount, RealVector &test_point)

Same as the other function of the same name, only this allows the user to specify the location of the candidate.

- Real calc_score_topo_alm_hybrid (int respFnCount, RealVector &test_point)

Same as the other function of the same name, only this allows the user to specify the location of the candidate.

- Real compute_rmspe ()

Using the validationSet, compute the RMSE over the surface.

- void compare_complices (int dim, std::ostream &output)

Using the validationSet, compute the approximate Morse-Smale complices of the true model over the validationSet as well as the surrogate model over the validationSet, and output some topological comparisons.

- void parse_options ()

Parse misc options specified in a user input deck.

- RealVectorArray drawNewX (int this_k, int respFnCount=0)
function to pick the next X value to be evaluated by the Iterated model

- void **output_round_data** (int round, int respFnCount=0)
  Temporary function for dumping validation data to output files to be visualized in TopoAS.

- void **update_amsc** (int respFnCount=0)
  Update the approximate Morse-Smale complex based on the training points and selected candidates. Uses surrogate function responses.

- void **construct_fsu_sampler** (Iterator &u_space_sampler, Model &u_model, int num_samples, int seed, unsigned short sample_type)
  Copy of construct_lhs only it allows for the construction of FSU sample designs. This can break the fsu_cvt, so it is not used at the moment, and these designs only affect the initial sample build not the candidate sets constructed at each round.

- void **output_for_optimization** (int dim)
  This function will write an input deck for a multi-start global optimization run of DAKOTA by extracting all of the local minima off the approximate Morse-Smale complex created from the validation set of the surrogate model.

- Real **median** (const RealVector &sorted_data)
  compute the median of the sorted values passed in

- void **pick_new_candidates** ()
  Pick new candidates from Emulator.

- void **score_new_candidates** ()
  Score New candidates based on the chosen metrics.

**Private Attributes**

- **Iterator gpBuild**
  LHS iterator for building the initial GP.

- **Iterator gpEval**
  LHS iterator for sampling on the GP.

- **Iterator gpFinalEval**
  LHS iterator for sampling on the final GP.

- **Model gpModel**
  GP model of response, one approximation per response function.

- int **numRounds**
  the number of rounds of additions of size batchSize to add to the original set of LHS samples

- int **numPtsTotal**
  the total number of points

- int **numEmulEval**
  the number of points evaluated by the GP each iteration

- int **numFinalEmulEval**
  number of points evaluated on the final GP

- int **scoringMethod**
  the type of scoring metric to use for sampling

- Real **finalProb**
  the final calculated probability (p)

- RealVectorArray **gpCvars**
Vector to hold the current values of the current sample inputs on the GP.

- **RealVectorArray gpMeans**
  
  Vector to hold the current values of the current mean estimates for the sample values on the GP.

- **RealVectorArray gpVar**
  
  Vector to hold the current values of the current variance estimates for the sample values on the GP.

- **RealVector emulEvalScores**
  
  Vector to hold the scored values for the current GP samples.

- **RealVector predictionErrors**
  
  Vector to hold the RMSE after each round of adaptively fitting the model.

- **RealVectorArray validationSet**
  
  Validation point set used to determine predictionErrors above.

- **RealVector yTrue**
  
  True function responses at the values corresponding to validationSet.

- **RealVector yModel**
  
  Surrogate function responses at the values corresponding to validationSet.

- **int validationSetSize**
  
  Number of points used in the validationSet.

- **int batchSize**
  
  Number of points to add each round, default = 1.

- **String batchStrategy**
  
  String describing the type of batch addition to use. Allowable values are naive, distance, topology.

- **String outputDir**
  
  Temporary string for dumping validation files used in TopoAS visualization.

- **String scoringMetric**
  
  String describing the method for scoring candidate points. Options are: alm, distance, gradient, highest_persistence, avg_persistence, bottleneck, alm_topo_hybrid. Note: alm and alm_topo_hybrid will fail when used with surrogates other than global_kriging as it is based on the variance of the surrogate. At the time of implementation, global_kriging is the only surrogate capable of yielding this information.

- **unsigned short sampleDesign**
  
  Enum describing the initial sample design. Options are: RANDOM_SAMPLING, FSU_CVT, FSU_HALTON, FSU_HAMMERSLEY

- **String approx_type**
  
  String describing type of surrogate is used to fit the data. Options are: global_kriging, global_mars, global_neural_network, global_polynomial, globabl_moving_least_squares, global_radial_basis.

- **MS_Complex *AMSC**
  
  The approximate Morse-Smale complex data structure.

- **int numKneighbors**
  
  The number of approximate nearest neighbors to use in computing the AMSC.

- **bool outputValidationData**
  
  Temporary variable for toggling writing of data files to be used by TopoAS.
Additional Inherited Members

13.77.1 Detailed Description

Class for testing various Adaptively sampling methods using geometric, statistical, and topological information of the surrogate.

NonDAdaptiveSampling implements an adaptive sampling method based on the work presented in Adaptive Sampling with Topological Scores by Dan Maljovec, Bei Wang, Ana Kupresanin, Gardar Johannesson, Valerio Pascucci, and Peer-Timo Bremer presented in IJUQ (insert issue). The method computes scores based on the topology of the known data and the topology of the surrogate model. A number of alternate adaption strategies are offered as well.

13.77.2 Constructor & Destructor Documentation

NonDAdaptiveSampling ( ProblemDescDB & problem_db, Model & model )

standard constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification.

References NonDAdaptiveSampling::AMSC, NonDAdaptiveSampling::approx_type, Iterator::assign_rep(), Model::assign_rep(), NonDAdaptiveSampling::batchSize, NonDAdaptiveSampling::batchStrategy, NonDAdaptiveSampling::construct_fsu_sampler(), NonD::construct_lhs(), ProblemDescDB::get_bool(), ProblemDescDB::get_int(), ProblemDescDB::get_sa(), ProblemDescDB::get_string(), NonDAdaptiveSampling::gpBuild, NonDAdaptiveSampling::gpEval, NonDAdaptiveSampling::gpFinalEval, NonDAdaptiveSampling::gpModel, Model::gradient_type(), Model::hessian_type(), Iterator::iteratedModel, Iterator::maxIterations, NonDAdaptiveSampling::numEmulEval, NonDAdaptiveSampling::numFinalEmulEval, NonDAdaptiveSampling::numKnighbors, NonDAdaptiveSampling::numRounds, NonDSampling::numSamples, NonDAdaptiveSampling::outputDir, Iterator::outputLevel, NonDAdaptiveSampling::outputValidationData, NonDAdaptiveSampling::parse_options(), Iterator::probDescDB, NonDSampling::randomSeed, NonDSampling::rngName, NonDAdaptiveSampling::sampleDesign, NonDAdaptiveSampling::scoringMetric, NonDSampling::vary_pattern(), and NonDSampling::varyPattern.

~NonDAdaptiveSampling ()

alternate constructor for sample generation and evaluation "on the fly" has not been implemented
destructor

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

- NonDAdaptiveSampling.hpp
- NonDAdaptiveSampling.cpp

13.78 NonDBayesCalibration Class Reference

Base class for Bayesian inference: generates posterior distribution on model parameters given experimental data.

Inheritance diagram for NonDBayesCalibration:
Public Member Functions

- `NonDBayesCalibration` (ProblemDescDB &problem_db, Model &model)
  *standard constructor*
- `~NonDBayesCalibration` ()
  *destructor*

Protected Member Functions

- `void init_communicators` ()
  *initialize the communicators associated with iteratedModel as well as any subordinate iterators or subordinate models*
- `void free_communicators` ()
  *free the communicators associated with iteratedModel as well as any subordinate iterators or subordinate models*
- `void quantify_uncertainty` ()
  *Mapping of the core run() virtual function for the NonD branch.*
- `const Model & algorithm_space_model` () const

Protected Attributes

- `Model emulatorModel`
  *Model instance employed in the likelihood function; provides response function values from Gaussian processes, stochastic expansions (PCE/SC), or direct access to simulations (no surrogate option)*
- `bool standardizedSpace`
  *flag indicating use of a variable transformation to standardized probability space*
- `Iterator stochExpIterator`
  *NonDPolynomialChaos or NonDStochCollocation instance for defining a PCE/SC-based emulatorModel.*
- `Iterator lhsIterator`
  *LHS iterator for generating samples for GP.*
CHAPTER 13. CLASS DOCUMENTATION

Private Attributes

- short emulatorType

the emulator type: NO_EMULATOR, GP_EMULATOR, PCE_EMULATOR, or SC_EMULATOR

Additional Inherited Members

13.78.1 Detailed Description

Base class for Bayesian inference: generates posterior distribution on model parameters given experimental data.

This class will eventually provide a general-purpose framework for Bayesian inference. In the short term, it only collects shared code between QUESO and GPMSA implementations.

13.78.2 Constructor & Destructor Documentation

NonDBayesCalibration ( ProblemDescDB & problem_db, Model & model )

standard constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification.

References Iterator::algorithm_space_model(), Iterator::assign_rep(), Model::assign_rep(), NonD::cdfFlag, NonDBayesCalibration::emulatorModel, NonDBayesCalibration::emulatorType, ProblemDescDB::get_bool(), ProblemDescDB::get_int(), ProblemDescDB::get_short(), ProblemDescDB::get_string(), ProblemDescDB::get_usa(), Model::gradient_type(), Model::hessian_type(), NonD::initialize_random_variable_correlations(), NonD::initialize_random_variable_transformation(), NonD::initialize_random_variable_types(), Iterator::iteratedModel, Iterator::iterator_rep(), NonDBayesCalibration::lhsIterator, Iterator::maxEvalConcurrency, Iterator::outputLevel, Iterator::probDescDB, NonD::requested_levels(), NonD::respLevelTarget, NonD::respLevelTargetReduce, NonDBayesCalibration::standardizedSpace, NonDBayesCalibration::stochExpIterator, NonD::transform_model(), and NonD::verify_correlation_support().

13.78.3 Member Function Documentation

const Model & algorithm_space_model ( ) const [inline], [protected], [virtual]

default definition that gets redefined in selected derived Minimizers

Reimplemented from Analyzer.

References NonDBayesCalibration::emulatorModel.

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

- NonDBayesCalibration.hpp
- NonDBayesCalibration.cpp

13.79 NonDCalibration Class Reference

Inheritance diagram for NonDCalibration:
Public Member Functions

- **NonDCalibration** (ProblemDescDB &problem_db, Model &model)
  
  *standard constructor*

- **~NonDCalibration** ()
  
  *destructor*

Protected Member Functions

- void set_configuration_vars (Model &model, const RealVector &x)
  
  *set the passed configuration variables into the model’s state vars*

Protected Attributes

- RealVector **expStdDeviations**
  
  *1 or numFunctions standard deviations*

- String **expDataFileName**
  
  *filename from which to read experimental data; optionally configuration vars x and standard deviations sigma*

- bool **expDataFileAnnotated**
  
  *whether the data file is in annotated format*

- size_t **numExperiments**
  
  *number of experiments to read from data file*

- IntVector **numReplicates**
  
  *number of replicates per experiment*

- size_t **numExpConfigVars**
  
  *number of columns in data file which are state variables*

- size_t **numExpStdDeviationsRead**
  
  *how many sigmas to read from the data file (1 or numFunctions)*

- ExperimentData **expData**
  
  *Container for experimental data to which to calibrate model.*
Private Member Functions

- `bool find_state_index (unsigned short state_type, UShortMultiArrayConstView variable_types, std::string context_message, size_t &start_index)`
  
  helper function to lookup a state_type enum variable type in the array of variables_types to find its start_index into the all array

Private Attributes

- `size_t continuousConfigVars`  
  number of continuous configuration variables
- `size_t discreteIntConfigVars`  
  number of discrete integer configuration variables
- `size_t discreteRealConfigVars`  
  number of discrete real configuration variables
- `size_t continuousConfigStart`  
  index of configuration variables in all continuous array
- `size_t discreteIntConfigStart`  
  index of configuration variables in all discrete integer array
- `size_t discreteRealConfigStart`  
  index of configuration variables in all discrete real array

Additional Inherited Members

13.79.1 Detailed Description

This class ...

13.79.2 Constructor & Destructor Documentation

NonDCalibration ( ProblemDescDB & problem_db, Model & model )

standard constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification.

References Dakota::abort_handler(), Model::all_continuous_variable_types(), Model::all_discrete_int_variable_types(), Model::all_discrete_real_variable_types(), NonDCalibration::continuousConfigStart, NonDCalibration::continuousConfigVars, NonDCalibration::discreteIntConfigStart, NonDCalibration::discreteIntConfigVars, NonDCalibration::discreteRealConfigStart, NonDCalibration::discreteRealConfigVars, NonDCalibration::expDataFileName, NonDCalibration::expStdDeviations, NonDCalibration::find_state_index(), ProblemDescDB::get_sizet(), Iterator::iteratedModel, NonDCalibration::numExpConfigVars, NonDCalibration::numExperiments, Analyzer::numFunctions, NonDCalibration::numReplicates, and Iterator::probDescDB.

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

- NonDCalibration.hpp
- NonDCalibration.cpp
13.80  NonDCubature Class Reference

Derived nondeterministic class that generates N-dimensional numerical cubature points for evaluation of expectation integrals.

Inheritance diagram for NonDCubature:

```
NonDCubature
   ^
   |  Iterator
   |  Analyzer
   |  NonD
   |  NonDIntegration
```

Public Member Functions

- **NonDCubature** (Model &model, const Pecos::ShortArray &u_types, unsigned short cub.int_order)
- unsigned short **integrand_order** () const
  
    return cubIntOrder

Protected Member Functions

- **NonDCubature** (ProblemDescDB &problem_db, Model &model)
  
    constructor
- **~NonDCubature** ()
  
    destructor
- void **initialize_grid** (const std::vector<Pecos::BasisPolynomial> &poly_basis)
  
    initialize integration grid by drawing from polynomial basis settings
- void **get_parameter_sets** (Model &model)
  
    Returns one block of samples (ndim * num_samples)
- void **sampling_reset** (int min_samples, bool all_data_flag, bool stats_flag)
- void **increment_grid** ()
  
    increment SSG level/TPQ order
- void **increment_grid_preference** (const RealVector &dim_pref)
- int **num_samples** () const

Private Member Functions

- void **check_integration** (const Pecos::ShortArray &u_types, const Pecos::AleatoryDistParams &adp)
  
    verify self-consistency of integration specification
- void **increment_reference** ()
  
    increment each cubIntOrderRef entry by 1
Private Attributes

- Pecos::CubatureDriver * cubDriver
  
  *convenience pointer to the numIntDriver representation*

- unsigned short cubIntOrderRef
  
  *reference point for Pecos::CubatureDriver::cubIntOrder: the original user specification for the number of Gauss points per dimension, plus any refinements posted by increment_grid()*

- unsigned short cubIntRule
  
  *the isotropic cubature integration rule*

Additional Inherited Members

### 13.80.1 Detailed Description

Derived nondeterministic class that generates N-dimensional numerical cubature points for evaluation of expectation integrals.

This class is used by NonDPolynomialChaos, but could also be used for general numerical integration of moments. It employs Stroud cubature rules and extensions by D. Xiu.

### 13.80.2 Constructor & Destructor Documentation

**NonDCubature ( Model & model, const Pecos::ShortArray & u_types, unsigned short cub_int_order )**

This alternate constructor is used for on-the-fly generation and evaluation of numerical cubature points.

References Model::aleatory_distribution_parameters(), NonDCubature::check_integration(), NonDCubature::cubDriver, NonDCubature::cubIntOrderRef, Iterator::iteratedModel, and NonDIntegration::numIntDriver.

**NonDCubature ( ProblemDescDB & problem_db, Model & model ) [protected]**

*constructor*

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification. It is not currently used, as there is not yet a separate nond_cubature method specification.

References Model::aleatory_distribution_parameters(), NonDCubature::check_integration(), NonDIntegration::check_variables(), NonDCubature::cubDriver, NonDCubature::cubIntOrderRef, NonDCubature::cubIntRule, Iterator::iteratedModel, Iterator::maxEvalConcurrency, NonD::natafTransform, and NonDIntegration::numIntDriver.

### 13.80.3 Member Function Documentation

**void sampling_reset ( int min_samples, bool all_data_flag, bool stats_flag ) [protected], [virtual]**

used by DataFitSurrModel::build_global() to publish the minimum number of points needed from the cubature routine in order to build a particular global approximation.

Reimplemented from Iterator.

References NonDCubature::cubDriver, and NonDCubature::cubIntOrderRef.
void increment_grid_preference ( const RealVector & dim pref ) [inline], [protected], [virtual]

Should not be used, but pure virtual must be defined.
    Reimplemented from NonDIntegration.
    References NonDCubature::increment_grid().

int num_samples ( ) const [inline], [protected], [virtual]

Return current number of evaluation points. Since the calculation of samples, collocation points, etc. might be costly, provide a default implementation here that backs out from the maxEvalConcurrency.
    Reimplemented from Analyzer.
    References NonDCubature::cubDriver.

void increment_reference ( ) [inline], [private]

increment each cubIntOrderRef entry by 1
    cubIntOrderRef is a reference point for CubatureDriver::cubIntOrder, e.g., a lower bound
    References NonDCubature::cubIntOrderRef.
    Referenced by NonDCubature::increment_grid().
    The documentation for this class was generated from the following files:
    • NonDCubature.hpp
    • NonDCubature.cpp

13.81 NonDDREAMBayesCalibration Class Reference

Bayesian inference using the DREAM approach.
    Inheritance diagram for NonDDREAMBayesCalibration:

    NonDDREAMBayesCalibration
    NonDBayesCalibration
    NonDCalibration
    NonD
    Analyzer
    Iterator

Public Member Functions
    • NonDDREAMBayesCalibration (ProblemDescDB & problem_db, Model & model)
        standard constructor
• `~NonDDREAMBayesCalibration()`
  destructor

### Static Public Member Functions

- static void `problem_size` (int &chain_num, int &cr_num, int &gen_num, int &pair_num, int &par_num)
  initializer for problem size characteristics in DREAM
- static void `problem_value` (std::string *chain_filename, std::string *gr_filename, double &gr_threshold, int &jumpstep, double limits[] , int par_num, int &printstep, std::string *restart_read_filename, std::string *restart_write_filename)
  Filename and data initializer for DREAM.
- static double `prior_density` (int par_num, double zp[])
  Compute the prior density at specified point zp.
- static double * `prior_sample` (int par_num)
  Sample the prior and return an array of parameter values.
- static double `sample_likelihood` (int par_num, double zp[])
  Likelihood function for call-back from DREAM to DAKOTA for evaluation.

### Protected Member Functions

- void `quantify_uncertainty` ()
  redefined from DakotaNonD

### Protected Attributes

- Real `likelihoodScale`
  scale factor for proposal covariance
- int `numSamples`
  number of samples in the chain (e.g. number of MCMC samples)
- bool `calibrateSigmaFlag`
  flag to indicate if the sigma terms should be calibrated (default true)
- int `randomSeed`
  random seed to pass to QUESO
- RealVector `paramMins`
  lower bounds on calibrated parameters
- RealVector `paramMaxs`
  upper bounds on calibrated parameters
- int `numChains`
  number of concurrent chains
- int `numGenerations`
  number of generations
- int `numCR`
  number of CR-factors
- int `crossoverChainPairs`
  number of crossover chain pairs
NonDDREAMBayesCalibration Class Reference

Real grThreshold
threshold for the Gelman-Rubin statistic

int jumpStep
how often to perform a long jump in generations

std::vector< boost::math::uniform > priorDistributions
uniform prior PDFs for each variable

boost::mt19937 numGenerator
random number engine for sampling the prior

std::vector< boost::uniform_real< double > > priorSamplers
samplers for the uniform prior PDFs for each variable

Private Attributes

short emulatorType
the emulator type: NO_EMULATOR, GP_EMULATOR, PCE_EMULATOR, or SC_EMULATOR

Static Private Attributes

static NonDDREAMBayesCalibration * NonDDREAMInstance
Pointer to current class instance for use in static callback functions.

Additional Inherited Members

13.81.1 Detailed Description

Bayesian inference using the DREAM approach.
This class performed Bayesian calibration using the DREAM (Markov Chain Monte Carlo acceleration by Differential Evolution) implementation of John Burkhardt (FSU), adapted from that of Guannan Zhang (ORNL)

13.81.2 Constructor & Destructor Documentation

NonDDREAMBayesCalibration ( ProblemDescDB & problem_db, Model & model )

standard constructor
This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification.
References NonDDREAMBayesCalibration::crossoverChainPairs, NonDDREAMBayesCalibration::grThreshold, NonDDREAMBayesCalibration::jumpStep, NonDDREAMBayesCalibration::numChains, NonDDREAMBayesCalibration::numCR, NonDDREAMBayesCalibration::numGenerations, and NonDDREAMBayesCalibration::numSamples.

13.81.3 Member Function Documentation

void problem_size ( int & chain_num, int & cr_num, int & gen_num, int & pair_num, int & par_num )
[static]
initializer for problem size characteristics in DREAM
See documentation in DREAM examples)
References NonDDREAMBayesCalibration::crossoverChainPairs, NonDDREAMBayesCalibration::NonDDREAMInstance, NonDDREAMBayesCalibration::numChains, Analyzer::numContinuousVars, NonDDREAMBayesCalibration::numCR, and NonDDREAMBayesCalibration::numGenerations.

```cpp
void problem_value ( std::string * chain_filename, std::string * gr_filename, double & gr_threshold, int & jumpstep, double limits[ ], int par_num, int & printstep, std::string * restart_read_filename, std::string * restart_write_filename ) [static]
```

Filename and data initializer for DREAM. See documentation in DREAM examples) References NonDDREAMBayesCalibration::grThreshold, NonDDREAMBayesCalibration::jumpStep, NonDDREAMBayesCalibration::NonDDREAMInstance, NonDDREAMBayesCalibration::numChains, NonDDREAMBayesCalibration::paramMaxs, and NonDDREAMBayesCalibration::paramMins.

```cpp
double prior_density ( int par_num, double zp[ ] ) [static]
```

Compute the prior density at specified point zp. See documentation in DREAM examples) References NonDDREAMBayesCalibration::NonDDREAMInstance, and NonDDREAMBayesCalibration::priorDistributions.

```cpp
double * prior_sample ( int par_num ) [static]
```

Sample the prior and return an array of parameter values. See documentation in DREAM examples) References NonDDREAMBayesCalibration::NonDDREAMInstance, NonDDREAMBayesCalibration::priorSamplers, and NonDDREAMBayesCalibration::numGenerator.

```cpp
double sample_likelihood ( int par_num, double zp[ ] ) [static]
```

Likelihood function for call-back from DREAM to DAKOTA for evaluation. Static callback function to evaluate the likelihood References NonDDREAMBayesCalibration::calibrateSigmaFlag, Model::compute_response(), Model::continuous_variables(), Model::current_response(), NonDBayesCalibration::emulatorModel, NonDDREAMBayesCalibration::emulatorType, NonDCalibration::expData, Response::function_values(), NonDDREAMBayesCalibration::likelihoodScale, NonDDREAMBayesCalibration::NonDDREAMInstance, Analyzer::numContinuousVars, NonDCalibration::numExperiments, Analyzer::numFunctions, NonDCalibration::numReplicates, Iterator::outputLevel, ExperimentData::scalar_data(), and ExperimentData::scalar_sigma().

```cpp
void quantify_uncertainty ( ) [protected], [virtual]
```

redefined from DakotaNonD Perform the uncertainty quantification Reimplemented from NonDBayesCalibration.
Level, NonDDREAMBayesCalibration::paramMaxs, NonDDREAMBayesCalibration::paramMins, NonDDREAMBayesCalibration::priorDistributions, NonDDREAMBayesCalibration::priorSamplers, NonDBayesCalibration::quantify_uncertainty(), NonDDREAMBayesCalibration::randomSeed, NonDDREAMBayesCalibration::numGenerator, ExperimentData::scalar_sigma(), NonDBayesCalibration::standardizedSpace, and NonDBayesCalibration::stochExpIterator.

13.81.4 Member Data Documentation

Real likelihoodScale  [protected]

scale factor for proposal covariance
scale factor for likelihood
Referenced by NonDDREAMBayesCalibration::sample_likelihood().
The documentation for this class was generated from the following files:

- NonDDREAMBayesCalibration.hpp
- NonDDREAMBayesCalibration.cpp

13.82 NonDExpansion Class Reference

Base class for polynomial chaos expansions (PCE) and stochastic collocation (SC)

Inheritance diagram for NonDExpansion:

```
NonDExpansion
  |                     
  v                     
NonD
  |                     
  v                     
NonDExpansion
  |                     
  v                     
NonDPolynomialChaos | NonDStochCollocation
```

Public Member Functions

- **NonDExpansion** (ProblemDescDB &problem_db, Model &model)
  - standard constructor
- **NonDExpansion** (unsigned short method_name, Model &model, short exp_coeffs_approach, short u_space_type, bool piecewise_basis, bool use_derivs)
  - alternate constructor
- **~NonDExpansion** ()
  - destructor
- void init_communicators ()
  - initialize the communicators associated with iteratedModel as well as any subordinate iterators or subordinate models
CHAPTER 13. CLASS DOCUMENTATION

• void free_communicators ()
  free the communicators associated with iteratedModel as well as any subordinate iterators or subordinate models
• void quantify_uncertainty ()
  perform a forward uncertainty propagation using PCE/SC methods
• void print_results (std::ostream &s)
  print the final statistics
• const Model & algorithm_space_model () const

Protected Member Functions

• virtual void resolve_inputs (short &u_space_type, short &data_order)
  perform error checks and mode overrides
• virtual void initialize_u_space_model ()
  initialize uSpaceModel polynomial approximations with PCE/SC data
• virtual void initialize_expansion ()
  initialize random variable definitions and final stats arrays
• virtual void compute_expansion ()
  form the expansion by calling uSpaceModel.build_approximation()
• virtual void increment_order ()
  uniformly increment the expansion order (PCE only)
• virtual void increment_specification_sequence ()
  increment the input specification sequence (PCE only)
• virtual void update_expansion ()
  update an expansion; avoids overhead in compute_expansion()
• virtual void print_coefficients (std::ostream &s)
  print expansion coefficients, as supported by derived instance
• virtual void archive_coefficients ()
  archive expansion coefficients, as supported by derived instance
• virtual Real compute_covariance_metric ()
  compute 2-norm of change in response covariance
• virtual Real compute_final_statistics_metric ()
  compute 2-norm of change in final statistics
• void initialize_response_covariance ()
  set covarianceControl defaults and shape respCovariance
• void update_final_statistics ()
  update function values within finalStatistics
• void update_final_statistics_gradients ()
  update function gradients within finalStatistics
• void initialize (short u_space_type)
  common constructor code for initialization of natafTransform
• void refine_expansion ()
  refine the reference expansion found by compute_expansion() using uniform/adaptive p-/h-refinement strategies
• void construct_cubature (Iterator &u_space_sampler, Model &g_u_model, unsigned short cub_int_order)
assign a NonDCubature instance within u_space_sampler

• void construct_quadrature (Iterator &u_space_sampler, Model &g_u_model, const UShortArray &quad_order_seq, const RealVector &dim_pref)
  assign a NonDQuadrature instance within u_space_sampler based on a quad_order specification

• void construct_quadrature (Iterator &u_space_sampler, Model &g_u_model, int filtered_samples, const RealVector &dim_pref)
  assign a NonDQuadrature instance within u_space_sampler that generates a filtered tensor product sample set

• void construct_quadrature (Iterator &u_space_sampler, Model &g_u_model, int random_samples, int seed, const UShortArray &quad_order_seq, const RealVector &dim_pref)
  assign a NonDQuadrature instance within u_space_sampler that samples randomly from a tensor product multi-index

• void construct_sparse_grid (Iterator &u_space_sampler, Model &g_u_model, const UShortArray &ssg_level_seq, const RealVector &ssg_dim_pref)
  assign a NonDSparseGrid instance within u_space_sampler

• void construct_expansion_sampler ()
  construct the expansionSampler operating on uSpaceModel

• void compute_statistics ()
  calculate analytic and numerical statistics from the expansion

• void archive_moments ()
  archive the central moments (numerical and expansion) to ResultsDB

Protected Attributes

• Model uSpaceModel
  Model representing the approximate response function in u-space, after u-space recasting and orthogonal polynomial data fit recursions.

• short expansionCoeffsApproach
  method for collocation point generation and subsequent calculation of the expansion coefficients

• short expansionBasisType
  type of expansion basis: DEFAULT BASIS or Pecos::{NODAL,HIERARCHICAL} INTERPOLANT for SC or Pecos:::{TENSOR_PRODUCT,TOTAL_ORDER,ADAPTED} BASIS for PCE regression

• size_t numUncertainQuant
  number of invocations of quantify_uncertainty()

• int numSamplesOnModel
  number of truth samples performed on g_u_model to form the expansion

• int numSamplesOnExpansion
  number of approximation samples performed on the polynomial expansion in order to estimate probabilities

• bool nestedRules
  flag for indicating state of nested and non_nested overrides of default rule nesting, which depends on the type of integration driver

• bool piecewiseBasis
  flag for piecewise specification, indicating usage of local basis polynomials within the stochastic expansion

• bool useDerivs
  flag for use_derivatives specification, indicating usage of derivative data (with respect to expansion variables) to enhance the calculation of the stochastic expansion.
• short refineType
  refinement type: NO_REFINEMENT, P_REFINEMENT, or H_REFINEMENT

• short refineControl
  refinement control: NO_CONTROL, UNIFORM_CONTROL, LOCAL_ADAPTIVE_CONTROL, DIMENSION_ADAPTIVE_CONTROL_SOBOL, DIMENSION_ADAPTIVE_CONTROL_DECAY, or DIMENSION_ADAPTIVE_CONTROL_GENERALIZED

• RealSymMatrix respCovariance
  symmetric matrix of analytic response covariance (full response covariance option)

• RealVector respVariance
  vector of response variances (diagonal response covariance option)

• RealVector initialPtU
  stores the initial variables data in u-space

Private Member Functions

• void reduce_total sobol sets (RealVector &avg sobol)
  compute average of total Sobol’ indices (from VBD) across the response set for use as an anisotropy indicator

• void reduce_decay_rate_sets (RealVector &min decay)
  compute minimum of spectral coefficient decay rates across the response set for use as an anisotropy indicator

• void initialize_sets ()
  initialization of adaptive refinement using generalized sparse grids

• Real increment_sets ()
  perform an adaptive refinement increment using generalized sparse grids

• void finalize_sets (bool converged_within_tol)
  finalization of adaptive refinement using generalized sparse grids

• void compute_covariance ()
  calculate the response covariance (diagonal or full matrix)

• void compute_diagonal_variance ()
  calculate respVariance or diagonal terms respCovariance(i,i)

• void compute_off_diagonal_covariance ()
  calculate respCovariance(i,j) for j<i

• void print_moments (std::ostream &s)
  print expansion and numerical moments

• void print_covariance (std::ostream &s)
  print respCovariance

• void print_sobol_indices (std::ostream &s)
  print global sensitivity indices

• void print_local_sensitivity (std::ostream &s)
  print local sensitivities evaluated at initialPtU

• void compute_print_increment_results ()
  manage print of results following a refinement increment

• void compute_print_iteration_results (bool initialize)
  manage print of results following a refinement increment

• void compute_print_converged_results (bool print_override=false)
  manage print of results following convergence of iterative refinement
Private Attributes

- short ruleNestingOverride
  
  *user override of default rule nesting: NO_NESTING_OVERRIDE, NESTED, or NON_NESTED*

- short ruleGrowthOverride
  
  *user override of default rule growth: NO_GROWTH_OVERRIDE, RESTRICTED, or UNRESTRICTED*

- Iterator expansionSampler
  
  *Iterator used for sampling on the uSpaceModel to generate approximate probability/reliability/response level statistics. Currently this is an LHS sampling instance, but AIS could also be used.*

- Iterator importanceSampler
  
  *Iterator used to refine the approximate probability estimates generated by the expansionSampler using importance sampling.*

- bool expSampling
  
  *flag to indicate calculation of numerical statistics by sampling on the expansion*

- bool impSampling
  
  *flag to use LHS sampling or MMAIS sampling on the expansion*

- RealMatrix expGradsMeanX
  
  *derivative of the expansion with respect to the x-space variables evaluated at the means (used as uncertainty importance metrics)*

- bool vbdFlag
  
  *flag indicating the activation of variance-based decomposition for computing Sobol’ indices*

- unsigned short vbdOrderLimit
  
  *limits the order of interactions within the component Sobol’ indices*

- Real vbdDropTol
  
  *tolerance for omitting output of small VBD indices*

- short covarianceControl
  
  *enumeration for controlling response covariance calculation and output: {DEFAULT,DIAGONAL,FULL}_COVARIANCE*

Additional Inherited Members

13.82.1 Detailed Description

Base class for polynomial chaos expansions (PCE) and stochastic collocation (SC)

The NonDExpansion class provides a base class for methods that use polynomial expansions to approximate the effect of parameter uncertainties on response functions of interest.

13.82.2 Member Function Documentation

**const Model & algorithm_space_model ( ) const**

*default definition that gets redefined in selected derived Minimizers*

Reimplemented from Analyzer.

References NonDExpansion::uSpaceModel.
void increment_specification_sequence() [protected], [virtual]
increment the input specification sequence (PCE only)
default implementation is overridden by PCE
Reimplemented in NonDPolynomialChaos.
References NonDIntegration::increment_specification_sequence(), Iterator::iterator_rep(), Model::subordinate_iterator(), and NonDExpansion::uSpaceModel.
Referenced by NonDPolynomialChaos::increment_specification_sequence(), and NonDExpansion::quantify_uncertainty().

Real compute_covariance_metric() [protected], [virtual]
compute 2-norm of change in response covariance
computes the default refinement metric based on change in respCovariance
Reimplemented in NonDStochCollocation.
References NonDExpansion::compute_covariance(), NonDExpansion::covarianceControl, NonDExpansion::respCovariance, and NonDExpansion::respVariance.
Referenced by NonDStochCollocation::compute_covariance_metric(), NonDExpansion::increment_sets(), and NonDExpansion::refine_expansion().

Real compute_final_statistics_metric() [protected], [virtual]
compute 2-norm of change in final statistics
computes a "goal-oriented" refinement metric employing finalStatistics
Reimplemented in NonDStochCollocation.
References NonDExpansion::compute_statistics(), NonD::finalStatistics, Response::function_values(), Analyzer::numFunctions, NonD::requestedGenRelLevels, NonD::requestedProbLevels, NonD::requestedRelLevels, and NonD::requestedRespLevels.
Referenced by NonDStochCollocation::compute_final_statistics_metric(), and NonDExpansion::increment_sets().

void compute_statistics() [protected]
calculate analytic and numerical statistics from the expansion
Calculate analytic and numerical statistics from the expansion and log results within final_stats for use in OUU.
References Dakota::abort_handler(), ResultsManager::active(), Iterator::active_set(), Response::active_set_derivative_vector(), Response::active_set_request_vector(), Iterator::all_responses(), Iterator::all_samples(), Model::approximation_data(), Model::approximations(), NonD::archive_allocate_mappings(), NonDExpansion::archive_coefficients(), NonD::archive_from_resp(), NonDExpansion::archive_moments(), NonD::archive_to_resp(), NonD::cdfFlag, PecosApproximation::compute_component_effects(), PecosApproximation::compute_moments(), NonDExpansion::compute_off_diagonal_covariance(), PecosApproximation::compute_total_effects(), NonD::computedGenRelLevels, NonD::computedProbLevels, NonD::computedRespLevels, Model::continuous_variable_ids(), Model::continuous_variable_labels(), Model::continuous_variables(), Dakota::copy_data(), NonDExpansion::covarianceControl, Model::current_labels(), PecosApproximation::expansion_coefficient_flag(), NonDExpansion::expansionSampler, NonDExpansion::expGradsMeanX, NonDExpansion::expSampling, NonDAadaptImpSampling::final_probability(), NonD::finalStatistics, Response::function_gradient(), Response::function_value(), Response::function_values(), NonDExpansion::importanceSampler, NonDExpansion::impSampling, Iterator::initial_points(), NonDAadaptImpSampling::initialize(), NonD::initialize_distribution_mappings(), NonD::initialize_random_variables(),
13.83. NONDGLOBALEVIDENCE CLASS REFERENCE

NonDExpansion::initialPtU, ResultsManager::insert(), Iterator::iteratedModel, Iterator::iteratorep(), PecosApproximation::mean_gradient(), PecosApproximation::moment(), NonD::natafTransform, NonD::numContDesVars, NonD::numContEpistUncVars, Analyzer::numContinuousVars, Analyzer::numFunctions, NonDExpansion::numSamplesOnExpansion, Iterator::outputLevel, ActiveSet::request_vector(), NonD::requestedGenRelLevels, NonD::requestedProbLevels, NonD::requestedRelLevels, NonD::requestedRespLevels, NonDExpansion::respCovariance, NonD::respLevelTarget, Model::response_labels(), Iterator::response_results(), NonDExpansion::respVariance, Iterator::resultsDB, Iterator::resultsNames, Iterator::run(), Iterator::run_identifier(), Iterator::subIteratorFlag, NonD::totalLevelRequests, NonDSampling::update_final_statistics(), NonDExpansion::uSpaceModel, PecosApproximation::variance_gradient(), and NonDExpansion::vbdFlag.

Referenced by NonDStochCollocation::compute_final_statistics_metric(), NonDExpansion::compute_final_statistics_metric(), NonDExpansion::compute_print_converged_results(), NonDExpansion::compute_print_increment_results(), and NonDExpansion::compute_print_iteration_results().

13.82.3 Member Data Documentation

**bool useDerivs** [protected]

flag for use_derivatives specification, indicating usage of derivative data (with respect to expansion variables) to enhance the calculation of the stochastic expansion.

This is part of the method specification since the instantiation of the global data fit surrogate is implicit with no user specification. This behavior is distinct from the usage of response derivatives with respect to auxiliary variables (design, epistemic) for computing derivatives of aleatory expansion statistics with respect to these variables.

Referenced by NonDExpansion::compute_expansion(), NonDStochCollocation::initialize_u_space_model(), NonDPolynomialChaos::initialize_u_space_model(), NonDStochCollocation::resolve_inputs(), NonDPolynomialChaos::resolve_inputs(), NonDPolynomialChaos::terms_ratio_to_samples(), and NonDPolynomialChaos::terms_samples_to_ratio().

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

- NonDExpansion.hpp
- NonDExpansion.cpp

13.83 NonDGlobalEvidence Class Reference

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

Inheritance diagram for NonDGlobalEvidence:
Public Member Functions

- **NonDGlobalEvidence** (ProblemDescDB &problem_db, Model &model)
  
  *constructor*

- ~**NonDGlobalEvidence** ()
  
  *destructor*

- void initialize ()
  
  *perform any required initialization*

- void set_cell_bounds ()
  
  *set the optimization variable bounds for each cell*

- void get_best_sample (bool maximize, bool eval_approx)
  
  *determine truthFnStar and approxFnStar*

- void post_process_cell_results (bool maximize)
  
  *post-process a cell minimization/maximization result*

- void post_process_response_fn_results ()
  
  *post-process the interval computed for a response function*

- void post_process_final_results ()
  
  *perform final post-processing*

Additional Inherited Members

13.83.1 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:
13.84 NonDGlobalInterval Class Reference

Class for using global nongradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification.

Inheritance diagram for NonDGlobalInterval:

```
NonDGlobalInterval
  NonDInterval
    NonD
      Iterator
      NonDGlobalInterval
      NonDGlobalEvidence
      NonDGlobalSingleInterval
```

**Public Member Functions**

- **NonDGlobalInterval (ProblemDescDB &problem_db, Model &model)** 
  constructor
- **~NonDGlobalInterval ()**
  destructor
- **void init_communicators ()**
  *initialize the communicators associated with iteratedModel as well as any subordinate iterators or subordinate models*
- **void free_communicators ()**
  *free the communicators associated with iteratedModel as well as any subordinate iterators or subordinate models*
- **void quantify_uncertainty ()**
  *Performs an optimization to determine interval bounds for an entire function or interval bounds on a particular statistical estimator.*
- **const Model & algorithm_space_model () const**

**Protected Member Functions**

- **virtual void initialize ()**
  *perform any required initialization*
- **virtual void set_cell_bounds ()**
  *set the optimization variable bounds for each cell*
• virtual void get_best_sample (bool maximize, bool eval_approx)
  determine truthFnStar and approxFnStar
• virtual void post_process_cell_results (bool maximize)
  post-process a cell minimization/maximization result
• virtual void post_process_response_fn_results ()
  post-process the interval computed for a response function
• virtual void post_process_final_results ()
  perform final post-processing
• void post_process_run_results (bool maximize)
  post-process an optimization execution: output results, update convergence controls, and update GP approximation
• void evaluate_response_star_truth ()
  evaluate the truth response at the optimal variables solution and update the GP with the new data

Protected Attributes

• Iterator daceIterator
  LHS iterator for constructing initial GP for all response functions.
• Model fHatModel
  GP model of response, one approximation per response function.
• Iterator intervalOptimizer
  optimizer for solving surrogate-based subproblem: NCSU DIRECT optimizer for maximizing expected improvement or mixed EA if discrete variables.
• Model intervalOptModel
  recast model which formulates the surrogate-based optimization subproblem (recasts as design problem; may assimilate mean and variance to enable max(expected improvement))
• Real approxFnStar
  approximate response corresponding to minimum/maximum truth response
• Real truthFnStar
  minimum/maximum truth response function value

Static Private Member Functions

• static void EIF_objective_min (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  static function used as the objective function in the Expected Improvement Function (EIF) for minimizing the GP
• static void EIF_objective_max (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  static function used as the objective function in the Expected Improvement Function (EIF) for maximizing the GP
• static void extract_objective (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  static function used to extract the active objective function when optimizing for an interval lower or upper bound (non-EIF formulations). The sense of the optimization is set separately.
**Private Attributes**

- `const int seedSpec`
  - the user seed specification (default is 0)
- `int numSamples`
  - the number of samples used in the surrogate
- `String rngName`
  - name of the random number generator
- `bool gpModelFlag`
  - flag indicating use of GP surrogate emulation
- `bool eifFlag`
  - flag indicating use of maximized expected improvement for GP iterate selection
- `unsigned short improvementConvergeCntr`
  - counter for number of successive iterations that the iteration improvement is less than the convergenceTol
- `unsigned short improvementConvergeLimit`
  - counter for number of successive iterations that the iteration improvement is less than the convergenceTol
- `Real distanceTol`
  - tolerance for $L_2$ change in optimal solution
- `unsigned short distanceConvergeCntr`
  - counter for number of successive iterations that the $L_2$ change in optimal solution is less than the convergenceTol
- `unsigned short distanceConvergeLimit`
  - counter for number of successive iterations that the $L_2$ change in optimal solution is less than the convergenceTol
- `RealVector prevCVStar`
  - stores previous optimal point for continuous variables; used for assessing convergence
- `IntVector prevDIVStar`
  - stores previous optimal point for discrete integer variables; used for assessing convergence
- `RealVector prevDRVStar`
  - stores previous optimal point for discrete real variables; used for assessing convergence
- `Real prevFnStar`
  - stores previous solution value for assessing convergence
- `size_t sbIterNum`
  - surrogate-based minimization/maximization iteration count
- `bool boundConverged`
  - flag indicating convergence of a minimization or maximization cycle
- `bool allResponsesPerIter`
  - flag for maximal response extraction (all response values obtained on each function call)
- `short dataOrder`
  - order of the data used for surrogate construction, in ActiveSet request vector 3-bit format; user may override responses spec

**Static Private Attributes**

- `static NonDGlobalInterval * nondGIInstance`
  - pointer to the active object instance used within the static evaluator functions in order to avoid the need for static data
Additional Inherited Members

13.84.1 Detailed Description

Class for using global nongradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification.

The NonDGlobalInterval class supports global nongradient-based optimization approaches to determining interval bounds for epistemic UQ. The interval bounds may be on the entire function in the case of pure interval analysis (e.g. intervals on input = intervals on output), or the intervals may be on statistics of an “inner loop” aleatory analysis such as intervals on means, variances, or percentile levels. The preliminary implementation will use a Gaussian process surrogate to determine interval bounds.

13.84.2 Member Function Documentation

const Model & algorithm_space_model ( ) const [inline], [virtual]
default definition that gets redefined in selected derived Minimizers

Reimplemented from Analyzer.
References NonDGlobalInterval::fHatModel.

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

- NonDGlobalInterval.hpp
- NonDGlobalInterval.cpp

13.85 NonDGlobalReliability Class Reference

Class for global reliability methods within DAKOTA/UQ.

Inheritance diagram for NonDGlobalReliability:

```
NonDGlobalReliability
\|-- NonDReliability
\.-- NonD
   \-- Iterator
      \-- Analyzer
         \-- NonD
            \-- NonDGlobalReliability
```

Public Member Functions

- NonDGlobalReliability (ProblemDescDB &problem_db, Model &model)
  constructor
- ~NonDGlobalReliability ()
  destructor
- void init_communicators ()
initialize the communicators associated with iteratedModel as well as any subordinate iterators or subordinate models

- void **free_communicators** ()
  
  free the communicators associated with iteratedModel as well as any subordinate iterators or subordinate models

- void **quantify_uncertainty** ()
  
  performs an uncertainty propagation using analytical reliability methods which solve constrained optimization problems to obtain approximations of the cumulative distribution function of response

- void **print_results** (std::ostream &s)
  
  print the approximate mean, standard deviation, and importance factors when using the mean value method or the CDF/CCDF information when using MPP-search-based reliability methods

**Private Member Functions**

- void **optimize_gaussian_process** ()
  
  construct the GP using EGO/SKO

- void **importance_sampling** ()
  
  perform multimodal adaptive importance sampling on the GP

- void **get_best_sample** ()
  
  determine current best solution from among sample data for expected improvemont function in Performance Measure Approach (PMA)

- Real **constraint_penalty** (const Real &constraint, const RealVector &c_variables)
  
  calculate the penalty to be applied to the PMA constraint value

- Real **expected_improvement** (const RealVector &expected_values, const Variables &recast_vars)
  
  expected improvement function for the GP

- Real **expected_feasibility** (const RealVector &expected_values, const Variables &recast_vars)
  
  expected feasibility function for the GP

**Static Private Member Functions**

- static void **EIF_objective_eval** (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  
  static function used as the objective function in the Expected Improvement (EIF) problem formulation for PMA

- static void **EFF_objective_eval** (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  
  static function used as the objective function in the Expected Feasibility (EFF) problem formulation for RIA

**Private Attributes**

- Real **fnStar**
  
  minimum penalized response from among true function evaluations

- short **meritFunctionType**
  
  type of merit function used to penalize sample data

- Real **lagrangeMult**
  
  Lagrange multiplier for standard Lagrangian merit function.

- Real **augLagrangeMult**
  
  Lagrange multiplier for augmented Lagrangian merit function.
- Real **penaltyParameter**
  
  *penalty parameter for augmented Lagrangian merit function*

- Real **lastConstraintViolation**
  
  *constraint violation at last iteration, used to determine if the current iterate should be accepted (must reduce violation)*

- bool **lastIterateAccepted**
  
  *flag to determine if last iterate was accepted this controls update of parameters for augmented Lagrangian merit fn*

- short **dataOrder**
  
  *order of the data used for surrogate construction, in ActiveSet request vector 3-bit format; user may override responses spec*

### Static Private Attributes

- static **NonDGlobalReliability * nondGlobRelInstance**

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

### Additional Inherited Members

#### 13.85.1 Detailed Description

Class for global reliability methods within DAKOTA/UQ.

The **NonDGlobalReliability** class implements EGO/SKO for global MPP search, which maximizes an expected improvement function derived from Gaussian process models. Once the limit state has been characterized, a multimodal importance sampling approach is used to compute probabilities.

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

- **NonDGlobalReliability.hpp**
- **NonDGlobalReliability.cpp**

### 13.86 NonDGlobalSingleInterval Class Reference

Class for using global nongradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification.

Inheritance diagram for NonDGlobalSingleInterval:
Public Member Functions

- **NonDGlobalSingleInterval (ProblemDescDB &problem_db, Model &model)**
  
  *constructor*

- **~NonDGlobalSingleInterval ()**
  
  *destructor*

Protected Member Functions

- **void initialize ()**
  
  *perform any required initialization*

- **void post_process_cell_results (bool maximize)**
  
  *post-process a cell minimization/maximization result*

- **void get_best_sample (bool maximize, bool eval_approx)**
  
  *determine truthFnStar and approxFnStar*

Private Attributes

- **size_t statCntr**
  
  *counter for finalStatistics*

Additional Inherited Members

13.86.1 Detailed Description

Class for using global non-gradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification.

The **NonDGlobalSingleInterval** class supports global non-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 preliminary implementation will use a Gaussian process surrogate to determine interval bounds.

The documentation for this class was generated from the following files:
CHAPTER 13. CLASS DOCUMENTATION

- NonDGlobalSingleInterval.hpp
- NonDGlobalSingleInterval.cpp

13.87 NonDGPImpSampling Class Reference

Class for the Gaussian Process-based Importance Sampling method.

Inheritance diagram for NonDGPImpSampling:

```
  NonDGPImpSampling
  |       |       |
  |       |       |
  |       |       |
  |       |       |
  |       |       |
  |       |       |
  |       |       |
  NonDGPImpSampling
```

Public Member Functions

- **NonDGPImpSampling** (ProblemDescDB &problem_db, Model &model)
  
  standard constructor

- **~NonDGPImpSampling** ()
  
  destructor

- void **init_communicators** ()
  
  initialize the communicators associated with iteratedModel as well as any subordinate iterators or subordinate models

- void **free_communicators** ()
  
  free the communicators associated with iteratedModel as well as any subordinate iterators or subordinate models

- void **quantify_uncertainty** ()
  
  perform the GP importance sampling and return probability of failure.

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

- Real **final_probability** ()
  
  returns the probability calculated by the importance sampling

Private Member Functions

- RealVector **calcExpIndicator** (const int respFnCount, const Real respThresh)
  
  function to calculate the expected indicator probabilities

- Real **calcExpIndPoint** (const int respFnCount, const Real respThresh, const RealVector this_mean, const RealVector this_var)
function to calculate the expected indicator probabilities for one point

- void calcRhoDraw()
  
  function to update the rhoDraw data, adding x values and rho draw values

- RealVector drawNewX(int this_k)
  
  function to pick the next X value to be evaluated by the Iterated model

Private Attributes

- Iterator gpBuild
  
  LHS iterator for building the initial GP.

- Iterator gpEval
  
  LHS iterator for sampling on the GP.

- Model gpModel
  
  GP model of response, one approximation per response function.

- Iterator sampleRhoOne
  
  LHS iterator for sampling from the rhoOneDistribution.

- int numPtsAdd
  
  the number of points added to the original set of LHS samples

- int numPtsTotal
  
  the total number of points

- int numEmulEval
  
  the number of points evaluated by the GP each iteration

- Real finalProb
  
  the final calculated probability (p)

- RealVectorArray gpCvars
  
  Vector to hold the current values of the current sample inputs on the GP.

- RealVectorArray gpMeans
  
  Vector to hold the current values of the current mean estimates for the sample values on the GP.

- RealVectorArray gpVar
  
  Vector to hold the current values of the current variance estimates for the sample values on the GP.

- RealVector expIndicator
  
  Vector to hold the expected indicator values for the current GP samples.

- RealVector rhoDraw
  
  Vector to hold the rhoDraw values for the current GP samples.

- RealVector normConst
  
  Vector to hold the normalization constant calculated for each point added.

- RealVector indicator
  
  IntVector to hold indicator for actual simulation values vs. threshold.

- RealVectorArray xDrawThis
  
  xDrawThis, appended to locally to hold the X values of emulator points chosen

- RealVector expIndThis
  
  expIndThis, appended locally to hold the expected indicator

- RealVector rhoDrawThis
**rhoDrawThis**, appended locally to hold the rhoDraw density for calculating draws

- **RealVector rhoMix**
  
rhoMix, mixture density
- **RealVector rhoOne**
  
rhoOne, original importance density

### Additional Inherited Members

#### 13.87.1 Detailed Description

Class for the Gaussian Process-based Importance Sampling method.

The **NonDGPImpSampling** implements a method developed by Keith Dalbey that uses a Gaussian process surrogate in the calculation of the importance density. Specifically, the mean and variance of the GP prediction are used to calculate an expected value that a particular point fails, and that is used as part of the computation of the "draw distribution." The normalization constants and the mixture distribution used are defined in (need to get SAND report).

#### 13.87.2 Constructor & Destructor Documentation

**NonDGPImpSampling ( ProblemDescDB & problem_db, Model & model )**

standard constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification.

References **Iterator::assign_rep(), Model::assign_rep(), NonD::construct_lhs(), ProblemDescDB::get_bool(), ProblemDescDB::get_int(), ProblemDescDB::get_string(), NonDGPImpSampling::gpBuild, NonDGPImpSampling::gpEval, NonDGPImpSampling::gpModel, Model::gradient_type(), Model::hessian_type(), Iterator::iteratedModel, Iterator::maxIterations, NonDGPImpSampling::numEmulEval, NonDGPImpSampling::numPtsAdd, NonDSampling::numSamples, Iterator::outputLevel, Iterator::probDescDB, NonDSampling::randomSeed, NonDSampling::rngName, NonDGPImpSampling::sampleRhoOne, NonDSampling::samplingVarsMode, NonDSampling::statsFlag, NonDSampling::vary_pattern(), and NonDSampling::varyPattern.

#### 13.87.3 Member Function Documentation

**void quantify_uncertainty ( ) [virtual]**

perform the GP importance sampling and return probability of failure.

Calculate the failure probabilities for specified probability levels using Gaussian process based importance sampling.

Implements **NonD**.

References **Model::acv(), Iterator::all_responses(), Analyzer::all_samples(), Iterator::all_samples(), Model::append_approximation(), Model::approximation_data(), Model::approximation_variances(), Model::build_approximation(), NonDGPImpSampling::calcExpIndicator(), NonDGPImpSampling::calcExpIndPoint(), NonDGPImpSampling::calcRhoDraw(), NonD::cdfFlag, Model::compute_response(), NonD::computedProbLevels, Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), Model::continuous_variables(), Model::current_response(), Model::current_variables(), NonDGPImpSampling::drawNewX(), Model::evaluation_id(), NonDGPImpSampling::expIndicator, NonDGPImpSampling::expIndThis, NonDGPImpSampling::finalProb, Response::function_values(), NonDGPImpSampling::gpCvars, NonDGPImpSampling::gpEval, NonDGPImpSampling::gpMeans, NonDGPImpSampling::gpModel, NonDGPImpSampling::gpVar, NonDGPImpSampling::indicators, NonD::initialize_distribution_mappings(), Iterator::iteratedModel, NonDGPImpSampling::normConst, NonDGPImpSampling::numEmulEval,
Analyzer::numFunctions, NonDGPImpSampling::numPtsAdd, NonDGPImpSampling::numPtsTotal, NonDSampling::numSamples, Iterator::outputLevel, Model::pop_approximation(), NonD::requestedRespLevels, NonDGPImpSampling::rhoDraw, NonDGPImpSampling::rhoDrawThis, NonDGPImpSampling::rhoMix, NonDGPImpSampling::rhoOne, Iterator::run(), NonDGPImpSampling::sampleRhoOne, and NonDGPImpSampling::xDrawThis.

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

- NonDGPImpSampling.hpp
- NonDGPImpSampling.cpp

## 13.88 NonDGPMsABayesCalibration Class Reference

Generates posterior distribution on model parameters given experiment data.

Inheritance diagram for NonDGPMsABayesCalibration:

```
NonDGPMsABayesCalibration
  NonDBayesCalibration
    NonDCalibration
      NonD
        Analyzer
          Iterator
```

### Public Member Functions

- **NonDGPMsABayesCalibration** (ProblemDescDB &problem_db, Model &model)
  
  *constructor*

- **~NonDGPMsABayesCalibration** ()
  
  *destructor*

### Public Attributes

- **rejectionType**
  
  *String*

  Rejection type (standard or delayed, in the DRAM framework)

- **metropolisType**
  
  *String*

  Metropolis type (hastings or adaptive, in the DRAM framework)

- **numSamples**
  
  *int*

  number of samples in the chain (e.g. number of MCMC samples)

- **emulatorSamples**
  
  *int*

  number of samples of the simulation to construct the GP
CHAPTER 13. CLASS DOCUMENTATION

- **RealVector** `proposalCovScale`
  
  scale factor for proposal covariance

- **Real** `likelihoodScale`
  
  scale factor for likelihood

- **bool** `calibrateSigmaFlag`
  
  flag to indicate if the sigma terms should be calibrated (default true)

- **String** `approxImportFile`
  
  name of file from which to import build points to build GP

- **bool** `approxImportAnnotated`
  
  annotate flag

## Protected Member Functions

- **void** `init_communicators()`
  
  initialize the communicators associated with iteratedModel as well as any subordinate iterators or subordinate models

- **void** `free_communicators()`
  
  free the communicators associated with iteratedModel as well as any subordinate iterators or subordinate models

- **void** `quantify_uncertainty()`
  
  performs a forward uncertainty propagation by using GPM/SA to generate a posterior distribution on parameters given a set of simulation parameter/response data, a set of experimental data, and additional variables to be specified here.

## Protected Attributes

- **int** `randomSeed`
  
  print the final statistics

## Private Attributes

- **Iterator** `lhsIter`
  
  LHS iterator for generating samples for GP.

## Static Private Attributes

- static **NonDGPMSABayesCalibration** *NonDGPMSAInstance*
  
  Pointer to current class instance for use in static callback functions.

## Additional Inherited Members

### 13.88.1 Detailed Description

Generates posterior distribution on model parameters given experiment data.

This class provides a wrapper for the functionality provided in the Los Alamos National Laboratory code called GPM/SA (Gaussian Process Models for Simulation Analysis). Although this is a code that provides input/output mapping, it DOES NOT provide the mapping that we usually think of in the NonDeterministic class hierarchy in DAKOTA, where uncertainty in parameter inputs are mapped to uncertainty in simulation responses.
Instead, this class takes a pre-existing set of simulation data as well as experimental data, and maps priors on input parameters to posterior distributions on those input parameters, according to a likelihood function. The goal of the MCMC sampling is to produce posterior values of parameter estimates which will produce simulation response values that “match well” to the experimental data. The MCMC is an integral part of the calibration. The data structures in GPM/SA are fairly detailed and nested. Part of this prototyping exercise is to determine what data structures need to be specified and initialized in DAKOTA and sent to GPM/SA, and what data structures will be returned.

### 13.88.2 Constructor & Destructor Documentation

**NonDGPMSABayesCalibration** (ProblemDescDB & problem_db, 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.

References Iterator::assign_rep(), NonDGPMSABayesCalibration::emulatorSamples, ProblemDescDB::get_string(), Iterator::iteratedModel, NonDGPMSABayesCalibration::lhsIter, Iterator::probDescDB, and NonDGPMSABayesCalibration::randomSeed.

### 13.88.3 Member Function Documentation

void quantify_uncertainty() [protected], [virtual]

performs a forward uncertainty propagation by using GPM/SA to generate a posterior distribution on parameters given a set of simulation parameter/response data, a set of experimental data, and additional variables to be specified here.

Perform the uncertainty quantification

Reimplemented from NonDBayesCalibration.

References Iterator::all_responses(), Analyzer::all_samples(), Iterator::all_samples(), NonDGPMSABayesCalibration::approxImportAnnotated, NonDGPMSABayesCalibration::approxImportFile, NonDGPMSABayesCalibration::calibrateSigmaFlag, ExperimentData::config_vars(), Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), Model::continuous_variables(), NonDGPMSABayesCalibration::emulatorSamples, NonDCalibration::expData, NonDCalibration::expDataFileAnnotated, NonDCalibration::expDataFileName, Iterator::iteratedModel, NonDGPMSABayesCalibration::lhsIter, ExperimentData::load_scalar(), NonDGPMSABayesCalibration::metropolis_Type, NonDGPMSABayesCalibration::NonDGPMSAInstance, NonDCalibration::numExpConfigVars, NonDCalibration::numExpExperiments, NonDCalibration::numExpStdDeviationsRead, Analyzer::numFunctions, NonDCalibration::numReplicates, NonDGPMSABayesCalibration::numSamples, NonD::numUncertainVars, Iterator::outputLevel, NonDGPMSABayesCalibration::rejectionType, Iterator::run(), and ExperimentData::scalar_data().

### 13.88.4 Member Data Documentation

int randomSeed [protected]

print the final statistics

random seed to pass to QUESO

Referenced by NonDGPMSABayesCalibration::NonDGPMSABayesCalibration().

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

- NonDGPMSABayesCalibration.hpp
- NonDGPMSABayesCalibration.cpp
13.89 NonDIncremLHSSampling Class Reference

Performs incremental LHS sampling for uncertainty quantification.

Inheritance diagram for NonDIncremLHSSampling:

```
  NonDIncremLHSSampling
  |                        |
  |    NonDSampling        |
  |    NonD                |
  |    Iterator            |
  |                         |
  Analyzer
```

Public Member Functions

- **NonDIncremLHSSampling (ProblemDescDB &problem_db, Model &model)**
  constructor
- **~NonDIncremLHSSampling ()**
  destructor
- **void quantify_uncertainty ()**
  performs a forward uncertainty propagation by using LHS to generate a set of parameter samples, performing function evaluations on these parameter samples, and computing statistics on the ensemble of results.
- **void print_results (std::ostream &s)**
  print the final statistics

Static Protected Member Functions

- static bool rank_sort (const int &x, const int &y)
  sort algorithm to compute ranks for rank correlations

Private Attributes

- int previousSamples
  number of samples in previous LHS run
- bool varBasedDecompFlag
  flags computation of VBD

Static Private Attributes

- static RealArray rawData
  static data used by static rank_sort() fn
Additional Inherited Members

13.89.1 Detailed Description

Performs incremental LHS sampling for uncertainty quantification.

The Latin Hypercube Sampling (LHS) package from Sandia Albuquerque’s Risk and Reliability organization provides comprehensive capabilities for Monte Carlo and Latin Hypercube sampling within a broad array of user-specified probabilistic parameter distributions. The incremental LHS sampling capability allows one to supplement an initial sample of size n to size 2n while maintaining the correct stratification of the 2n samples and also maintaining the specified correlation structure. The incremental version of LHS will return a sample of size n, which when combined with the original sample of size n, allows one to double the size of the sample.

13.89.2 Constructor & Destructor Documentation

NonDIncremLHSSampling (ProblemDescDB & problem_db, Model & model)

constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification.

13.89.3 Member Function Documentation

void quantify_uncertainty ( ) [virtual]

performs a forward uncertainty propagation by using LHS to generate a set of parameter samples, performing function evaluations on these parameter samples, and computing statistics on the ensemble of results.

Generate incremental samples. Loop over the set of samples and compute responses. Compute statistics on the set of responses if statsFlag is set.

Implements NonD.

References Dakota::abort_handler(), Model::aleatory_distribution_parameters(), Analyzer::allResponses, Analyzer::allSamples, NonDSampling::compute_statistics(), Dakota::copy_data(), Dakota::data_pairs, Analyzer::evaluate_parameters(), NonDSampling::get_parameter_sets(), Iterator::iteratedModel, NonD::numBetaVars, NonD::numBinomialVars, Analyzer::numContinuousVars, NonD::numExponentialVars, NonD::numFrechetVars, NonD::numGammaVars, NonD::numGeometricVars, NonD::numGumbelVars, NonD::numHistogramBinVars, NonD::numHistogramPtVars, NonD::numHyperGeomVars, NonD::numLognormalVars, NonD::numLoguniformVars, NonD::numNegBinomialVars, NonD::numNormalVars, NonD::numPoissonVars, NonDSampling::numSamples, NonD::numTriangularVars, NonD::numUniformVars, NonD::numWeibullVars, NonDIncremLHSSampling::previousSamples, NonDIncremLHSSampling::rank_sort(), NonDIncremLHSSampling::rawData, NonDSampling::sampleRanks, NonDSampling::sampleRanksMode, NonDSampling::samplesRef, NonDSampling::sampleType, Iterator::submethod_enum_to_string(), and NonDSampling::varyPattern.

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

- NonDIncremLHSSampling.hpp
- NonDIncremLHSSampling.cpp

13.90 NonDIntegration Class Reference

Derived nondeterministic class that generates N-dimensional numerical integration points for evaluation of expectation integrals.

Inheritance diagram for NonDIntegration:
Public Member Functions

- virtual void initialize_grid (const std::vector<Pecos::BasisPolynomial> &polybasis)=0
  
  initialize integration grid by drawing from polynomial basis settings
- virtual void increment_grid ()=0
  
  increment SSG level/TPQ order
- virtual void increment_grid_preference (const RealVector &dim_pref)
  
  increment SSG level/TPQ order and update anisotropy
- virtual void increment_grid_weights (const RealVector &aniso_wts)
  
  increment SSG level/TPQ order and update anisotropy
- virtual void increment_specification_sequence ()
  
  increment sequenceIndex and update active orders/levels
- const Pecos::IntegrationDriver & driver () const
  
  return numIntDriver

Static Public Member Functions

- static void dimension_preference_to_anisotropic_order (unsigned short scalar_order_spec, const RealVector &dim_pref_spec, size_t num_v, UShortArray &aniso_order)
  
  convert scalar_order_spec and vector dim_pref_spec to vector aniso_order
- static void anisotropic_order_to_dimension_preference (const UShortArray &aniso_order, unsigned short &scalar_order, RealVector &dim_pref)
  
  convert vector aniso_order to scalar_order and vector dim_pref

Protected Member Functions

- NonDIntegration (ProblemDescDB &problem_db, Model &model)
  
  constructor
- NonDIntegration (unsigned short method_name, Model &model)
  
  alternate constructor for instantiations "on the fly"
- NonDIntegration (unsigned short method_name, Model &model, const RealVector &dim_pref)
  
  alternate constructor for instantiations "on the fly"
13.90. NONDINTEGRATION CLASS REFERENCE

- ~NonDIntegration () 
  destructor
- void quantify_uncertainty ()
  Mapping of the core run() virtual function for the NonD branch.
- void check_variables (const Pecos::ShortArray &x_types)
  verify self-consistency of variables data
- void print_points_weights (const String &tabular_name)
  output integration points and weights to a tabular file

Protected Attributes

- Pecos::IntegrationDriver numIntDriver
  Pecos utility class for managing interface to tensor-product grids and VPI SparseGrid utilities for Smolyak sparse grids and cubature.
- size_t numIntegrations
  counter for number of integration executions for this object
- size_t sequenceIndex
  index into NonDQuadrature::quadOrderSpec and NonDSparseGrid::ssgLevelSpec that defines the current instance of several possible refinement levels
- RealVector dimPrefSpec
  the user specification for anisotropic dimension preference

Additional Inherited Members

13.90.1 Detailed Description

Derived nondeterministic class that generates N-dimensional numerical integration points for evaluation of expectation integrals.

This class provides a base class for shared code among NonDQuadrature and NonDSparseGrid.

13.90.2 Constructor & Destructor Documentation

NonDIntegration ( ProblemDescDB & problem_db, Model & model ) [protected]
constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification. It is not currently used, as there are not yet separate nond_quadrature/nond_sparse_grid method specifications.

References Dakota::abort_handler(), NonD::initialize_final_statistics(), NonD::initialize_random_variable_correlations(), NonD::initialize_random_variable_transformation(), NonD::initialize_random_variable_types(), Analyzer::numDiscrete-IntVars, Analyzer::numDiscreteRealVars, and NonD::verify_correlation_support().

NonDIntegration ( unsigned short method_name, 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.
alternate constructor for instantiations "on the fly"
This alternate constructor is used for on-the-fly generation and evaluation of numerical integration points.

13.90.3 Member Function Documentation

void dimension_preference_to_anisotropic_order ( unsigned short scalar_order_spec, const RealVector & dim_pref_spec, size_t num_v, UShortArray & aniso_order ) [static]
convert scalar_order_spec and vector dim_pref_spec to vector aniso_order
Converts a scalar order specification and a vector anisotropic dimension preference into an anisotropic order vector. It is used for initialization and does not enforce a reference lower bound (see also NonDQuadrate-::update_anisotropic_order()).

Referenced by NonDPolynomialChaos::increment_specification_sequence(), NonDQuadtrature::initialize_dimension_quadrature_order(), and NonDPolynomialChaos::NonDPolynomialChaos().

void anisotropic_order_to_dimension_preference ( const UShortArray & aniso_order, unsigned short & scalar_order, RealVector & dim_pref ) [static]
convert vector aniso_order to scalar_order and vector dim_pref
Converts a vector anisotropic order into a scalar order and vector anisotropic dimension preference.

Referenced by NonDPolynomialChaos::NonDPolynomialChaos().

void check_variables ( const Pecos::ShortArray & x_types ) [protected]
verify self-consistency of variables data
Virtual function called from probDescDB-based constructors and from NonDIntegration::quantify_uncertainty()
References Dakota::abort_handler(), NonD::numContAleatUncVars, NonD::numContDesVars, NonD::numContEpistUncVars, Analyzer::numContinuousVars, and NonD::numContStateVars.

Referenced by NonDCubature::NonDCubature(), NonDQuadtrature::NonDQuadtrature(), NonDSparseGrid:::NonDSparseGrid(), and NonDIntegration::quantify_uncertainty().
The documentation for this class was generated from the following files:

- NonDIntegration.hpp
- NonDIntegration.cpp

13.91 NonDInterval Class Reference

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

Inheritance diagram for NonDInterval:
Public Member Functions

- **NonDInterval** (ProblemDescDB &problem_db, 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 ()`
  - method for computing belief and plausibility values for response levels or vice-versa
- void `calculate_cells_and_bpas ()`
  - computes the interval combinations (cells) and their bpas replaces CBPIIC.F77 from wrapper calculate_basic_prob_intervals()
- void `calculate_cbf_cpf` (bool complementary=true)
  - function to compute (complementary) distribution functions on belief and plausibility replaces CCBFPF.F77 from wrapper calculate_cum_belief_plaus()  

Protected Attributes

- bool `singleIntervalFlag`
  - flag for SingleInterval derived class
- RealVectorArray `ccBelFn`
  - Storage array to hold CCBF values.
- RealVectorArray `ccPlausFn`
  - Storage array to hold CCPF values.
- RealVectorArray `ccBelVal`
  - Storage array to hold CCB response values.
- RealVectorArray `ccPlausVal`
  - Storage array to hold CCP response values.
- RealVectorArray `cellContLowerBounds`
  - Storage array to hold cell lower bounds for continuous variables.
CHAPTER 13. CLASS DOCUMENTATION

- RealVectorArray `cellContUpperBounds`
  Storage array to hold cell upper bounds for continuous variables.
- IntVectorArray `cellIntRangeLowerBounds`
  Storage array to hold cell lower bounds for discrete int range variables.
- IntVectorArray `cellIntRangeUpperBounds`
  Storage array to hold cell upper bounds for discrete int range variables.
- IntVectorArray `cellIntSetBounds`
  Storage array to hold cell values for discrete integer set variables.
- IntVectorArray `cellRealSetBounds`
  Storage array to hold cell value for discrete real set variables.
- RealVectorArray `cellFnLowerBounds`
  Storage array to hold cell min.
- RealVectorArray `cellFnUpperBounds`
  Storage array to hold cell max.
- RealVector `cellBPA`
  Storage array to hold cell bpa.
- `size_t respFnCnt`
  response function counter
- `size_t cellCnt`
  cell counter
- `size_t numCells`
  total number of interval combinations

Additional Inherited Members

13.91.1 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.hpp
- NonDInterval.cpp

13.92 NonDLHSEvidence Class Reference

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

Inheritance diagram for NonDLHSEvidence:
13.93. NONDLHSINTERVAL CLASS REFERENCE

Public Member Functions

- NonDLHSEvidence (ProblemDescDB &problem_db, Model &model)
  
  constructor
- ~NonDLHSEvidence ()
  
  destructor
- void initialize ()
  
  perform any required initialization
- void post_process_samples ()
  
  post-process the output from executing lhsSampler

Additional Inherited Members

13.92.1 Detailed Description

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

The NonDEvidence class implements the propagation of epistemic uncertainty using Dempster-Shafer theory of evidence. In this approach, one assigns a set of basic probability assignments (BPA) to intervals defined for the uncertain variables. Input interval combinations are calculated, along with their BPA. Currently, the response function is evaluated at a set of sample points, then a response surface is constructed which is sampled extensively to find the minimum and maximum within each input interval cell, corresponding to the belief and plausibility within that cell, respectively. This data is then aggregated to calculate cumulative distribution functions for belief and plausibility.

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

- NonDLHSEvidence.hpp
- NonDLHSEvidence.cpp

13.93 NonDLHSInterval Class Reference

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

Inheritance diagram for NonDLHSInterval:
Public Member Functions

- **NonDLHSInterval** (ProblemDescDB &problem_db, Model &model)
  
  constructor

- **~NonDLHSInterval** ()
  
  destructor

- void **init_communicators** ()
  
  initialize the communicators associated with iteratedModel as well as any subordinate iterators or subordinate models

- void **free_communicators** ()
  
  free the communicators associated with iteratedModel as well as any subordinate iterators or subordinate models

- 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
Additional Inherited Members

13.93.1 Detailed Description

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

The NonDLHSInterval class implements the propagation of epistemic uncertainty using LHS-based methods. The documentation for this class was generated from the following files:

- NonDLHSInterval.hpp
- NonDLHSInterval.cpp

13.94 NonDLHSSampling Class Reference

Performs LHS and Monte Carlo sampling for uncertainty quantification.

Inheritance diagram for NonDLHSSampling:

```
    NonDLHSSampling
      |       
    NonDSampling
      |       
    NonD
      |       
    Iterator
```

Public Member Functions

- NonDLHSSampling (ProblemDescDB &problem_db, Model &model)
  - standard constructor

- NonDLHSSampling (Model &model, unsigned short sample_type, int samples, int seed, const String &rng, bool vary_pattern=true, short sampling_vars_mode=ACTIVE)
  - alternate constructor for sample generation and evaluation "on the fly"

- NonDLHSSampling (unsigned short sample_type, int samples, int seed, const String &rng, const RealVector &lower_bnds, const RealVector &upper_bnds)
  - alternate constructor for sample generation "on the fly"

- ~NonDLHSSampling ()
  - destructor

Protected Member Functions

- void pre_run ()
  - generate LHS samples in non-VBD cases

- void post_input ()
read tabular data for post-run mode

- void quantify_uncertainty ()
  perform the evaluate parameter sets portion of run

- void post_run (std::ostream &s)
  generate statistics for LHS runs in non-VBD cases

- void print_results (std::ostream &s)
  print the final statistics

Private Attributes

- size_t numResponseFunctions
  number of response functions; used to distinguish NonD from opt/NLS usage

- bool varBasedDecompFlag
  flags computation of variance-based decomposition indices

Additional Inherited Members

13.94.1 Detailed Description

Performs LHS and Monte Carlo sampling for uncertainty quantification.

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

13.94.2 Constructor & Destructor Documentation

NonDLHSSampling ( ProblemDescDB & problem_db, 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.

NonDLHSSampling ( Model & model, unsigned short sample_type, int samples, int seed, const String & rng, bool vary_pattern = true, short sampling_vars_mode = ACTIVE )

alternate constructor for sample generation and evaluation “on the fly”

This alternate constructor is used for generation and evaluation of Model-based sample sets. A set_db_list_-nodes has not been performed so required data must be passed through the constructor. It’s purpose is to avoid the need for a separate LHS specification within methods that use LHS sampling.

NonDLHSSampling ( unsigned short sample_type, int samples, int seed, const String & rng, const RealVector & lower_bnds, const RealVector & upper_bnds )

alternate constructor for sample generation ”on the fly”

This alternate constructor is used by ConcurrentStrategy for generation of uniform, uncorrelated sample sets. It is not a letter-envelope instantiation and a set_db_list_nodes has not been performed. It is called with all needed data passed through the constructor and is designed to allow more flexibility in variables set definition (i.e., relax
connection to a variables specification and allow sampling over parameter sets such as multiobjective weights). In this case, a Model is not used and the object must only be used for sample generation (no evaluation).

References NonDSampling::get_parameter_sets().

### 13.94.3 Member Function Documentation

```cpp
void quantify_uncertainty() [protected], [virtual]
```

perform the evaluate parameter sets portion of run

Loop over the set of samples and compute responses. Compute statistics on the set of responses if statsFlag is set.

Implements NonD.

References NonDSampling::allDataFlag, Analyzer::evaluate_parameter_sets(), Iterator::iteratedModel, Analyzer::numContinuousVars, Analyzer::numDiscreteIntVars, Analyzer::numDiscreteRealVars, NonDLHSSampling::numResponseFunctions, NonDSampling::numSamples, NonDSampling::statsFlag, NonDLHSSampling::varBasedDecompFlag, and Analyzer::variance_based_decomp().

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

- NonDLHSSampling.hpp
- NonDLHSSampling.cpp

### 13.95 NonDLHSSingleInterval Class Reference

Class for pure interval propagation using LHS.

Inheritance diagram for NonDLHSSingleInterval:

```
NonDLHSSingleInterval
NonDLHSInterval
NonDInterval
NonD
Analyzer
Iterator
```

**Public Member Functions**

- **NonDLHSSingleInterval (ProblemDescDB &problem_db, Model &model)**  
  *constructor*

- **~NonDLHSSingleInterval ()**  
  *destructor*
CHAPTER 13. CLASS DOCUMENTATION

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

Additional Inherited Members

13.95.1 Detailed Description

Class for pure interval propagation using LHS.

The NonDSingleInterval class implements the propagation of epistemic uncertainty using ... The documentation for this class was generated from the following files:

- NonDLHSSingleInterval.hpp
- NonDLHSSingleInterval.cpp

13.96 NonDLocalEvidence Class Reference

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

Inheritance diagram for NonDLocalEvidence:

```
Analyzer
   NonD
     NonDInterval
       NonDLocalInterval
         NonDLocalEvidence
```

Public Member Functions

- NonDLocalEvidence (ProblemDescDB &problem_db, Model &model)
  constructor
- ~NonDLocalEvidence ()
  destructor
Protected Member Functions

- void initialize ()
  perform any required initialization
- void set_cell_bounds ()
  set the optimization variable bounds for each cell
- void truncate_to_cell_bounds (RealVector &initial_pt)
  truncate initial_pt to respect current cell lower/upper bounds
- void post_process_cell_results (bool maximize)
  post-process a cell minimization/maximization result
- void post_process_response_fn_results ()
  post-process the interval computed for a response function
- void post_process_final_results ()
  perform final post-processing

Additional Inherited Members

13.96.1 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:

- NonDLocalEvidence.hpp
- NonDLocalEvidence.cpp

13.97 NonDLocalInterval Class Reference

Class for using local gradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification.

Inheritance diagram for NonDLocalInterval:
Public Member Functions

- **NonDLocalInterval** (ProblemDescDB &problem_db, Model &model)
  constructor
- **~NonDLocalInterval** ()
  destructor
- void **init_communicators** ()
  initialize the communicators associated with iteratedModel as well as any subordinate iterators or subordinate models
- void **free_communicators** ()
  free the communicators associated with iteratedModel as well as any subordinate iterators or subordinate models
- void **quantify_uncertainty** ()
  Performs a gradient-based optimization to determine interval bounds for an entire function or interval bounds on a particular statistical estimator.
- unsigned short **uses_method** () const
  return name of active optimizer method
- void **method_recourse** ()
  perform an MPP optimizer method switch due to a detected conflict

Protected Member Functions

- virtual void **initialize** ()
  perform any required initialization
- virtual void **set_cell_bounds** ()
  set the optimization variable bounds for each cell
- virtual void **truncate_to_cell_bounds** (RealVector &initial_pt)
  truncate initial_pt to respect current cell lower/upper bounds
- virtual void **post_process_cell_results** (bool maximize)
  post-process a cell minimization/maximization result
- virtual void **post_process_response_fn_results** ()
post-process the interval computed for a response function

- virtual void post_process_final_results()
  perform final post-processing

Protected Attributes

- Iterator minMaxOptimizer
  local gradient-based optimizer
- Model minMaxModel
  recast model which extracts the active objective function

Static Private Member Functions

- static void extract_objective(const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  static function used to extract the active objective function when optimizing for an interval lower or upper bound

Private Attributes

- bool npsolFlag
  flag representing the gradient-based optimization algorithm selection (NPSOL SQP or OPT++ NIP)

Static Private Attributes

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

Additional Inherited Members

13.97.1 Detailed Description

Class for using local gradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification.

The NonDLocalInterval class supports local gradient-based optimization approaches to determining interval bounds for epistemic UQ. The interval bounds may be on the entire function in the case of pure interval analysis (e.g. intervals on input = intervals on output), or the intervals may be on statistics of an "inner loop" aleatory analysis such as intervals on means, variances, or percentile levels.

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

- NonDLocalInterval.hpp
- NonDLocalInterval.cpp
13.98 NonDLocalReliability Class Reference

Class for the reliability methods within DAKOTA/UQ.

Inheritance diagram for NonDLocalReliability:

```
NonDLocalReliability
  NonDReliability
    NonD
      Analyzer
        Iterator
```

Public Member Functions

- **NonDLocalReliability (ProblemDescDB &problem_db, Model &model)**
  
  constructor

- **~NonDLocalReliability ()**
  
  destructor

- **void init_communicators ()**
  
  initialize the communicators associated with iteratedModel as well as any subordinate iterators or subordinate models

- **void free_communicators ()**
  
  free the communicators associated with iteratedModel as well as any subordinate iterators or subordinate models

- **void quantify_uncertainty ()**
  
  performs an uncertainty propagation using analytical reliability methods which solve constrained optimization problems to obtain approximations of the cumulative distribution function of response

- **void print_results (std::ostream &s)**
  
  print the approximate mean, standard deviation, and importance factors when using the mean value method or the CDF/CCDF information when using MPP-search-based reliability methods

- **unsigned short uses_method () const**
  
  return name of active MPP optimizer

- **void method_recourse ()**
  
  perform an MPP optimizer method switch due to a detected conflict

Private Member Functions

- **void initial_taylor_series ()**
  
  convenience function for performing the initial limit state Taylor-series approximation

- **void mean_value ()**
13.98. NONLOCALRELIABILITY CLASS REFERENCE

• void mpp_search()
  convenience function for encapsulating the reliability methods that employ a search for the most probable point (AMV, AMV+, FORM, SORM)

• void initialize_class_data()
  convenience function for initializing class scope arrays

• void initialize_level_data()
  convenience function for initializing/warm starting MPP search data for each response function prior to level 0

• void initialize_mpp_search_data()
  convenience function for initializing/warm starting MPP search data for each z/p/beta level for each response function

• void update_mpp_search_data(const Variables &vars_star, const Response &resp_star)
  convenience function for updating MPP search data for each z/p/beta level for each response function

• void update_level_data()
  convenience function for updating z/p/beta level data and final statistics following MPP convergence

• void update_pma_maximize(const RealVector &mpp_u, const RealVector &fn_grad_u, const RealSymMatrix &fn_hess_u)
  update pmaMaximizeG from prescribed probabilities or prescribed generalized reliabilities by inverting second-order integrations

• void update_limit_state_surrogate()
  convenience function for passing the latest variables/response data to the data fit embedded within uSpaceModel

• void assign_mean_data()
  update mostProbPointX/U, computedRespLevel, fnGradX/U, and fnHessX/U from ranVarMeansX/U, fnValsMeanX, fnGradsMeanX, and fnHessiansMeanX

• void dg_ds_eval(const RealVector &x_vars, const RealVector &fn_grad_x, RealVector &final_stat_grad)
  convenience function for evaluating dg/ds

• Real dp2_dbeta_factor(Real beta, bool cdf_flag)
  compute factor for derivative of second-order probability with respect to reliability index (from differentiating BR-EITUNG or HOHENRACK expressions)

• Real signed_norm(const RealVector &mpp_u, const RealVector &fn_grad_u, bool cdf_flag)
  convert norm of mpp_u (u-space solution) to a signed reliability index

• Real signed_norm(Real norm_mpp_u)
  convert norm of u-space vector to a signed reliability index

• Real signed_norm(Real norm_mpp_u, const RealVector &mpp_u, const RealVector &fn_grad_u, bool cdf_flag)
  shared helper function

• Real probability(Real beta)
  Convert reliability to probability using a first-order integration.

• Real probability(Real beta, bool cdf_flag, const RealVector &mpp_u, const RealVector &fn_grad_u, const RealSymMatrix &fn_hess_u)
  Convert computed reliability to probability using either a first-order or second-order integration.

• Real probability(Real beta, bool cdf_flag, const RealVector &mpp_u, const RealVector &fn_grad_u, const RealSymMatrix &fn_hess_u)
  Convert provided reliability to probability using either a first-order or second-order integration.
• Real reliability (Real p)
  Convert probability to reliability using the inverse of a first-order integration.
• Real reliability (Real p, bool cdf_flag, const RealVector &mpp_u, const RealVector &fn_grad_u, const RealSymMatrix &fn_hess_u)
  Convert probability to reliability using the inverse of a first-order or second-order integration.
• bool reliability_residual (const Real &p, const Real &beta, const RealVector &kappa, Real &res)
  compute the residual for inversion of second-order probability corrections using Newton’s method (called by reliability(p))
• Real reliability_residual_derivative (const Real &p, const Real &beta, const RealVector &kappa)
  compute the residual derivative for inversion of second-order probability corrections using Newton’s method (called by reliability(p))
• void principal_curvatures (const RealVector &mpp_u, const RealVector &fn_grad_u, const RealSymMatrix &fn_hess_u, RealVector &kappa_u)
  Compute the kappaU vector of principal curvatures from fnHessU.
• void scale_curvature (Real beta, bool cdf_flag, const RealVector &kappa, RealVector &scaled_kappa)
  scale copy of principal curvatures by -1 if needed; else take a view

Static Private Member Functions

• static void RIA_objective_eval (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  static function used as the objective function in the Reliability Index Approach (RIA) problem formulation. This equality-constrained optimization problem performs the search for the most probable point (MPP) with the objective function of \((\|u\|^2)\).
• static void RIA_constraint_eval (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  static function used as the constraint function in the Reliability Index Approach (RIA) problem formulation. This equality-constrained optimization problem performs the search for the most probable point (MPP) with the constraint of \(G(u) = \text{response level}\).
• static void PMA_objective_eval (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  static function used as the objective function in the Performance Measure Approach (PMA) problem formulation. This equality-constrained optimization problem performs the search for the most probable point (MPP) with the objective function of \(G(u)\).
• static void PMA_constraint_eval (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  static function used as the constraint function in the first-order Performance Measure Approach (PMA) problem formulation. This optimization problem performs the search for the most probable point (MPP) with the equality constraint of \((\|u\|^2) = (\beta-\beta)\).
• static void PMA2_constraint_eval (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  static function used as the constraint function in the second-order Performance Measure Approach (PMA) problem formulation. This optimization problem performs the search for the most probable point (MPP) with the equality constraint of \(\beta = \beta-\beta\).
• static void PMA2_set_mapping (const Variables &recast_vars, const ActiveSet &recast_set, ActiveSet &sub_model_set)
  static function used to augment the sub-model ASV requests for second-order PMA
Private Attributes

- Real `computedRespLevel`
  
  output response level calculated

- Real `computedRelLevel`
  
  output reliability level calculated for RIA and 1st-order PMA

- Real `computedGenRelLevel`
  
  output generalized reliability level calculated for 2nd-order PMA

- RealVector `fnGradX`
  
  actual x-space gradient for current function from most recent response evaluation

- RealVector `fnGradU`
  
  u-space gradient for current function updated from fnGradX and Jacobian dx/du

- RealSymMatrix `fnHessX`
  
  actual x-space Hessian for current function from most recent response evaluation

- RealSymMatrix `fnHessU`
  
  u-space Hessian for current function updated from fnHessX and Jacobian dx/du

- RealVector `kappaU`
  
  principal curvatures derived from eigenvalues of orthonormal transformation of fnHessU

- RealVector `fnValsMeanX`
  
  response function values evaluated at mean x

- RealMatrix `fnGradsMeanX`
  
  response function gradients evaluated at mean x

- RealSymMatrixArray `fnHessiansMeanX`
  
  response function Hessians evaluated at mean x

- RealVector `ranVarMeansU`
  
  vector of means for all uncertain random variables in u-space

- bool `initialPtUserSpec`
  
  flag indicating user specification of (any portion of) initialPtU

- RealVector `initialPtUSpec`
  
  user specification or default initial guess for local optimization

- RealVector `initialPtU`
  
  current starting point for MPP searches in u-space

- RealVector `mostProbPointX`
  
  location of MPP in x-space

- RealVector `mostProbPointU`
  
  location of MPP in u-space

- RealVectorArray `prevMPPULev0`
  
  array of converged MPP’s in u-space for level 0. Used for warm-starting initialPtU within RBDO.

- RealMatrix `prevFnGradDLev0`
  
  matrix of limit state sensitivities w.r.t. inactive/design variables for level 0. Used for warm-starting initialPtU within RBDO.

- RealMatrix `prevFnGradULev0`
  
  matrix of limit state sensitivities w.r.t. active/uncertain variables for level 0. Used for warm-starting initialPtU within RBDO.
• RealVector prevICVars
  previous design vector. Used for warm-starting initialPtU within RBDO.
• ShortArray prevCumASVLev0
  accumulation (using \( = \)) of all previous design ASV’s from requested finalStatistics. Used to detect availability of prevFnGradDLev0 data for warm-starting initialPtU within RBDO.
• bool npsolFlag
  flag representing the optimization MPP search algorithm selection (NPSOL SQP or OPT++ NIP)
• bool warmStartFlag
  flag indicating the use of warm starts
• bool nipModeOverrideFlag
  flag indicating the use of move overrides within OPT++ NIP
• bool curvatureDataAvailable
  flag indicating that sufficient data (i.e., fnGradU, fnHessU, mostProbPointU) is available for computing principal curvatures
• bool kappaUpdated
  track when kappaU requires updating via principal_curvatures()
• short integrationOrder
  integration order (1 or 2) provided by integration specification
• short secondOrderIntType
  type of second-order integration: Breitung, Hohenbichler-Rackwitz, or Hong
• Real curvatureThresh
  cut-off value for \( 1/\sqrt{\cdot} \) term in second-order probability corrections.
• short taylorOrder
  order of Taylor series approximations (1 or 2) in MV/AMV/AMV+ derived from hessian type
• RealMatrix impFactor
  importance factors predicted by MV
• int npsolDerivLevel
  derivative level for NPSOL executions (1 = analytic grads of objective fn, 2 = analytic grads of constraints, 3 = analytic grads of both).
• unsigned short warningBits
  set of warnings accumulated during execution

**Static Private Attributes**

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

**Additional Inherited Members**

13.98.1 Detailed Description

Class for the reliability methods within DAKOTA/UQ.

The NonDLocalReliability class implements the following reliability methods through the support of different limit state approximation and integration options: mean value (MVFOSM/MVSOSM), advanced mean value
method (AMV, AMV^2) in x- or u-space, iterated advanced mean value method (AMV+, AMV^2+) in x- or u-space, two-point adaptive nonlinearity approximation (TANA) in x- or u-space, first order reliability method (FORM), and second order reliability method (SORM). All options except mean value employ an optimizer (currently NPSOL SQP or OPT++ NIP) to solve an equality-constrained optimization problem for the most probable point (MPP). The MPP search may be formulated as the reliability index approach (RIA) for mapping response levels to reliabilities/probabilities or as the performance measure approach (PMA) for performing the inverse mapping of reliability/probability levels to response levels.

13.98.2 Member Function Documentation

void RIA\_objective\_eval ( const Variables & sub\_model\_vars, const Variables & recast\_vars, const Response & sub\_model\_response, Response & recast\_response ) [static],[private]

static function used as the objective function in the Reliability Index Approach (RIA) problem formulation. This equality-constrained optimization problem performs the search for the most probable point (MPP) with the objective function of (\|u\|^2).

This function recasts a G(u) response set (already transformed and approximated in other recursions) into an RIA objective function.

References Response::active\_set\_request\_vector(), Variables::continuous\_variables(), Response::function\_gradient\_view(), Response::function\_hessian\_view(), and Response::function\_value().

Referenced by NonDLocalReliability::mpp\_search().

void RIA\_constraint\_eval ( const Variables & sub\_model\_vars, const Variables & recast\_vars, const Response & sub\_model\_response, Response & recast\_response ) [static],[private]

static function used as the constraint function in the Reliability Index Approach (RIA) problem formulation. This equality-constrained optimization problem performs the search for the most probable point (MPP) with the constraint of G(u) = response level.

This function recasts a G(u) response set (already transformed and approximated in other recursions) into an RIA equality constraint.

References Response::active\_set\_request\_vector(), Response::function\_gradient(), Response::function\_gradient\_view(), Response::function\_hessian(), Response::function\_value(), NonDLocalReliability::nondLocRelInstance, NonDReliability::requested\_Target\_Level, and NonDReliability::resp\_Fn\_Count.

Referenced by NonDLocalReliability::mpp\_search().

void PMA\_objective\_eval ( const Variables & sub\_model\_vars, const Variables & recast\_vars, const Response & sub\_model\_response, Response & recast\_response ) [static],[private]

static function used as the objective function in the Performance Measure Approach (PMA) problem formulation. This equality-constrained optimization problem performs the search for the most probable point (MPP) with the objective function of G(u).

This function recasts a G(u) response set (already transformed and approximated in other recursions) into a PMA objective function.

References Response::active\_set\_request\_vector(), Variables::continuous\_variables(), NonDLocalReliability::curvature\_Data\_Available, Response::function\_gradient(), Response::function\_gradient\_view(), Response::function\_hessian(), Response::function\_hessian\_view(), Response::function\_value(), NonDLocalReliability::integration\_Order, NonDLocalReliability::kappa\_Updated, NonDReliability::mpp\_Search\_Type, NonDLocalReliability::nondLocRel\_Instance, NonDReliability::pma\_Maximize\_G, NonDReliability::resp\_Fn\_Count, and NonDLocalReliability::update\_pma\_maximize().

Referenced by NonDLocalReliability::mpp\_search().
void PMA_constraint_eval ( const Variables & sub_model_vars, const Variables & recast_vars, const Response & sub_model_response, Response & recast_response ) [static], [private]

static function used as the constraint function in the first-order Performance Measure Approach (PMA) problem formulation. This optimization problem performs the search for the most probable point (MPP) with the equality constraint of (norm u)² = (beta-bar)².

This function recasts a G(u) response set (already transformed and approximated in other recursions) into a first-order PMA equality constraint on reliability index beta.

References Response::active_set_request_vector(), Variables::continuous_variables(), Response::function_gradient_view(), Response::function_hessian_view(), Response::function_value(), NonDLocalReliability::nondLocRelInstance, and NonDReliability::requestedTargetLevel.

Referenced by NonDLocalReliability::mpp_search().

void PMA2_constraint_eval ( const Variables & sub_model_vars, const Variables & recast_vars, const Response & sub_model_response, Response & recast_response ) [static], [private]

static function used as the constraint function in the second-order Performance Measure Approach (PMA) problem formulation. This optimization problem performs the search for the most probable point (MPP) with the equality constraint of beta* = beta*-bar.

This function recasts a G(u) response set (already transformed and approximated in other recursions) into a second-order PMA equality constraint on generalized reliability index beta-star.

References Dakota::abort_handler(), Response::active_set_request_vector(), NonD::cdfFlag, NonDLocalReliability::computedGenRelLevel, NonDLocalReliability::computedRelLevel, Variables::continuous_variables(), NonDLocalReliability::dp2_dbeta_factor(), NonDLocalReliability::fnGradU, NonDLocalReliability::fnHessU, Response::function_gradient_view(), Response::function_hessian(), Response::function_value(), NonDLocalReliability::mostProbPointU, NonDReliability::mppSearchType, NonDLocalReliability::nondLocRelInstance, NonDLocalReliability::probability(), NonDLocalReliability::reliability(), NonDReliability::requestedTargetLevel, NonDReliability::respFnCount, and NonDLocalReliability::signed_norm().

Referenced by NonDLocalReliability::mpp_search().

void initial_taylor_series ( ) [private]

convenience function for performing the initial limit state Taylor-series approximation

An initial first- or second-order Taylor-series approximation is required for MV/AMV/AMV+/TANA or for the case where momentStats (from MV) are required within finalStatistics for subIterator usage of NonDLocalReliability.

References Response::active_set_request_vector(), Iterator::activeSet, Model::component_parallel_mode(), Model::compute_response(), Model::continuous_variables(), Model::current_response(), NonD::finalStatistics, NonDLocalReliability::fnGradsMeanX, NonDLocalReliability::fnHessiansMeanX, NonDLocalReliability::fnValsMeanX, Response::function_gradients(), Response::function_hessians(), Response::function_values(), Model::hessian_type(), Iterator::iteratedModel, NonD::momentStats, NonDReliability::mppSearchType, NonD::natafTransform, Analyzer::numFunctions, NonD::numUncertainVars, ActiveSet::request_vector(), NonD::requestedGenRelLevels, NonD::requestedProbLevels, NonD::requestedRespLevels, Iterator::subIteratorFlag, NonDLocalReliability::taylorOrder, and NonDReliability::uSpaceModel.

Referenced by NonDLocalReliability::mean_value(), and NonDLocalReliability::mpp_search().

void initialize_class_data ( ) [private]

convenience function for initializing class scope arrays

Initialize class-scope arrays and perform other start-up activities, such as evaluating median limit state responses.
References Response::active_set_derivative_vector(), NonD::finalStatistics, NonDReliability::importanceSampler, NonD::initialize_random_variables(), NonDReliability::integrationRefinement, Iterator::iterator_rep(), NonDReliability::mppModel, NonD::natafTransform, Analyzer::numFunctions, NonDReliability::numRelAnalyses, NonD::numUncertainVars, NonDLocalReliability::prevCumASVLev0, NonDLocalReliability::prevCumGradASVLev0, NonDLocalReliability::prevCumGradULev0, NonDLocalReliability::prevMPPULev0, NonDLocalReliability::ranVarMeansU, Iterator::subIteratorFlag, Model::update_from_subordinate_model(), and NonDLocalReliability::warmStartFlag.

Referenced by NonDLocalReliability::mpp_search().

```cpp
void initialize_level_data() [private]
```

convenience function for initializing/warm starting MPP search data for each response function prior to level 0

For a particular response function prior to the first z/p/beta level, initialize/warm-start optimizer initial guess (initialPtU), expansion point (mostProbPointX/U), and associated response data (computedRespLevel, fnGradX/U, and fnHessX/U).

References Iterator::activeSet, NonDLocalReliability::assign_mean_data(), Model::component_parallel_mode(), Model::compute_response(), NonDLocalReliability::computedRespLevel, Model::continuous_variable_ids(), Model::continuous_variables(), Dakota::copy_data(), Model::current_response(), NonDLocalReliability::curvatureDataAvailable, NonDLocalReliability::fnGradU, NonDLocalReliability::fnGradX, NonDLocalReliability::fnHessU, NonDLocalReliability::fnHessX, Response::function_gradient_copy(), Response::function_hessian(), Response::function_value(), Model::inactive_continuous_variables(), NonDLocalReliability::initialPtU, NonDLocalReliability::initialPtUSpec, Iterator::iteratedModel, NonDLocalReliability::kappaUpdated, NonDLocalReliability::mostProbPointU, NonDLocalReliability::mostProbPointX, NonDReliability::mppSearchType, NonD::natafTransform, NonDReliability::numRelAnalyses, NonD::numUncertainVars, NonDLocalReliability::prevCumASVLev0, NonDLocalReliability::prevCumGradASVLev0, NonDLocalReliability::prevCumGradULev0, NonDLocalReliability::prevICVars, NonDLocalReliability::prevMPPULev0, ActiveSet::request_value(), ActiveSet::request_values(), NonD::requestedRespLevels, NonDReliability::respFnCount, Iterator::subIteratorFlag, Model::surrogate_function_indices(), NonDLocalReliability::taylorOrder, NonDLocalReliability::update_limit_state_surrogate(), NonDReliability::uSpaceModel, and NonDLocalReliability::warmStartFlag.

Referenced by NonDLocalReliability::mpp_search().

```cpp
void initialize_mpp_search_data() [private]
```

convenience function for initializing/warm starting MPP search data for each z/p/beta level for each response function

For a particular response function at a particular z/p/beta level, warm-start or reset the optimizer initial guess (initialPtU), expansion point (mostProbPointX/U), and associated response data (computedRespLevel, fnGradX/U, and fnHessX/U).

References NonDLocalReliability::assign_mean_data(), NonD::computedGenRelLevels, NonD::computedRelLevels, NonDLocalReliability::fnGradU, Model::hessian_type(), NonDLocalReliability::initialPtU, NonDLocalReliability::initialPtUSpec, NonDLocalReliability::integrationOrder, Iterator::iteratedModel, NonDReliability::levelCount, NonDLocalReliability::mostProbPointU, NonDReliability::mppSearchType, NonD::numUncertainVars, NonD::requestedProbLevels, NonD::requestedRelLevels, NonD::requestedRespLevels, NonDReliability::requestedTargetLevel, NonDReliability::respFnCount, NonDLocalReliability::taylorOrder, and NonDLocalReliability::warmStartFlag.

Referenced by NonDLocalReliability::mpp_search().

```cpp
void update_mpp_search_data( const Variables & vars_star, const Response & resp_star ) [private]
```

convenience function for updating MPP search data for each z/p/beta level for each response function
Includes case-specific logic for updating MPP search data for the AMV/AMV+/TANA/NO_APPROX methods.

References Response::active_set(), Response::active_set_request_vector(), Iterator::activeSet, NonDReliability::approxConverged, NonDReliability::approxIters, Model::component_parallel_mode(), Model::compute_response(), NonDLocalReliability::computedRelLevel, NonDLocalReliability::computedRespLevel, Model::continuous_variable_ids(), Variables::continuous_variables(), Model::continuous_variables(), Iterator::convergenceTol, Variables::copy(), Dakota::copy_data(), Model::current_response(), Model::current_variables(), NonDLocalReliability::curvatureDataAvailable, Dakota::data_pairs, NonD::finalStatistics, NonDLocalReliability::fnGradX, NonDLocalReliability::fnHessU, NonDLocalReliability::fnHessX, Response::function_gradient_copy(), Response::function_hessian(), Response::function_value(), Response::function_values(), NonDLocalReliability::initialPtU, NonDLocalReliability::integrationOrder, Model::interface_id(), Iterator::iteratedModel, NonDLocalReliability::kappaUpdated, NonDReliability::levelCount, Dakota::lookup_by_val(), Iterator::maxIterations, NonDLocalReliability::mostProbPointU, NonDLocalReliability::mostProbPointX, NonDReliability::mppSearchType, NonD::natafTransform, Analyzer::numFunctions, NonD::numNormalVars, NonD::numUncertainVars, NonDReliability::pmaMaximizeG, ActiveSet::request_value(), ActiveSet::request_values(), ActiveSet::request_vector(), NonD::requestedProbLevels, NonD::requestedRelLevels, NonD::requestedRespLevels, NonDReliability::respFnCount, NonDLocalReliability::signed_norm(), NonDReliability::statCount, NonDLocalReliability::taylorOrder, NonDLocalReliability::update_limit_state_surrogate(), NonDLocalReliability::update_pma_maximize(), NonDReliability::uSpaceModel, NonDLocalReliability::warmStartFlag, and NonDLocalReliability::warningBits.

Referenced by NonDLocalReliability::mpp_search().

```cpp
void update_level_data() [private]
```

convenience function for updating z/p/beta level data and final statistics following MPP convergence

Updates computedRespLevels/computedProbLevels/computedRelLevels, finalStatistics, warm start, and graphics data.

References Response::active_set_derivative_vector(), Response::active_set_request_vector(), Graphics::add_datapoint(), NonD::cdfFlag, NonDLocalReliability::computedGenRelLevel, NonD::computedGenRelLevels, NonD::computedProbLevels, NonDLocalReliability::computedRelLevel, NonD::computedRelLevels, NonD::computedRespLevels, Dakota::dakota_graphics, NonD::finalStatistics, NonDLocalReliability::fnGradU, NonDLocalReliability::fnHessU, Response::function_gradient(), NonDLocalReliability::integrationOrder, NonDLocalReliability::levelCount, NonDLocalReliability::mostProbPointU, NonDLocalReliability::mostProbPointX, Graphics::new_dataset(), Analyzer::numFunctions, NonD::numUncertainVars, NonDLocalReliability::prevCumASVLev0, NonDLocalReliability::prevGradDLev0, NonDLocalReliability::prevGradULev0, NonDLocalReliability::prevMPPULev0, NonDLocalReliability::probability(), NonDLocalReliability::reliability(), NonD::requestedGenRelLevels, NonD::requestedProbLevels, NonD::requestedRelLevels, NonD::requestedRespLevels, NonDReliability::respFnCount, NonD::respLevelTarget, NonD::respLevelTargetReduce, NonDReliability::statCount, Iterator::subIteratorFlag, NonD::totalLevelRequests, and NonDLocalReliability::warmStartFlag.

Referenced by NonDLocalReliability::mpp_search().

```cpp
void dg_ds_eval ( const RealVector & x_vars, const RealVector & fn_grad_x, RealVector & final_stat_grad ) [private]
```

convenience function for evaluating dg/ds

Computes dg/ds where s = design variables. Supports potentially overlapping cases of design variable augmentation and insertion.

References Response::active_set_derivative_vector(), Iterator::activeSet, Model::all_continuous_variable_ids(), Model::component_parallel_mode(), Model::compute_response(), Dakota::contains(), Model::continuous_variable_ids(), Model::continuous_variables(), Dakota::copy_data(), Model::current_response(), ActiveSet::derivative_vector(),
Real dp2_dbeta_factor ( Real beta, bool cdf_flag ) [private]

compute factor for derivative of second-order probability with respect to reliability index (from differentiating BREITUNG or HOHENRACK expressions)

  Compute sensitivity of second-order probability w.r.t. beta for use in derivatives of p^2 or beta^2 w.r.t. auxiliary parameters s (design, epistemic) or derivatives of beta w.r.t. u in PMA2_constraint_eval().

  References Dakota::abort_handler(), NonDLocalReliability::curvatureDataAvailable, NonDLocalReliability::curvatureThresh, NonDLocalReliability::kappaU, NonD::numUncertainVars, NonDLocalReliability::probability(), NonDLocalReliability::scale_curvature(), NonDLocalReliability::secondOrderIntType, and NonDLocalReliability::warningBits.

  Referenced by NonDLocalReliability::PMA2_constraint_eval(), and NonDLocalReliability::update_level_data().

Real probability ( Real beta, bool cdf_flag, const RealVector & mpp_u, const RealVector & fn_grad_u, const RealSymMatrix & fn_hess_u ) [private]

Convert provided reliability to probability using either a first-order or second-order integration.

  Converts beta into a probability using either first-order (FORM) or second-order (SORM) integration. The SORM calculation first calculates the principal curvatures at the MPP (using the approach in Ch. 8 of Haldar & Mahadevan), and then applies correction formulations from the literature (Breitung, Hohenbichler-Rackwitz, or Hong).

  References NonDLocalReliability::curvatureDataAvailable, NonDLocalReliability::curvatureThresh, NonDAdaptImpSampling::final_probability(), NonDReliability::importanceSampler, NonDLocalReliability::integrationOrder, NonDReliability::integrationRefinement, Iterator::iterator_rep(), NonDLocalReliability::kappaU, NonDLocalReliability::kappaUpdated, NonD::numUncertainVars, Iterator::outputLevel, NonDLocalReliability::principal_curvatures(), NonDLocalReliability::probability(), NonDReliability::requestedTargetLevel, NonDReliability::respFnCount, Iterator::run(), NonDLocalReliability::scale_curvature(), NonDLocalReliability::secondOrderIntType, NonDLocalReliability::warningBits, and Dakota::write_precision.

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

  - NonDLocalReliability.hpp
  - NonDLocalReliability.cpp

13.99 NonDLocalSingleInterval Class Reference

Class for using local gradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification.

  Inheritance diagram for NonDLocalSingleInterval:
Public Member Functions

- `NonDLocalSingleInterval (ProblemDescDB &problem_db, Model &model)`
  constructor
- `~NonDLocalSingleInterval ()`
  destructor

Protected Member Functions

- `void initialize ()`
  perform any required initialization
- `void post_process_cell_results (bool maximize)`
  post-process a cell minimization/maximization result

Private Attributes

- `size_t statCntr`
  counter for finalStatistics

Additional Inherited Members

13.99.1 Detailed Description

Class for using local gradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification.

The `NonDLocalSingleInterval` class supports local gradient-based optimization approaches to determining interval bounds for epistemic UQ. The interval bounds may be on the entire function in the case of pure interval analysis (e.g. intervals on input = intervals on output), or the intervals may be on statistics of an “inner loop” aleatory analysis such as intervals on means, variances, or percentile levels.

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

- `NonDLocalSingleInterval.hpp`
- `NonDLocalSingleInterval.cpp`
13.100  NonDPOFDarts Class Reference

Base class for POF Dart methods within DAKOTA/UQ.

Inheritance diagram for NonDPOFDarts:

- NonDPOFDarts
  - NonD
  - Analyzer
  - Iterator

Public Member Functions

- NonDPOFDarts (ProblemDescDB &problem_db, Model &model)
  constructor
- ~NonDPOFDarts()
  destructor
- void quantify_uncertainty()
  perform POFDart analysis and return probability of failure

Protected Member Functions

- void initiate_random_number_generator(unsigned long x)
  POF Darts Methods.
- double generate_a_random_number()
- void init_pof_darts()
- void exit_pof_darts()
- void execute(size_t kd)
- void print_POF_results(double lower, double upper)
- void print_results(std::ostream &s)
  print the final statistics
- void classical_dart_throwing_games(size_t game_index)
- void line_dart_throwing_games(size_t game_index)
- bool valid_dart(double *x)
- bool valid_line_flat(size_t flat_dim, double *flat_dart)
- void add_point(double *x)
- void assign_sphere_radius_POF(double *x, size_t isample)
- void compute_response_update_Lip(double *x)
- void shrink_big_spheres()
- double area_triangle(double x1, double y1, double x2, double y2, double x3, double y3)
- void initialize_surrogates()
- void add_surrogate_data(const Variables &vars, const Response &resp)
• void build_surrogate (size_t fn_index)
• Real eval_surrogate (size_t fn_index, double *vin)
• double f_true (double *x)
• void plot_vertices_2d ()

Protected Attributes
• int samples
• int seed
• double Q [1220]
• int indx
• double cc
• double c
• double zc
• double zx
• double zy
• size_t qlen
• size_t n_dim
• double * xmin
• double * xmax
• double _failure_threshold
• double _num_darts
• double _num_successive_misses_p
• double _num_successive_misses_m
• double _max_num_successive_misses
• double _max_radius
• double _accepted_void_ratio
• size_t _num_inserted_points
• size_t _total_budget
• double ** _sample_points
• double * _dart
• size_t _flat_dim
• size_t * _line_flat
• size_t _num_flat_segments
• double * _line_flat_start
• double * _line_flat_end
• double * _line_flat_length
• double * _Lip
• double ** _fval
• size_t _active_response_function
• SharedApproxData sharedData
• std::vector< Approximation > gpApproximations
• Variables gpEvalVars
Additional Inherited Members

13.100.1 Detailed Description

Base class for POF Dart methods within DAKOTA/UQ.

The NonDPoFDart class implements the calculation of a failure probability for a specified threshold for a
specified response function using the concepts developed by Mohamed Ebeida. The approach works by throwing
down a number of Poisson disk samples of varying radii, and identifying each disk as either in the failure or
safe region. The center of each disk represents a "true" function evaluation. kd-darts are used to place additional
points, in such a way to target the failure region. When the disks cover the space sufficiently, Monte Carlo methods
or a box volume approach is used to calculate both the lower and upper bounds on the failure probability.

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

- NonDPOFDarts.hpp
- NonDPOFDarts.cpp

13.101 NonDPolynomialChaos Class Reference

Nonintrusive polynomial chaos expansion approaches to uncertainty quantification.

Inheritance diagram for NonDPolynomialChaos:

```
Iterator
  Analyzer
    NonD
      NonDExpansion
        NonDPolynomialChaos
```

Public Member Functions

- **NonDPolynomialChaos** (ProblemDescDB &problem_db, Model &model)
  *standard constructor*
- **NonDPolynomialChaos** (Model &model, short exp_coeffs_approach, unsigned short num_int_level, short
  u_space_type, bool piecewise_basis, bool use_derivs)
  *alternate constructor*
- **~NonDPolynomialChaos** ()
  *destructor*

Protected Member Functions

- void init_communicators ()
initialize the communicators associated with iteratedModel as well as any subordinate iterators or subordinate models

- void freeCommunicators()
  free the communicators associated with iteratedModel as well as any subordinate iterators or subordinate models

- void resolveInputs(short &u_space_type, short &data_order)
  perform error checks and mode overrides

- void initialize_u_space_model()
  initialize uSpaceModel polynomial approximations with PCE/SC data

- void incrementSpecificationSequence()
  increment the input specification sequence (PCE only)

- void computeExpansion()
  form or import an orthogonal polynomial expansion using PCE methods

- void incrementOrder()
  uniformly increment the order of the polynomial chaos expansion

- void printCoefficients(std::ostream &s)
  print the PCE coefficient array for the orthogonal basis

- void archiveCoefficients()
  archive the PCE coefficient array for the orthogonal basis

Private Member Functions

- int terms_ratio_to_samples(size_t num_exp_terms, Real colloc_ratio, Real terms_order)
  convert number of expansion terms and collocation ratio to a number of collocation samples

- Real terms_samples_to_ratio(size_t num_exp_terms, int samples, Real terms_order)
  convert number of expansion terms and number of collocation samples to a collocation ratio

- void order_to_dim_preference(const UShortArray &order, unsigned short &p, RealVector &dim_pref)
  convert an isotropic/anisotropic expansion_order vector into a scalar plus a dimension preference vector

Private Attributes

- String expansionExportFile
  filename for export of chaos coefficients

- String expansionImportFile
  filename for import of chaos coefficients

- Real collocRatio
  factor applied to terms'^termsOrder in computing number of regression points, either user specified or inferred

- Real termsOrder
  exponent applied to number of expansion terms for computing number of regression points

- bool tensorRegression
  option for regression PCE using a filtered set tensor-product points

- bool crossValidation
  flag for use of cross-validation for selection of parameter settings in regression approaches

- RealVector noiseTols
  noise tolerance for compressive sensing algorithms; vector form used in cross-validation
13.101. NONDPOLYNOMIALCHAOS CLASS REFERENCE

- **Real l2Penalty**
  
  $L_2$ penalty for LASSO algorithm (elastic net variant)

- **UShortArray expOrderSeqSpec**
  
  user specification for expansion order (array for multifidelity)

- **RealVector dimPrefSpec**
  
  user specification for dimension preference

- **SizetArray collocPtsSeqSpec**
  
  user specification for collocation points (array for multifidelity)

- **SizetArray expSamplesSeqSpec**
  
  user specification for expansion samples (array for multifidelity)

- **size_t sequenceIndex**
  
  sequence index for \{expOrder,collocPts,expSamples\}SeqSpec

- **RealMatrix pceGradsMeanX**
  
  derivative of the PCE with respect to the x-space variables evaluated at the means (used as uncertainty importance metrics)

- **bool normalizedCoeffOutput**
  
  user request for use of normalization when outputting PCE coefficients

Additional Inherited Members

13.101.1 Detailed Description

Nonintrusive polynomial chaos expansion approaches to uncertainty quantification.

The NonDPolynomialChaos class uses a polynomial chaos expansion (PCE) approach to approximate the effect of parameter uncertainties on response functions of interest. It utilizes the OrthogPolyApproximation class to manage multiple types of orthogonal polynomials within a Wiener-Askey scheme to PCE. It supports PCE coefficient estimation via sampling, quadrature, point-collocation, and file import.

13.101.2 Constructor & Destructor Documentation

NonDPolynomialChaos (ProblemDescDB & problem_db, Model & model)

standard constructor

This constructor is called for a standard letter-envelope iterator instantiation using the ProblemDescDB.

References Dakota::abort_handler(), NonDIntegration::anisotropic_order_to_dimension_preference(), Model::assign_rep(), NonDPolynomialChaos::collocPtsSeqSpec, NonDPolynomialChaos::collocRatio, ParallelLibrary::command_line_check(), NonDExpansion::construct_cubature(), NonDExpansion::construct_expansion_sampler(), NonD::construct_lhs(), NonDExpansion::construct_quadrature(), NonDExpansion::construct_sparse_grid(), NonDIntegration::dimension_preference_to_anisotropic_order(), NonDPolynomialChaos::dimPrefSpec, NonDExpansion::expansionBasisType, NonDExpansion::expansionCoeffsApproach, NonDPolynomialChaos::expansionImportFile, NonDPolynomialChaos::expOrderSeqSpec, NonDPolynomialChaos::expSamplesSeqSpec, ProblemDescDB::get_bool(), ProblemDescDB::get_int(), ProblemDescDB::get_real(), ProblemDescDB::get_short(), ProblemDescDB::get_string(), ProblemDescDB::get_usa(), ProblemDescDB::get_ushort(), NonDExpansion::initialize(), NonDPolynomialChaos::initialize_u_space_model(), Iterator::iteratedModel, Iterator::maxEvalConcurrency, NonD::numContDesVars, NonD::numContEpistUncVars, Analyzer::numContinuousVars, NonD::numContStateVars, NonDExpansion::numSamplesOnModel, Iterator::outputLevel, ProblemDescDB::parallel_library(), Iterator::probDescDB, NonDExpansion::refineControl, NonDExpansion::refineType, NonDPolynomialChaos::resolve_inputs(),
NonDPolynomialChaos::sequenceIndex, NonDPolynomialChaos::tensorRegression, NonDPolynomialChaos::terms_ratio_to_samples(), NonDPolynomialChaos::terms_samples_to_ratio(), NonDPolynomialChaos::termsOrder, NonD::transform_model(), NonDExpansion::uSpaceModel, and Analyzer::vary_pattern().

NonDPolynomialChaos ( Model & model, short exp_coeffs_approach, unsigned short num_int_level, short u_space_type, bool piecewise_basis, bool use_derivs )

alternate constructor

This constructor is used for helper iterator instantiation on the fly.

References Model::assign_rep(), NonDExpansion::construct_cubature(), NonDExpansion::construct_quadrature(), NonDExpansion::construct_sparse_grid(), NonDExpansion::expansionCoeffsApproach, NonDExpansion::initialize(), NonDPolynomialChaos::initialize_u_space_model(), Iterator::iteratedModel, NonD::numContDesVars, NonD::numContEpistUncVars, NonD::numContStateVars, Iterator::outputLevel, NonDPolynomialChaos::resolve_inputs(), NonD::transform_model(), and NonDExpansion::uSpaceModel.

13.101.3 Member Function Documentation

void increment_specification_sequence ( ) [protected], [virtual]

increment the input specification sequence (PCE only)

default implementation is overridden by PCE

Reimplemented from NonDExpansion.

References NonDPolynomialChaos::collocPtsSeqSpec, NonDPolynomialChaos::collocRatio, SharedApproxData::data_rep(), NonDIntegration::dimension_preference_to_anisotropic_order(), NonDPolynomialChaos::dimPrefSpec, SharedPecosApproxData::expansion_order(), NonDExpansion::expansionBasisType, NonDExpansion::expansionCoeffsApproach, NonDPolynomialChaos::expOrderSeqSpec, NonDPolynomialChaos::expSamplesSeqSpec, NonDExpansion::increment_specification_sequence(), Iterator::iterator_rep(), NonDQuadrature::mode(), Model::model_rep(), Analyzer::numContinuousVars, NonDExpansion::numSamplesOnModel, NonDQuadrature::quadrature_order(), NonDQuadrature::samples(), Iterator::sampling_reference(), NonDPolynomialChaos::sequenceIndex, Model::shared_approximation(), Model::subordinate_iterator(), NonDPolynomialChaos::tensorRegression, NonDPolynomialChaos::terms_ratio_to_samples(), NonDPolynomialChaos::termsOrder, DataFitSurrModel::total_points(), NonDQuadrature::update(), and NonDExpansion::uSpaceModel.

void increment_order ( ) [protected], [virtual]

uniformly increment the order of the polynomial chaos expansion

Used for uniform refinement of regression-based PCE.

Reimplemented from NonDExpansion.

References NonDPolynomialChaos::collocRatio, SharedApproxData::data_rep(), SharedPecosApproxData::expansion_order(), NonDQuadrature::increment_grid(), SharedPecosApproxData::increment_order(), Iterator::iterator_rep(), NonDQuadrature::mode(), Model::model_rep(), NonDExpansion::numSamplesOnModel, NonDQuadrature::samples(), Iterator::sampling_reference(), Model::shared_approximation(), Model::subordinate_iterator(), NonDPolynomialChaos::tensorRegression, NonDPolynomialChaos::terms_ratio_to_samples(), NonDPolynomialChaos::termsOrder, DataFitSurrModel::total_points(), NonDQuadrature::update(), and NonDExpansion::uSpaceModel.

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

- NonDPolynomialChaos.hpp
- NonDPolynomialChaos.cpp
13.102 NonDQuadrature Class Reference

Derived nondeterministic class that generates N-dimensional numerical quadrature points for evaluation of expectation integrals over uncorrelated standard normals/uniforms/exponentials/betas/gammas.

Inheritance diagram for NonDQuadrature:

```
NonDQuadrature
  |          |      |
  |          |      |
  |          |      |
  NonDInte       Analyzer          Iterator
  NonD
  NonDIntegration
```

Public Member Functions

- **NonDQuadrature** (Model &model, const UShortArray &quad_order_seq, const RealVector &dim_pref)
  alternate constructor for instantiations "on the fly" based on a quadrature order specification
- **NonDQuadrature** (Model &model, int num_filt_samples, const RealVector &dim_pref)
  alternate constructor for instantiations "on the fly" that generate a filtered tensor product sample set
- **NonDQuadrature** (Model &model, int num_rand_samples, int seed, const UShortArray &quad_order_seq, const RealVector &dim_pref)
  alternate constructor for instantiations "on the fly" that sample randomly from a tensor product multi-index
- void **increment_grid** ()
  increment SSG level/TPQ order
- void **update** ()
  propagate any numSamples updates and/or grid updates/increments
- const Pecos::UShortArray & **quadrature_order** () const
  return Pecos::TensorProductDriver::quadOrder
- void **quadrature_order** (const Pecos::UShortArray &dim_quad_order)
  set dimQuadOrderRef and map to Pecos::TensorProductDriver::quadOrder
- void **samples** (size_t samples)
  set numSamples
- short **mode** () const
  return quadMode

Protected Member Functions

- **NonDQuadrature** (ProblemDescDB &problem_db, Model &model)
  constructor
- ~**NonDQuadrature** ()
CHAPTER 13. CLASS DOCUMENTATION

destructor

- **void initialize_grid** (const std::vector<Pecos::BasisPolynomial> &poly_basis)
- **void get_parameter_sets** (Model &model)

  Returns one block of samples (ndim * num_samples)

- **void reset**()

  restore initial state for repeated sub-iterator executions

- **void sampling_reset** (int min_samples, bool all_data_flag, bool stats_flag)
- **void increment_grid_preference** (const RealVector &dim_pref)

  increment SSG level/TPQ order and update anisotropy

- **void increment_specification_sequence**()

  increment sequenceIndex and update active orders/levels

- **int num_samples** () const

Private Member Functions

- **void increment_grid** (UShortArray &dim_quad_order)

  convenience function used to make increment_grid() more modular

- **void increment_grid_preference** (const RealVector &dim_pref, UShortArray &dim_quad_order)

  convenience function used to make increment_grid_preference() more modular

- **void compute_minimum_quadrature_order** (size_t min_samples, const RealVector &dim_pref, UShortArray &dim_quad_order)

  calculate smallest dim_quad_order with at least min_samples

- **void filter_parameter_sets**()

  prune allSamples back to size numSamples, retaining points with highest product weight

- **void update_anisotropic_order** (const RealVector &dim_quad_order_ref)

  update quad_order_ref based on an updated dimension preference, enforcing previous values as a lower bound

- **void initialize_dimension_quadrature_order** (unsigned short quad_order_spec, const RealVector &dim_pref_spec, UShortArray &dim_quad_order)

  initialize dim_quad_order from quad_order_spec and dim_pref_spec

- **void increment_dimension_quadrature_order** (UShortArray &dim_quad_order)

  increment each dim_quad_order entry by 1

- **void increment_dimension_quadrature_order** (const RealVector &dim_pref, UShortArray &dim_quad_order)

  increment the dim_quad_order entry with maximum preference by 1 and then rebalance

Private Attributes

- Pecos::TensorProductDriver * tpqDriver

  convenience pointer to the numIntDriver representation

- **bool nestedRules**

  for studies involving refinement strategies, allow for use of nested quadrature rules such as Gauss-Patterson

- **UShortArray quadOrderSeqSpec**

  a sequence of scalar quadrature orders, one per refinement level

- **UShortArray dimQuadOrderRef**

  reference point for Pecos::TensorProductDriver::quadOrder: the original user specification for the number of Gauss points per dimension, plus any refinements posted by increment_grid()
13.102. NONDQUADRATURE CLASS REFERENCE

- short quadMode
  
  point generation mode: FULL_TENSOR, FILTERED_TENSOR, RANDOM_TENSOR

- size_t numSamples
  
  size of a subset of tensor quadrature points (filtered based on product weight or sampled uniformly from the tensor multi-index); used by the regression PCE approach known as "probabilistic collocation"

- int randomSeed
  
  seed for the random number generator used in sampling of the tensor multi-index

Additional Inherited Members

13.102.1 Detailed Description

Derived nondeterministic class that generates N-dimensional numerical quadrature points for evaluation of expectation integrals over uncorrelated standard normals/uniforms/exponentials/betas/gammas.

This class is used by NonDPolynomialChaos, but could also be used for general numerical integration of moments. It employs Gauss-Hermite, Gauss-Legendre, Gauss-Laguerre, Gauss-Jacobi and generalized Gauss--Laguerre quadrature for use with normal, uniform, exponential, beta, and gamma density functions and integration bounds. The abscissas and weights for one-dimensional integration are extracted from the appropriate Orthogonal-Polynomial class and are extended to n-dimensions using a tensor product approach.

13.102.2 Constructor & Destructor Documentation

NonDQuadrature ( Model & model, const UShortArray & quad ORDER_SEQ, const RealVector & dim PREF )

alternate constructor for instantiations "on the fly" based on a quadrature order specification

This alternate constructor is used for on-the-fly generation and evaluation of numerical quadrature points.

References NonDIntegration::numIntDriver, and NonDQuadrature::tpqDriver.

NonDQuadrature ( Model & model, int num filt samples, const RealVector & dim PREF )

alternate constructor for instantiations "on the fly" that generate a filtered tensor product sample set

This alternate constructor is used for on-the-fly generation and evaluation of filtered tensor quadrature points.

References NonDIntegration::numIntDriver, and NonDQuadrature::tpqDriver.

NonDQuadrature ( Model & model, int num rand samples, int seed, const UShortArray & quad ORDER_SEQ, const RealVector & dim PREF )

alternate constructor for instantiations "on the fly" that sample randomly from a tensor product multi-index

This alternate constructor is used for on-the-fly generation and evaluation of random sampling from a tensor quadrature multi-index.

References NonDIntegration::numIntDriver, and NonDQuadrature::tpqDriver.

NonDQuadrature ( ProblemDescDB & problem_db, Model & model ) [protected]

constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification. It is not currently used, as there is not yet a separate nond_quadrature method specification.
References NonDIntegration::check_variables(), ProblemDescDB::get<bool>(), ProblemDescDB::get_short(),
Iterator::maxEvalConcurrency, NonD::natafTransform, NonDQuadrature::nestedRules, NonDIntegration::num-
IntDriver, Iterator::probDescDB, NonDQuadrature::reset(), and NonDQuadrature::tpqDriver.

13.102.3 Member Function Documentation

void initialize_grid ( const std::vector<Pecos::BasisPolynomial> & polybasis ) [protected], [virtual]

Used in combination with alternate NonDQuadrature constructor.

Implements NonDIntegration.

References Iterator::maxEvalConcurrency, NonDQuadrature::nestedRules, Analyzer::numContinuousVars, Non-
DQuadrature::numSamples, NonDQuadrature::quadMode, NonDQuadrature::reset(), NonDQuadrature::tpqDriver,
and NonDQuadrature::update().

void sampling_reset ( int min_samples, bool all_data_flag, bool stats_flag ) [protected], [virtual]

used by DataFitSurrModel::build_global() to publish the minimum number of points needed from the quadrature
routine in order to build a particular global approximation.

Reimplemented from Iterator.

References NonDQuadrature::compute_minimum_quadrature_order(), NonDIntegration::dimPrefSpec, Non-
DQuadrature::dimQuadOrderRef, NonDQuadrature::nestedRules, Analyzer::numContinuousVars, and NonDQuadrature-
::tpqDriver.

Referenced by NonDQuadrature::update().

int num_samples ( ) const [inline], [protected], [virtual]

Return current number of evaluation points. Since the calculation of samples, collocation points, etc. might be
costly, provide a default implementation here that backs out from the maxEvalConcurrency.

Reimplemented from Analyzer.

References NonDQuadrature::numSamples, NonDQuadrature::quadMode, and NonDQuadrature::tpqDriver.

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

- NonDQuadrature.hpp
- NonDQuadrature.cpp

13.103 NonDQUESOBayesCalibration Class Reference

Bayesian inference using the QUESO library from UT Austin.

Inheritance diagram for NonDQUESOBayesCalibration:
Public Member Functions

- **NonDQUESOBayesCalibration (ProblemDescDB &problem_db, Model &model)**
  
  *standard constructor*

- **~NonDQUESOBayesCalibration ()**

  *destructor*

Public Attributes

- **String mcmcType**
  
  *MCMC type (DRAM or Multilevel, both within QUESO)*

- **String rejectionType**
  
  *Rejection type (standard or delayed, in the DRAM framework)*

- **String metropolisType**
  
  *Metropolis type (hastings or adaptive, in the DRAM framework)*

- **int numSamples**
  
  *number of samples in the chain (e.g. number of MCMC samples)*

- **RealVector proposalCovScale**
  
  *scale factor for proposal covariance*

- **Real likelihoodScale**
  
  *scale factor for likelihood*

- **bool calibrateSigmaFlag**
  
  *flag to indicated if the sigma terms should be calibrated (default true)*

Protected Member Functions

- **void quantify_uncertainty ()**

  *redefined from DakotaNonD*
CHAPTER 13. CLASS DOCUMENTATION

Static Protected Member Functions

- static double dakotaLikelihoodRoutine (const QUESO::GslVector &paramValues, const QUESO::GslVector *paramDirection, const void *functionDataPtr, QUESO::GslVector *gradVector, QUESO::GslMatrix *hessianMatrix, QUESO::GslVector *hessianEffect)

  Likelihood function for callback from QUESO to DAKOTA for evaluation.

Protected Attributes

- int randomSeed

  random seed to pass to QUESO

Private Attributes

- short emulatorType

  the emulator type: NO_EMULATOR, GP_EMULATOR, PCE_EMULATOR, or SC_EMULATOR

Static Private Attributes

- static NonDQUESOBayesCalibration * NonDQUESOInstance

  Pointer to current class instance for use in static callback functions.

Additional Inherited Members

13.103.1 Detailed Description

Bayesian inference using the QUESO library from UT Austin.

This class provides a wrapper to the QUESO library developed as part of the Predictive Science Academic Alliance Program (PSAAP), specifically the PECOS (Predictive Engineering and Computational Sciences) Center at UT Austin. The name QUESO stands for Quantification of Uncertainty for Estimation, Simulation, and Optimization.

13.103.2 Constructor & Destructor Documentation

NonDQUESOBayesCalibration ( ProblemDescDB & problem_db, Model & model )

standard constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification.

13.103.3 Member Function Documentation

void quantify_uncertainty ( ) [protected], [virtual]

redefined from DakotaNonD

Perform the uncertainty quantification

Reimplemented from NonDBayesCalibration.

References NonDQUESOBayesCalibration::calibrateSigmaFlag, Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), Model::continuous_variables(), NonDQUESOBayesCalibration::dakotaLikelihoodRoutine(), NonDBayesCalibration::emulatorModel, NonDQUESOBayesCalibration::emulatorType, NonDCalibration::expData, NonDCalibration::expDataFileAnnotated, NonDCalibration::expDataFileName, Iterator::iterator_rep(),
13.104. NONDRELIABILITY CLASS REFERENCE

Base class for the reliability methods within DAKOTA/UQ.

Inheritance diagram for NonDReliability:

```
NonDReliability
  |     |
  |     | iterator
  |     | analyzer
  |     | NonD
  |     | NonDReliability
  |     | NonDGlobalReliability
  |     | NonDLocalReliability
```

Protected Member Functions

- **NonDReliability** (ProblemDescDB &problem_db, Model &model)
  \textit{constructor}
- \textit{~NonDReliability} ()
  \textit{destructor}
- void initialize_graphics (int iterator_server_id=1)
  \textit{initialize graphics customized for reliability methods}
- const Model & algorithm_space_model () const

Protected Attributes

- Model uSpaceModel
  \textit{Model representing the limit state in u-space, after any recastings and data fits.}
- Model mppModel
  \textit{RecastModel which formulates the optimization subproblem: RIA, PMA, EGO.}
- Iterator mppOptimizer

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

- NonDQUESOBayesCalibration.hpp
- NonDQUESOBayesCalibration.cpp

ExperimentsData::load_scalar(), NonDQUESOBayesCalibration::mcmcType, NonDQUESOBayesCalibration::metropolisType, NonDQUESOBayesCalibration::NonDQUESOInstance, Analyzer::numContinuousVars, NonDCalibration::numExpConfigVars, NonDCalibration::numExperiments, NonDCalibration::numExpStdDeviationsRead, Analyzer::numFunctions, NonDCalibration::numReplicates, NonDQUESOBayesCalibration::numSamples, Iterator::outputLevel, NonDQUESOBayesCalibration::proposalCovScale, NonDBayesCalibration::quantify_uncertainty(), NonDQUESOBayesCalibration::randomSeed, NonDQUESOBayesCalibration::rejectionType, ExperimentData::scalar_sigma(), NonDBayesCalibration::standardizedSpace, NonDBayesCalibration::stochExpIterator, and Dakota::strends().
Additional Inherited Members

13.104.1 Detailed Description

Base class for the reliability methods within DAKOTA/UQ.

The \texttt{NonDReliability} class provides a base class for \texttt{NonDLocalReliability}, which implements traditional MPP-based reliability methods, and \texttt{NonDGlobalReliability}, which implements global limit state search using Gaussian process models in combination with multimodal importance sampling.

13.104.2 Member Function Documentation

\texttt{const Model & algorithm\_space\_model() const} \[inline\], \[protected\], \[virtual\]

default definition that gets redefined in selected derived Minimizers

Reimplemented from \texttt{Analyzer}.

References \texttt{NonDReliability::uSpaceModel}.

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

- \texttt{NonDReliability.hpp}
- \texttt{NonDReliability.cpp}
13.105  NonDSampling Class Reference

Base class for common code between NonDLHSSampling, NonDIncremLHSSampling, and NonDAdaptImp Sampling.

Inheritance diagram for NonDSampling:

```
Inheritance Diagram for NonDSampling:

NonDSampling
  NonD
  Analyzer
  Iterator
```

Public Member Functions

- void compute_distribution_mappings (const IntResponseMap &samples)
  called by compute_statistics() to calculate CDF/CCDF mappings of z to p/beta and of p/beta to z
- void update_final_statistics ()
  update finalStatistics from minValues/maxValues, momentStats, and computedProbLevels/computedRelLevels/computed-RespLevels
- void print_pdf_mappings (std::ostream &s) const
  prints the PDFs computed in compute_statistics()

Protected Member Functions

- NonDSampling (ProblemDescDB &problem_db, Model &model)
  constructor
- NonDSampling (unsigned short method_name, Model &model, unsigned short sample_type, int samples, int seed, const String &rng, bool vary_pattern, short sampling_vars_mode)
  alternate constructor for sample generation and evaluation "on the fly"
- NonDSampling (unsigned short sample_type, int samples, int seed, const String &rng, const RealVector &lower_bnds, const RealVector &upper_bnds)
  alternate constructor for sample generation "on the fly"
- ~NonDSampling ()
  destructor
- int num_samples () const
- void sampling_reset (int min_samples, bool all_data_flag, bool stats_flag)
  resets number of samples and sampling flags
- void sampling_reference (int samples_ref)
  set reference number of samples, which is a lower bound during reset
- unsigned short sampling_scheme () const
  return sampleType
- void vary_pattern (bool pattern_flag)
set varyPattern

- void get_parameter_sets (Model &model)
  Uses lhsDriver to generate a set of samples from the distributions/bounds defined in the incoming model.
- void get_parameter_sets (const RealVector &lower_bnds, const RealVector &upper_bnds)
  Uses lhsDriver to generate a set of uniform samples over lower_bnds/upper_bnds.
- void update_model_from_sample (Model &model, const Real *sample_vars)
  Override default update of continuous vars only.
- void initialize_lhs (bool write_message)
  increments numLHSRuns, sets random seed, and initializes lhsDriver
- void compute_statistics (const RealMatrix &vars_samples, const IntResponseMap &resp_samples)
  For the input sample set, computes mean, standard deviation, and probability/reliability/response levels (aleatory uncertainties) or intervals (epistemic or mixed uncertainties)
- void compute_intervals (const IntResponseMap &samples)
  called by compute_statistics() to calculate min/max intervals
- void compute_moments (const IntResponseMap &samples)
  called by compute_statistics() to calculate means, std deviations, and confidence intervals
- void print_statistics (std::ostream &s) const
  prints the statistics computed in compute_statistics()
- void print_intervals (std::ostream &s) const
  prints the intervals computed in compute_intervals()
- void print_moments (std::ostream &s) const
  prints the moments computed in compute_moments()
- void view_design_counts (const Model &model, size_t &num_cdv, size_t &num_didv, size_t &num_dr dv)
  compute sampled subsets (all, active, uncertain) within all variables (acv/adiv/adrv) from samplingVarsMode and model
- void view_aleatory_uncertain_counts (const Model &model, size_t &num_cauv, size_t &num_diauv, size_t &num_drauv)
  compute sampled subsets (all, active, uncertain) within all variables (acv/adiv/adrv) from samplingVarsMode and model
- void view_epistemic_uncertain_counts (const Model &model, size_t &num_ceuv, size_t &num_dieuv, size_t &num_dreuv)
  compute sampled subsets (all, active, uncertain) within all variables (acv/adiv/adrv) from samplingVarsMode and model
- void view_uncertain_counts (const Model &model, size_t &num_cuv, size_t &num_diu v, size_t &num_druv)
  compute sampled subsets (all, active, uncertain) within all variables (acv/adiv/adrv) from samplingVarsMode and model
- void view_state_counts (const Model &model, size_t &num_csv, size_t &num_disv, size_t &num_drsv)
  compute sampled subsets (all, active, uncertain) within all variables (acv/adiv/adrv) from samplingVarsMode and model
- void mode_counts (const Model &model, size_t &cv_start, size_t &num_cv, size_t &div_start, size_t &num_div, size_t &drv_start, size_t &num_drv)
  compute sampled subsets (all, active, uncertain) within all variables (acv/adiv/adrv) from samplingVarsMode and model
Protected Attributes

- const int seedSpec
  the user seed specification (default is 0)
- int randomSeed
  the current seed
- const int samplesSpec
  initial specification of number of samples
- int samplesRef
  reference number of samples updated for refinement
- int numSamples
  the current number of samples to evaluate
- String rngName
  name of the random number generator
- unsigned short sampleType
  the sample type: default, random, lhs, incremental random, or incremental lhs
- Pecos::LHSDriver lhsDriver
  the C++ wrapper for the F90 LHS library
- bool statsFlag
  flags computation/output of statistics
- bool allDataFlag
  flags update of allResponses (allVariables or allSamples already defined)
- short samplingVarsMode
  the sampling mode: ALEATORY, EPISTEMIC, UNCERTAIN, ACTIVE, or ALL. This is a secondary control on top of the variables view that allows sampling over subsets of variables that may differ from the view.
- short sampleRanksMode
  mode for input/output of LHS sample ranks: IGNORE, GET, SET, or SET_GET
- bool varyPattern
  flag for generating a sequence of seed values within multiple get_parameter_sets() calls so that these executions (e.g., for SBO/SBNLS) are not repeated, but are still repeatable
- RealMatrix sampleRanks
  data structure to hold the sample ranks
- SensAnalysisGlobal nonDSampCorr
  initialize statistical post processing

Private Member Functions

- void archive_allocate_pdf ()
  allocate results array storage for pdf histograms
- void archive_pdf (size_t fn_index)
  archive a single pdf histogram for specified function
CHAPTER 13.  CLASS DOCUMENTATION

Private Attributes

- size_t numLHSRuns
  counter for number of executions of get_parameter_sets() for this object
- RealMatrix momentCIs
  Matrix of confidence internals on moments, with rows for mean_lower, mean_upper, sd_lower, sd_upper (calculated in compute_moments())
- RealMatrix extremeValues
  Minimum (row 0) and maximum (row 1) values of response functions for epistemic calculations (calculated in compute_intervals())
- RealVectorArray computedPDFAbscissas
  sorted response PDF intervals bounds extracted from min/max sample and requested/computedRespLevels (vector lengths = num bins + 1)
- RealVectorArray computedPDFOrdinates
  response PDF densities computed from bin counts divided by (unequal) bin widths (vector lengths = num bins)

Additional Inherited Members

13.105.1 Detailed Description

Base class for common code between NonDLHSSampling, NonDIncremLHSSampling, and NonDAdaptImp-Sampling.

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

1. The 1970’s vintage LHS (that had been f2c’d and converted to incomplete classes) has been removed.

13.105.2 Constructor & Destructor Documentation

NonDSampling ( ProblemDescDB & problem_db, Model & model )  [protected]

constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification.

References Dakota::abort_handler(), NonD::epistemicStats, NonD::initialize_final_statistics(), Iterator::max-EvalConcurrency, NonDSampling::numSamples, NonDSampling::sampleType, and NonD::totalLevelRequests.

NonDSampling ( unsigned short method_name, Model & model, unsigned short sample_type, int samples, int seed, const String & rng, bool vary_pattern, short sampling_vars_mode )  [protected]

alternate constructor for sample generation and evaluation ”on the fly”

This alternate constructor is used for generation and evaluation of on-the-fly sample sets.

References NonD::epistemicStats, Iterator::maxEvalConcurrency, NonD::numEpistemicUncVars, NonDSampling-::numSamples, NonDSampling::sampleType, NonDSampling::samplingVarsMode, and Iterator::subIteratorFlag.
NonDSampling ( unsigned short sample_type, int samples, int seed, const String & rng, const RealVector & lower_bnds, const RealVector & upper_bnds ) [protected]

alternate constructor for sample generation "on the fly"

This alternate constructor is used by ConcurrentStrategy for generation of uniform, uncorrelated sample sets.

References Iterator::maxEvalConcurrency, NonDSampling::numSamples, NonDSampling::sampleType, and Iterator::subIteratorFlag.

### 13.105.3 Member Function Documentation

```cpp
typedef int num_samples ( ) const [inline], [protected], [virtual]
```

Return current number of evaluation points. Since the calculation of samples, collocation points, etc. might be costly, provide a default implementation here that backs out from the maxEvalConcurrency.

Reimplemented from Analyzer.

References NonDSampling::numSamples.

Referenced by NonDAdaptImpSampling::evaluate_samples(), and NonDAdaptImpSampling::select_rep_points().

```cpp
void sampling_reset ( int min_samples, bool all_data_flag, bool stats_flag ) [inline], [protected], [virtual]
```

resets number of samples and sampling flags

used by DataFitSurrModel::build_global() to publish the minimum number of samples needed from the sampling routine (to build a particular global approximation) and to set allDataFlag and statsFlag. In this case, allDataFlag is set to true (vectors of variable and response sets must be returned to build the global approximation) and statsFlag is set to false (statistics computations are not needed).

Reimplemented from Iterator.

References NonDSampling::allDataFlag, NonDSampling::numSamples, NonDSampling::samplesRef, and NonDSampling::statsFlag.

```cpp
void get_parameter_sets ( Model & model ) [protected], [virtual]
```

Uses lhsDriver to generate a set of samples from the distributions/bounds defined in the incoming model.

This version of get_parameter_sets() extracts data from the user-defined model in any of the four sampling modes.

Reimplemented from Analyzer.

References Dakota::abort_handler(), Model::acv(), Model::adiv(), Model::aleatory_distribution_parameters(), Model::all_continuous_lower_bounds(), Model::all_continuous_upper_bounds(), Model::all_discrete_int_lower_bounds(), Model::all_discrete_int_upper_bounds(), Model::all_discrete_bank(), Analyzer::allSamples, Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), Model::current_variables(), Model::discrete_design_set_int_values(), Model::discrete_design_set_real_values(), Model::discrete_set_int_values(), Model::discrete_set_real_values(), Model::epistemic_distribution_parameters(), NonDSampling::initialize_lhs(), NonDSampling::lhsDriver, NonDSampling::mode_counts(), NonD::numContDesVars, NonD::numContStateVars, NonD::numDiscIntDesVars, NonD::numDiscIntStateVars, NonD::numDiscRealDesVars, NonD::numDiscRealStateVars, NonDSampling::numSamples, NonDSampling::samplingRanks, NonDSampling::samplingVarsMode, and Variables::view().

Referenced by NonDLHSSampling::NonDLHSSampling(), NonDLHSSampling::pre_run(), NonDIncremLHSSampling::quantify_uncertainty(), and NonDAdaptImpSampling::quantify_uncertainty().
void get_parameter_sets ( const RealVector & lower_bnds, const RealVector & upper_bnds ) [protected]

Uses lhsDriver to generate a set of uniform samples over lower_bnds/upper_bnds.

This version of get_parameter_sets() does not extract data from the user-defined model, but instead relies on the incoming bounded region definition. It only support a UNIFORM sampling mode, where the distinction of ACTIVE_UNIFORM vs. ALL_UNIFORM is handled elsewhere.

References Analyzer::allSamples, NonDSampling::initialize_lhs(), NonDSampling::lhsDriver, and NonDSampling::numSamples.

void view_design_counts ( const Model & model, size_t & num_cdv, size_t & num_didv, size_t & num_drdv ) const [protected]

compute sampled subsets (all, active, uncertain) within all variables (acv/adiv/adrv) from samplingVarsMode and model

This function computes total design variable counts, not active counts, for use in defining offsets and counts within all variables arrays.

References Model::all_continuous_variable_types(), Model::all_discrete_int_variable_types(), Model::all_discrete_real_variable_types(), Model::current_variables(), Variables::cv_start(), Variables::div_start(), Variables::drv_start(), NonD::numContDesVars, NonD::numDiscIntDesVars, NonD::numDiscRealDesVars, and Variables::view().

Referenced by NonDSampling::mode_counts().

void view_aleatory_uncertain_counts ( const Model & model, size_t & num_cauv, size_t & num_diauv, size_t & num_drauv ) const [protected]

compute sampled subsets (all, active, uncertain) within all variables (acv/adiv/adrv) from samplingVarsMode and model

This function computes total aleatory uncertain variable counts, not active counts, for use in defining offsets and counts within all variables arrays.

References Model::aleatory_distribution_parameters(), Model::current_variables(), Variables::cv(), Variables::div(), Variables::drv(), NonD::numContAleatUncVars, NonD::numDiscIntAleatUncVars, NonD::numDiscRealAleatUncVars, and Variables::view().

Referenced by NonDSampling::mode_counts().

void view_epistemic_uncertain_counts ( const Model & model, size_t & num_ceuv, size_t & num_dieuv, size_t & num_dreuv ) const [protected]

compute sampled subsets (all, active, uncertain) within all variables (acv/adiv/adrv) from samplingVarsMode and model

This function computes total epistemic uncertain variable counts, not active counts, for use in defining offsets and counts within all variables arrays.

References Model::current_variables(), Variables::cv(), Variables::div(), Variables::drv(), Model::epistemic_distribution_parameters(), NonD::numContEpistUncVars, NonD::numDiscIntEpistUncVars, NonD::numDiscRealEpistUncVars, and Variables::view().

Referenced by NonDSampling::mode_counts().

void view_uncertain_counts ( const Model & model, size_t & num_cuv, size_t & num_diuv, size_t & num_druv ) const [protected]

compute sampled subsets (all, active, uncertain) within all variables (acv/adiv/adrv) from samplingVarsMode and model
This function computes total uncertain variable counts, not active counts, for use in defining offsets and counts within all variables arrays.

References Model::aleatory_distribution_parameters(), Model::current_variables(), Variables::cv(), Variables::div(), Variables::drv(), Model::epistemic_distribution_parameters(), NonD::numContAleatUncVars, NonD::numContEpistUncVars, NonD::numDiscIntAleatUncVars, NonD::numDiscIntEpistUncVars, NonD::numDiscRealAleatUncVars, NonD::numDiscRealEpistUncVars, and Variables::view().

Referenced by NonDSampling::mode_counts().

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

- NonDSampling.hpp
- NonDSampling.cpp

13.106 NonDSparseGrid Class Reference

Derived nondeterministic class that generates N-dimensional Smolyak sparse grids for numerical evaluation of expectation integrals over independent standard random variables.

Inheritance diagram for NonDSparseGrid:

```
NonDSparseGrid
    NonDIntegration
        NonD
            Analyzer
                Iterator
```

Public Member Functions

- **NonDSparseGrid** (Model &model, short exp_coeffs_approach, const UShortArray &ssg_level_seq, const RealVector &dim_pref, short growth_rate=Pecos::MODERATE_RESTRICTED_GROWTH, short refine_control=Pecos::NO_CONTROL, bool track_uniq_prod_wts=true, bool track_colloc_indices=true)
- void increment_grid ()
- void increment_grid_weights (const RealVector &aniso_wts)
- void increment_specification_sequence ()
- const std::set<UShortArray> & active_multi_index () const
- const std::set<UShortArray> & old_multi_index () const
- void print_smolyak_multi_index () const
invokes SparseGridDriver::print_smolyak_multi_index()

- void initialize_sets()
  invokes SparseGridDriver::initialize_sets()

- void update_reference()
  invokes SparseGridDriver::update_reference()

- void increment_set(const UShortArray &set)
  invokes SparseGridDriver::push_trial_set()

- int increment_size() const
  invokes SparseGridDriver::unique_trial_points()

- void restore_set()
  invokes SparseGridDriver::restore_set()

- void evaluate_set()
  invokes SparseGridDriver::compute_trial_grid()

- void decrement_set()
  invokes SparseGridDriver::pop_trial_set()

- void update_sets(const UShortArray &set_star)
  invokes SparseGridDriver::update_sets()

- void print_final_sets(bool converged_within_tol)
  invokes SparseGridDriver::print_final_sets(bool)

- void finalize_sets()
  invokes SparseGridDriver::finalize_sets()

- void evaluate_grid_increment()
  invokes SparseGridDriver::evaluate_grid_increment()

- int num_samples() const

Protected Member Functions

- NonDSparseGrid(ProblemDescDB &problem_db, Model &model)
  constructor

- ~NonDSparseGrid()
  destructor

- void initialize_grid(const std::vector<Pecos::BasisPolynomial> &poly_basis)
  initialize integration grid by drawing from polynomial basis settings

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

- void reset()
  restore initial state for repeated sub-iterator executions

- void sampling_reset(int min_samples, bool all_data_flag, bool stats_flag)
13.106. NONDSPARSEGRID CLASS REFERENCE

Private Attributes

- Pecos::SparseGridDriver ∗ ssgDriver
  convenience pointer to the numIntDriver representation
- UShortArray ssgLevelSeqSpec
  the user specification for the Smolyak sparse grid level, defining a sequence of refinement levels.
- unsigned short ssgLevelRef
  reference point (e.g., lower bound) for the Smolyak sparse grid level maintained within ssgDriver

Additional Inherited Members

13.106.1 Detailed Description

Derived nondeterministic class that generates N-dimensional Smolyak sparse grids for numerical evaluation of expectation integrals over independent standard random variables.

This class is used by NonDPolynomialChaos and NonDStochCollocation, but could also be used for general numerical integration of moments. It employs 1-D Clenshaw-Curtis and Gaussian quadrature rules within Smolyak sparse grids.

13.106.2 Constructor & Destructor Documentation

NonDSparseGrid ( Model & model, short exp_coeffs_approach, const UShortArray & ssg_level_seq, const RealVector & dim_pref, short growth_rate = Pecos::MODERATE_RESTRICTED_GROWTH, short refine_control = Pecos::NO_CONTROL, bool track uniq prod wts = true, bool track colloc indices = true )

This alternate constructor is used for on-the-fly generation and evaluation of sparse grids within PCE and SC.

References NonDIntegration::numIntDriver, and NonDSparseGrid::ssgDriver.

NonDSparseGrid ( ProblemDescDB & problem_db, Model & model ) [protected]

constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification. It is not currently used, as there is not a separate sparse_grid method specification.

References Model::aleatory_distribution_parameters(), NonDIntegration::check_variables(), NonDIntegration::dimPrefSpec, ProblemDescDB::get_bool(), ProblemDescDB::get_short(), Iterator::iteratedModel, Iterator::max-EvalConcurrency, NonD::natafTransform, NonDIntegration::numIntDriver, Iterator::probDescDB, NonDSparseGrid::ssgDriver, and NonDSparseGrid::ssgLevelRef.

13.106.3 Member Function Documentation

int num_samples ( ) const [inline], [virtual]

Return current number of evaluation points. Since the calculation of samples, collocation points, etc. might be costly, provide a default implementation here that backs out from the maxEvalConcurrency.

Reimplemented from Analyzer.

References NonDSparseGrid::ssgDriver.
void sampling_reset ( int min_samples, bool all_data_flag, bool stats_flag ) [protected], [virtual]

used by DataFitSurrModel::build_global() to publish the minimum number of points needed from the sparse grid routine in order to build a particular global approximation.

Reimplemented from Iterator.

References NonDSparseGrid::ssgDriver, and NonDSparseGrid::ssgLevelRef.

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

* NonDSparseGrid.hpp
* NonDSparseGrid.cpp

13.107 NonDStochCollocation Class Reference

Nonintrusive stochastic collocation approaches to uncertainty quantification.

Inheritance diagram for NonDStochCollocation:

```
  NonDStochCollocation
   `-- NonDExpansion
      `-- NonD
         |    `-- Analyzer
```

Public Member Functions

- **NonDStochCollocation (ProblemDescDB &problem_db, Model &model)**
  *standard constructor*

- **NonDStochCollocation (Model &model, short exp_coeffs_approach, unsigned short num_int_level, short u_space_type, bool piecewise_basis, bool use_derivs)**
  *alternate constructor*

- **~NonDStochCollocation ()**
  *destructor*

Protected Member Functions

- **void resolve_inputs** (short &u_space_type, short &data_order)
  *perform error checks and mode overrides*

- **void initialize_u_space_model ()**
  *initialize uSpaceModel polynomial approximations with PCE/SC data*

- **void update_expansion ()**
  *update an expansion; avoids overhead in compute_expansion()  

```
• Real compute.covariance_metric ()
  compute 2-norm of change in response covariance
• Real compute.final_statistics_metric ()
  compute 2-norm of change in final statistics

Additional Inherited Members

13.107.1 Detailed Description

Nonintrusive stochastic collocation approaches to uncertainty quantification.

The NonDStochCollocation class uses a stochastic collocation (SC) approach to approximate the effect of parameter uncertainties on response functions of interest. It utilizes the InterpPolyApproximation class to manage multidimensional Lagrange polynomial interpolants.

13.107.2 Constructor & Destructor Documentation

NonDStochCollocation ( ProblemDescDB & problem_db, Model & model )

standard constructor

This constructor is called for a standard letter-envelope iterator instantiation using the ProblemDescDB.

References Model::assign_rep(), ParallelLibrary::command_line_check(), NonDExpansion::construct_expansion_sampler(), NonDExpansion::construct_quadrature(), NonDExpansion::construct_sparse_grid(), NonDExpansion::expansionBasisType, NonDExpansion::expansionCoeffsApproach, ProblemDescDB::get_bool(), ProblemDescDB::get_rv(), ProblemDescDB::get_short(), ProblemDescDB::get_string(), ProblemDescDB::get_usa(), NonDExpansion::initialize(), NonDStochCollocation::initialize_u_space_model(), Iterator::iteratedModel, NonDExpansion::nested_Rules, NonD::numContDesVars, NonD::numContEpistUncVars, NonD::numContStateVars, NonDExpansion::num_SamplesOnModel, Iterator::outputLevel, ProblemDescDB::parallel_library(), NonDExpansion::piecewiseBasis, Iterator::probDescDB, NonDExpansion::refineControl, NonDStochCollocation::resolve_inputs(), NonD::transform_model(), and NonDExpansion::uSpaceModel.

NonDStochCollocation ( Model & model, short exp_coeffs_approach, unsigned short num_int_level, short u_space_type, bool piecewise_basis, bool use_derivs )

alternate constructor

This constructor is used for helper iterator instantiation on the fly.

References Model::assign_rep(), NonDExpansion::construct_quadrature(), NonDExpansion::construct_sparse_grid(), NonDExpansion::expansionBasisType, NonDExpansion::expansionCoeffsApproach, NonDExpansion::initialize(), NonDStochCollocation::initialize_u_space_model(), Iterator::iteratedModel, NonD::numContDesVars, NonD::numContEpistUncVars, NonD::numContStateVars, Iterator::outputLevel, NonDExpansion::piecewiseBasis, NonDStochCollocation::resolve_inputs(), NonD::transform_model(), and NonDExpansion::uSpaceModel.

13.107.3 Member Function Documentation

Real compute.covariance_metric ( ) [protected], [virtual]

compute 2-norm of change in response covariance
computes the default refinement metric based on change in respCovariance
Reimplemented from NonDExpansion.
References Model::approximations(), NonDExpansion::compute_covariance_metric(), PecosApproximation::delta_covariance(), PecosApproximation::expansion_coefficient_flag(), NonDExpansion::expansionBasisType, NonDExpansion::initialPtU, NonD::numContDesVars, NonD::numContEpistUncVars, NonD::numContStateVars, Analyzer::numFunctions, NonDExpansion::respCovariance, and NonDExpansion::uSpaceModel.

**Real compute_final_statistics_metric ( ) [protected], [virtual]**

compute 2-norm of change in final statistics
computes a ”goal-oriented” refinement metric employing finalStatistics
Reimplemented from NonDExpansion.
References Model::approximations(), NonD::cdfFlag, NonDExpansion::compute_final_statistics_metric(), NonDExpansion::compute_statistics(), PecosApproximation::delta_beta(), PecosApproximation::delta_z(), PecosApproximation::expansion_coefficient_flag(), NonDExpansion::expansionBasisType, NonD::finalStatistics, Response::function_values(), NonDExpansion::initialPtU, Response::num_functions(), NonD::numContDesVars, NonD::numContEpistUncVars, NonD::numContStateVars, Analyzer::numFunctions, NonD::requestedGenRelLevels, NonD::requestedProbLevels, NonD::requestedRelLevels, NonD::requestedRespLevels, NonD::respLevelTarget, and NonDExpansion::uSpaceModel.

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

- NonDStochCollocation.hpp
- NonDStochCollocation.cpp

### 13.108 NonlinearCGOptimizer Class Reference

Inheritance diagram for NonlinearCGOptimizer:

```
NonlinearCGOptimizer
  ^
  |  
  v
Minimizer
  ^
  |  
  v
  
  Optimizer
    ^
    |  
    v
    
  Iterator
```

**Public Member Functions**

- **NonlinearCGOptimizer (ProblemDescDB &problem_db, Model &model)**
  
  *standard constructor*

- **~NonlinearCGOptimizer ()**
  
  *destructor*

- **Real linesearch_eval (const Real &trial_step, short req_val=1)**
  
  *evaluate the objective function given a particular step size (public for use in boost_Js_eval functor; could use friend)*
Protected Member Functions

- void find_optimum ()
  
  Used within the optimizer branch for computing the optimal solution. Redefines the run virtual function for the optimizer branch.

Private Member Functions

- void parse_options ()
  
  constructor helper function to parse misc_options from ProblemDescDB

- void compute_direction ()
  
  compute next direction via choice of method

- bool compute_step ()
  
  compute step: fixed, simple decrease, sufficient decrease

- void bracket_min (Real &xa, Real &xb, Real &xc, Real &fa, Real &fb, Real &fc)
  
  bracket the 1-D minimum in the linesearch

- Real brent_minimize (Real a, Real b, Real tol)
  
  Perform 1-D minimization for the stepLength using Brent's method.

Private Attributes

- Real initialStep
  
  initial step length

- Real linesearchTolerance
  
  approximate accuracy of absissa in LS

- unsigned linesearchType
  
  type of line search (if any)

- unsigned maxLinesearchIters
  
  maximum evaluations in line search

- Real relFunctionTol
  
  stopping criterion for rel change in fn

- Real relGradientTol
  
  stopping criterion for rel reduction in g

- bool resetStep
  
  whether to reset step with each linesearch

- unsigned restartIter
  
  iter at which to reset to steepest descent

- unsigned updateType
  
  type of CG direction update

- unsigned iterCurr
  
  current iteration number

- RealVector designVars
  
  current decision variables in the major iteration

- RealVector trialVars
  
  decision variables in the linesearch
• Real functionCurr
current function value
• Real functionPrev
previous function value
• RealVector gradCurr
current gradient
• RealVector gradPrev
previous gradient
• RealVector gradDiff
temporary for gradient difference (gradCurr - gradPrev)
• RealVector searchDirection
current aggregate search direction
• Real stepLength
current step length parameter alpha
• Real gradDotGrad_init
initial gradient norm squared
• Real gradDotGrad_curr
gradCurr dot gradCurr
• Real gradDotGrad_prev
gradPrev dot gradPrev

Additional Inherited Members

13.108.1 Detailed Description
Experimental implementation of nonlinear CG optimization

13.108.2 Member Function Documentation

Real brent_minimize ( Real a, Real b, Real tol ) [private]
Perform 1-D minimization for the stepLength using Brent’s method.

Perform 1-D minimization for the stepLength using Brent’s method. This is a C translation of fmin.f from
Netlib.
References NonlinearCGOptimizer::linesearch_eval(), NonlinearCGOptimizer::maxLinesearchIters, and Iterator-::outputLevel.
Referenced by NonlinearCGOptimizer::compute_step().
The documentation for this class was generated from the following files:

• NonlinearCGOptimizer.hpp
• NonlinearCGOptimizer.cpp
13.109 NPSOLOptimizer Class Reference

Wrapper class for the NPSOL optimization library.

Inheritance diagram for NPSOLOptimizer:

```
NPSOLOptimizer

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Iterator</td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td>Minimizer</td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td>Optimizer</td>
</tr>
<tr>
<td>SOLBase</td>
</tr>
<tr>
<td>----------------</td>
</tr>
</tbody>
</table>

NPSOLOptimizer
```

Public Member Functions

- **NPSOLOptimizer (ProblemDescDB &problem_db, Model &model)**
  
  *standard constructor*

- **NPSOLOptimizer (Model &model)**
  
  *alternate constructor for Iterator instantiations by name*

- **NPSOLOptimizer (Model &model, const int &derivative_level, const Real &conv_tol)**
  
  *alternate constructor for instantiations "on the fly"*

- **NPSOLOptimizer (const RealVector &initial_point, const RealVector &var_lower_bnds, const RealVector &var_upper_bnds, const RealMatrix &lin_ineq_coeffs, const RealVector &lin_ineq_lower_bnds, const RealVector &lin_ineq_upper_bnds, const RealMatrix &lin_eq_coeffs, const RealVector &lin_eq_targets, const RealVector &nonlin_ineq_lower_bnds, const RealVector &nonlin_ineq_upper_bnds, const RealVector &nonlin_eq_targets, void(*user_obj_eval)(int &, int &, double *, double &, double *, int &), void(*user_con_eval)(int &, int &, int &, int &, int *, double *, double *, double *, int &), const int &derivative_level, const Real &conv_tol)**

  *alternate constructor for instantiations "on the fly"*

- **~NPSOLOptimizer ()**

  *destructor*

- **void find_optimum ()**

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

Private Member Functions

- **void find_optimum_on_model ()**

  *called by find_optimum for setUpType == "model"*

- **void find_optimum_on_user_functions ()**

  *called by find_optimum for setUpType == "user functions"*
CHAPTER 13. CLASS DOCUMENTATION

Static Private Member Functions

- static void objective_eval (int &mode, int &n, double *x, double &f, double *gradf, int &nstate)

  OBJFUN in NPSOL manual: computes the value and first derivatives of the objective function (passed by function pointer to NPSOL).

Private Attributes

- String setUpType

  controls iteration mode: "model" (normal usage) or "user functions" (user-supplied functions mode for "on the fly" instantiations). NonDReliability currently uses the user functions mode.

- RealVector initialPoint

  holds initial point passed in for "user functions" mode.

- RealVector lowerBounds

  holds variable lower bounds passed in for "user functions" mode.

- RealVector upperBounds

  holds variable upper bounds passed in for "user functions" mode.

- void(* userObjectiveEval )(int &, int &, double *, double &, double *, int &)

  holds function pointer for objective function evaluator passed in for "user functions" mode.

- void(* userConstraintEval )(int &, int &, int &, int &, int *, int *, double *, double *, double *, int &)

  holds function pointer for constraint function evaluator passed in for "user functions" mode.

Static Private Attributes

- static NPSOLOptimizer * npsolInstance

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

Additional Inherited Members

13.109.1 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, outputverbosity, verify_level, function_precision, and linesearch_tolerance are mapped into NPSOL’s "Major Iteration Limit", "Optimality Tolerance", "Major Print Level" (verbose: Major Print Level = 20; quiet: Major Print Level = 10), "Verify Level", "Function Precision", and "Linesearch Tolerance" parameters, respectively, using NPSOL’s npoptn() subroutine (as wrapped by npoptn2() from the npoptn_wrapper.f file). Refer to [Gill, P.E., Murray, W., Saunders, M.A., and Wright, M.H., 1986] for information on NPSOL’s optional input parameters and the npoptn() subroutine.
### 13.109.2 Constructor & Destructor Documentation

**NPSOLOptimizer ( ProblemDescDB & problem_db, Model & model )**

standard constructor

This is the primary constructor. It accepts a Model reference.

References Minimizer::constraintTol, Iterator::convergenceTol, Model::fd_gradient_step_size(), ProblemDescDB::get_int(), ProblemDescDB::get_real(), Model::gradient_type(), Iterator::iteratedModel, Iterator::maxIterations, Iterator::outputLevel, Iterator::probDescDB, SOLBase::set_options(), Minimizer::speculativeFlag, and Minimizer::vendorNumericalGradFlag.

**NPSOLOptimizer ( Model & model )**

alternate constructor for Iterator instantiations by name

This is an alternate constructor which accepts a Model but does not have a supporting method specification from the ProblemDescDB.

References Minimizer::constraintTol, Iterator::convergenceTol, Model::fd_gradient_step_size(), Model::gradient_type(), Iterator::iteratedModel, Iterator::maxIterations, Iterator::outputLevel, SOLBase::set_options(), Minimizer::speculativeFlag, and Minimizer::vendorNumericalGradFlag.

**NPSOLOptimizer ( Model & model, const int & derivative_level, const Real & conv_tol )**

alternate constructor for instantiations "on the fly"

This is an alternate constructor for instantiations on the fly using a Model but no ProblemDescDB.

**NPSOLOptimizer ( const RealVector & initial_point, const RealVector & var_lower_bnds, const RealVector & var_upper_bnds, const RealMatrix & lin_ineq_coeffs, const RealVector & lin_ineq_lower_bnds, const RealVector & lin_ineq_upper_bnds, const RealMatrix & lin_eq_coeffs, const RealVector & lin_eq_targets, const RealVector & nonlin_ineq_lower_bnds, const RealVector & nonlin_ineq_upper_bnds, const RealVector & nonlin_eq_targets, void(*)(int &, int &, double *, double &, double *, int &) user_obj_eval, void(*)(int &, int &, int &, int &, int *, double *, double *, double *, int &) user_con_eval, const int & derivative_level, const Real & conv_tol )**

alternate constructor for instantiations "on the fly"

This is an alternate constructor for performing an optimization using the passed in objective function and constraint function pointers.

References SOLBase::allocate_arrays(), SOLBase::allocate_workspace(), SOLBase::augment_bounds(), NPSOLOptimizer::lowerBounds, Minimizer::numContinuousVars, Minimizer::numLinearConstraints, Minimizer::numNonlinearConstraints, and NPSOLOptimizer::upperBounds.

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

- NPSOLOptimizer.hpp
- NPSOLOptimizer.cpp

### 13.110 OptDartsOptimizer Class Reference

Wrapper class for OptDarts Optimizer.

Inheritance diagram for OptDartsOptimizer:
CHAPTER 13. CLASS DOCUMENTATION

Public Member Functions

- **OptDartsOptimizer (ProblemDescDB &problem_db, Model &model)**
  Constructor.
- **OptDartsOptimizer (Model &model)**
  alternate constructor for Iterator instantiations by name
- **~OptDartsOptimizer ()**
  Destructor.
- **void find optimum ()**
  Calls the OptDarts algorithm.

Private Member Functions

- **void load_parameters (Model &model)**
  Convenience function for Parameter loading.
- **double opt_darts_f ()**
  Function evaluation.
- **void opt_darts_execute (size_t num_dim, size_t budget, double *xmin, double *xmax, double TOL, size_t problem_index, double fw_MC, double fb_MC)**
  Run the OPT-DARTS method.
- **void opt_darts_initiate (double *xmin, double *xmax)**
  Initialize OPT-DARTS.
- **void opt_darts_reset_convex_hull ()**
- **size_t opt_darts_pick_candidate (size_t ifunc)**
  Choose the next trial iterate.
- **void retrieve_extended_neighbors (size_t icandidate)**
- **void opt_darts_sample_from_candidate_neighborhood (size_t icandidate, size_t ifunc)**
- **void DIRECT_sample_from_candidate_neighborhood (size_t icandidate)**
- **void opt_darts_add_dart ()**
- **void opt_darts_update_K_h_approximate_Voronoi (size_t isample)**
- **void opt_darts_terminate ()**
  Release memory and exit cleanly.
- **void opt_darts_plot_discs_2d (size_t icandidate)**
  Convenience function for plotting iterates.
• void opt_darts_plot_hull_2d (size_t icandidate, size_t ifunc)
  Convenience function for plotting convex hull.
• void initiate_random_generator (unsigned long x)
• double generate_a_random_number ()
• void sample_uniformly_from_unit_sphere_surface (double *dart, size_t num_dim)
• bool trim_line_using_Hyperplane (size_t num_dim, double *st, double *end, double *qH, double *nH)

Private Attributes
• double * xmin
• double * xmax
• double * dart
• double * st
• double * end
• double * _tmp_point
• double * _qH
• double * _nH
• double ** _x
• double ** _xc
• double ** _f
• double ** _K
• double * _h
• double * _r
• size_t ** _neighbors
• size_t * _tmp_neighbors
• size_t * _ext_neighbors
• size_t _num_ext_neighbors
• bool _use_opt_darts
• bool _estimate_K
• size_t _ib
• size_t _num_samples
• size_t _budget
• size_t _num_dim
• double _diag
• size_t _problem_index
• double _fb
• double _fw
• double _fval
• size_t _corner_index
• size_t _num_corners
• size_t * _corners
• double _epsilon
• double _fb_MC
• double _fw_MC
• double ** _xm
• double ** _xp
Additional Inherited Members

13.110.1 Detailed Description

Wrapper class for OptDarts Optimizer.

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

- OptDartsOptimizer.hpp
- OptDartsOptimizer.cpp

13.111 Optimizer Class Reference

Base class for the optimizer branch of the iterator hierarchy.

Inheritance diagram for Optimizer:
Static Public Member Functions

- static void **not available** (const std::string &package_name)

  Static helper function: third-party opt packages which are not available.

Protected Member Functions

- **Optimizer ()**
  
  default constructor

- **Optimizer (ProblemDescDB &problem_db, Model &model)**

  alternate constructor; accepts a model

- **Optimizer (unsigned short method_name, Model &model)**

  alternate constructor for "on the fly" instantiations

- **Optimizer (unsigned short method_name, size_t num_cv, size_t num_div, size_t num_drv, size_t num_lin_ineq, size_t num_lin_eq, size_t num_nln_ineq, size_t num_nln_eq)**

  alternate constructor for "on the fly" instantiations

- **~Optimizer ()**

  destructor
• void initialize_run()
• void core_run()
  
  *core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post*
• void post_run (std::ostream &s)
• void finalize_run()
  
  *utility function to perform common operations following post_run(); deallocation and resetting of instance pointers*
• void print_results (std::ostream &s)
• virtual void find_optimum ()=0
  
  *Used within the optimizer branch for computing the optimal solution. Redefines the run virtual function for the optimizer branch.*

**Protected Attributes**

• size_t numObjectiveFns
  
  *number of objective functions (iterator view)*
• bool localObjectiveRecast
  
  *flag indicating whether local recasting to a single objective is used*
• Optimizer * prevOptInstance
  
  *pointer containing previous value of optimizerInstance*

**Static Protected Attributes**

• static Optimizer * optimizerInstance
  
  *pointer to Optimizer instance used in static member functions*

**Private Member Functions**

• void reduce_model (bool local_nls_recast, bool require_hessians)
  
  *Wrap iteratedModel in a RecastModel that performs (weighted) multi-objective or sum-of-squared residuals transformation.*
• void objective_reduction (const Response &full_response, const BoolDeque &sense, const RealVector &full_wts, Response &reduced_response) const
  
  *forward mapping: maps multiple primary response functions to a single weighted objective for single-objective optimizers*
• void local_objective_recast_retrieve (const Variables &vars, Response &response) const
  
  *infers MOO/NLS solution from the solution of a single-objective optimizer*

**Static Private Member Functions**

• static void primary_resp_reducer (const Variables &full_vars, const Variables &reduced_vars, const Response &full_response, Response &reduced_response)
  
  *Recast callback to reduce multiple objectives or residuals to a single objective, with gradients and Hessians as needed.*
Additional Inherited Members

13.111.1 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.

13.111.2 Member Function Documentation

void initialize_run( ) [protected], [virtual]

Implements portions of initialize_run specific to Optimizers. This function should be invoked (or reimplemented) by any derived implementations of initialize_run() (which would otherwise hide it).

Reimplemented from Minimizer.

Reimplemented in SNLLOptimizer, NLPLPOptimizer, and DOTOptimizer.

References Minimizer::initialize_run(), Iterator::iteratedModel, Minimizer::minimizerRecasts, Optimizer::optimizerInstance, Optimizer::prevOptInstance, and Model::update_from_subordinate_model().

Referenced by DOTOptimizer::initialize_run(), CONMINOptimizer::initialize_run(), NLPLPOptimizer::initialize_run(), and SNLLOptimizer::initialize_run().

void core_run( ) [inline], [protected], [virtual]

core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post Virtual run function for the iterator class hierarchy. All derived classes need to redefine it.

Reimplemented from Iterator.

References Optimizer::find_optimum().

void post_run( ostream & s ) [protected], [virtual]

Implements portions of post_run specific to Optimizers. This function should be invoked (or reimplemented) by any derived implementations of post_run() (which would otherwise hide it).

Reimplemented from Minimizer.

Reimplemented in SNLLOptimizer.

References Dakota::abort_handler(), Response::active_set_request_vector(), Iterator::bestResponseArray, Iterator::bestVariablesArray, Variables::continuous_variables(), Response::copy(), Minimizer::cvScaleMultipliers, Minimizer::cvScaleOffsets, Minimizer::cvScaleTypes, Minimizer::expData, Response::function_value(), Response::function_values(), Optimizer::local_objective_recast_retrieve(), Optimizer::localObjectiveRecast, Minimizer::modify_s2n(), Minimizer::need_resp_trans_byvars(), Minimizer::numExperiments, Minimizer::numNonlinearConstraints, Minimizer::numReplicates, Minimizer::numUserPrimaryFns, Minimizer::obsDataFlag, Minimizer::post_run(), Minimizer::primaryRespScaleFlag, Minimizer::response_modify_s2n(), ExperimentData::scalar_data(), Minimizer::secondaryRespScaleFlag, Response::update_partial(), and Minimizer::varsScaleFlag.

Referenced by COLINOptimizer::post_run(), and SNLLOptimizer::post_run().

void finalize_run( ) [inline], [protected], [virtual]

utility function to perform common operations following post_run(); deallocation and resetting of instance pointers Optional: perform finalization phases of run sequence, like deallocating memory and resetting instance pointers. Commonly used in sub-iterator executions. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s finalize_run(), typically after performing its own implementation steps.

Reimplemented from Minimizer.
Reimplemented in SNLLOptimizer.
References Minimizer::finalize_run(), Optimizer::optimizerInstance, and Optimizer::prevOptInstance.
Referenced by SNLLOptimizer::finalize_run().

void print_results ( std::ostream & s ) [protected], [virtual]
Redefines default iterator results printing to include optimization results (objective functions and constraints).
Reimplemented from Iterator.
References Dakota::abort_handler(), Minimizer::archive_allocate_best(), Minimizer::archive_best(), Iterator-
::bestResponseArray, Iterator::bestVariablesArray, Dakota::data_pairs, Model::interface_iD(), Iterator::iteratedModel, Dakota::lookup_by_val(), Minimizer::numContinuousVars, Minimizer::numFunctions, Minimizer::numNonlinear-
Constraints, Minimizer::optimizationFlag, Model::primary_response_fn_weights(), Model::subordinate_model(), and Dakota::write_precision.

void reduce_model ( bool local_nls_recast, bool require_hessians ) [private]
Wrap iteratedModel in a RecastModel that performs (weighted) multi-objective or sum-of-squared residuals transform-
ation.
Reduce model for least-squares or multi-objective transformation. Doesn’t map variables, or secondary re-
sponses. Maps active set for Gauss-Newton. Maps primary responses to single objective so user vs. iterated matters.
References Iterator::activeSet, Model::assign_rep(), Model::current_response(), Minimizer::gnewton_set_recast(), Model::hessian_type(), Iterator::iteratedModel, Minimizer::numContinuousVars, Minimizer::numNonlinearConstraints, Minimizer::numNonlinearIneqConstraints, Optimizer::numObjectiveFns, Minimizer::numRowsExpData, Minimizer-
::numUserPrimaryFns, Minimizer::obsDataFlag, Iterator::outputLevel, Optimizer::primary_resp_reducer(), Model-
::primary_response_fn_sense(), Model::primary_response_fn_weights(), ActiveSet::request_vector(), Response::reshape(), and Minimizer::secondary_resp_copier().
Referenced by Optimizer::Optimizer().

void primary_resp_reducer ( const Variables & full_vars, const Variables & reduced_vars, const Response & full_response, Response & reduced_response ) [static], [private]
Recast callback to reduce multiple objectives or residuals to a single objective, with gradients and Hessians as needed.
Objective function map from multiple primary responses (objective or residuals) to a single objective. Cur-
tently supports weighted sum; may later want more general transformations, e.g., goal-oriented
References Iterator::iteratedModel, Optimizer::objective_reduction(), Optimizer::optimizerInstance, Iterator-
::outputLevel, Model::primary_response_fn_sense(), Model::primary_response_fn_weights(), and Model::subordinate-
model().
Referenced by Optimizer::reduce_model().

void objective_reduction ( const Response & full_response, const BoolDeque & sense, const RealVector & full_wts, Response & reduced_response ) const [private]
forward mapping: maps multiple primary response functions to a single weighted objective for single-objective optimizers
This function is responsible for the mapping of multiple objective functions into a single objective for pub-
ishing to single-objective optimizers. Used in DOTOptimizer, NPSOLOptimizer, SNLLOptimizer, and SGO-
PTApplication on every function evaluation. The simple weighting approach (using primaryRespFnWts) is the
only technique supported currently. The weightings are used to scale function values, gradients, and Hessians as needed.

References Response::active_set_request_vector(), Response::function_gradient_view(), Response::function_grads(), Response::function_hessian_view(), Response::function_hessians(), Response::function_value(), Response::function_values(), Response::num_functions(), Minimizer::numConstraints, Minimizer::objective(), Minimizer::objective_gradient(), Minimizer::objective_hessian(), Iterator::outputLevel, and Dakota::write_precision.

Referenced by Optimizer::primary_res_reducer().

void local_objective_recast_retrieve ( const Variables & vars, Response & response ) const [private]
infers MOO/NLS solution from the solution of a single-objective optimizer

Retrieve a MOO/NLS response based on the data returned by a single objective optimizer by performing a

References Response::active_set(), Dakota::data_pairs, Response::function_value(), Model::interface_id(), Iterator::iteratedModel, Dakota::lookup_by_val(), Minimizer::numRowsExpData, Minimizer::numUserPrimaryFns, Minimizer::obsDataFlag, ActiveSet::reshape(), and Response::update().

Referenced by Optimizer::post_run().

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

- DakotaOptimizer.hpp
- DakotaOptimizer.cpp

13.112 OutputManager Class Reference

Class to manage redirection of stdout/stderr, and keep track of current redir state, and manage rank 0 output.

Public Member Functions

- **OutputManager ()**
  Default constructor (needed for default environment ctors)

- **OutputManager (const ProgramOptions &prog_opts, int dakota_world_rank=0, bool dakota_mpirun_flag=false)**
  Standard constructor, taking user-specified program options and optionally taking the rank of this process in
  Dakota’s MPI_Comm.

- **~OutputManager ()**
  Destructor that closes streams and other outputs.

- **void close_streams ()**
  helper to close streams during destructor or abnormal abort

- **void parse (const ProblemDescDB &problem_db)**
  Extract environment options from ProblemDescDB.

- **void startup_message (const String &start_msg)**
  Set the Dakota startup message (“Running on...”)

- **void file_tag (const String &iterator_tag)**
  Update the tag to use on files.

- **void redirect_cout (const ProgramOptions &prog_opts, bool force_cout_redirect=false)**
  Redirect cout based on program options filenames and force flag.

- **void redirect_cerr (const ProgramOptions &prog_opts)**
Redirect cerr based on program options filenames only.

- void init_resultsdb (ProgramOptions &prog_opts)
  Initialize results DB based on problem DB.
- void init_restart (const ProgramOptions &prog_opts)
  Initialize restart DB based on program options filenames.
- void output_version (std::ostream &os=Cout) const
  Output the current Dakota version.
- void output_startup_message (std::ostream &os=Cout) const
  Output the startup header and time.
- void output_helper (const String &message, std::ostream &os) const
  Output only on Dakota world rank 0 (for version, help, etc.)
- void append_restart (const ParamResponsePair &prp)
  Append a parameter/response set to the restart file

Public Attributes

- bool graph2DFlag
  flag for using 2D graphics plots
- bool tabularDataFlag
  flag for file tabulation of graphics data
- bool resultsOutputFlag
  whether to output results data
- String tabularDataFile
  filename for tabulation of graphics data
- String resultsOutputFile
  filename for results data

Private Member Functions

- void redirect_cout (const String &new_filename)
  implementation of cout redirection
- void redirect_cerr (const String &new_filename)
  implementation of cerr redirection
- void read_write_restart (bool read_restart_flag, const String &read_restart_filename, size_t stop_restart_eval, const String &write_restart_filename)
  conditionally import evaluations from restart file, then always create or overwrite restart file

Private Attributes

- int worldRank
  output manager handles rank 0 only output when needed
- bool mpirunFlag
  some output is only for MPI runs
- String fileTag
  tag for various input/output files (default none)
13.113. PARALLELCONFIGURATION CLASS REFERENCE

- **String** `startupMessage`
  
  message to print at startup when proceeding to instantiate objects

- **String** `coutFilename`
  
  current (potentially tagged) filename for output, `ProgramOptions` has root

- **String** `cerrFilename`
  
  current (potentially tagged) filename for error, `ProgramOptions` has root

- **std::ofstream** `output_ofstream`
  
  tagged file redirection of stdout

- **std::ofstream** `error_ofstream`
  
  tagged file redirection of stderr

- **std::ofstream** `restartOutputFS`
  
  Binary stream to which restart data is written.

- **boost::archive::binary_oarchive** `restartOutputArchive`
  
  Binary output archive to which data is written (ptr as no default ctor)

13.112.1 Detailed Description

Class to manage redirection of stdout/stderr, and keep track of current redir state, and manage rank 0 output.

13.112.2 Constructor & Destructor Documentation

`OutputManager ( const ProgramOptions & prog_opts, int dakota_world_rank = 0, bool dakota_mpirun_flag = false )`

Standard constructor, taking user-specified program options and optionally taking the rank of this process in Dakota’s MPI_Comm.

Only get minimal information off `ProgramOptions` as may be updated later by broadcast.

References `OutputManager::mpirunFlag`, `ProgramOptions::output_file()`, `OutputManager::redirect_cout()`, Dakota::start_dakota_heartbeat(), `ProgramOptions::user_stdout_redirect()`, and `OutputManager::worldRank`

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

- `OutputManager.hpp`
- `OutputManager.cpp`

13.113 ParallelConfiguration Class Reference

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

**Public Member Functions**

- **ParallelConfiguration ()**
  
  default constructor

- **ParallelConfiguration (const ParallelConfiguration &pl)**
  
  copy constructor

- **~ParallelConfiguration ()**
  
  destructor
• **ParallelConfiguration & operator= (const ParallelConfiguration &pl)**
  assignment operator

• **const ParallelLevel & w_parallel_level () const**
  return the ParallelLevel corresponding to wPLIter

• **const ParallelLevel & si_parallel_level () const**
  return the ParallelLevel corresponding to siPLIter

• **const ParallelLevel & ie_parallel_level () const**
  return the ParallelLevel corresponding to iePLIter

• **const ParallelLevel & ea_parallel_level () const**
  return the ParallelLevel corresponding to eaPLIter

**Private Member Functions**

• **void assign (const ParallelConfiguration &pl)**
  assign the attributes of the incoming pl to this object

**Private Attributes**

• **short numParallelLevels**
  number of parallel levels

• **ParLevLIter wPLIter**
  list iterator for MPI_COMM_WORLD (not strictly required, but improves modularity by avoiding explicit usage of MPI_COMM_WORLD)

• **ParLevLIter siPLIter**
  list iterator for concurrent iterator partitions (there may be more than one per parallel configuration instance)

• **ParLevLIter iePLIter**
  list iterator identifying the iterator-evaluation parallelLevel (there can only be one)

• **ParLevLIter eaPLIter**
  list iterator identifying the evaluation-analysis parallelLevel (there can only be one)

**Friends**

• class **ParallelLibrary**
  the ParallelLibrary class has special access priveleges in order to streamline implementation

### 13.113.1 Detailed Description

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

Rather than containing the multilevel parallel configuration directly, ParallelConfiguration instead provides a set of list iterators which point into a combined list of ParallelLevels. This approach allows different configurations to reuse ParallelLevels without copying them. A list of ParallelConfigurations is contained in ParallelLibrary (ParallelLibrary::parallelConfigurations).

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

• ParallelLibrary.hpp
13.114 ParallelDirectApplicInterface Class Reference

Sample derived interface class for testing parallel simulator plug-ins using \texttt{assign\_rep()}. Inheritance diagram for ParallelDirectApplicInterface:

![Inheritance diagram](image)

**Public Member Functions**

- \texttt{ParallelDirectApplicInterface} (const Dakota::ProblemDescDB &problem\_db, const MPI\_Comm &analysis\_comm)
  - constructor
- \texttt{\sim ParallelDirectApplicInterface} ()
  - destructor

**Protected Member Functions**

- int \texttt{derived\_map\_ac} (const Dakota::String &ac\_name)
  - execute an analysis code portion of a direct evaluation invocation
- void \texttt{derived\_map\_asynch} (const Dakota::ParamResponsePair &pair)
  - no-op hides base error; job batching occurs within \texttt{wait\_local\_evaluations()}
- void \texttt{wait\_local\_evaluations} (Dakota::PRPQueue &prp\_queue)
  - evaluate the batch of jobs contained in prp\_queue
- void \texttt{test\_local\_evaluations} (Dakota::PRPQueue &prp\_queue)
  - invokes \texttt{wait\_local\_evaluations()} (no special nowait support)
- void \texttt{set\_communicators\_checks} (int max\_eval\_concurrency)
  - no-op hides default run-time error checks at DirectApplicInterface level

**Private Member Functions**

- int \texttt{text\_book} (const Dakota::RealVector &c\_vars, const Dakota::ShortArray &asv, Dakota::RealVector &fn\_vals, Dakota::RealMatrix &fn\_grads, Dakota::RealSymMatrixArray &fn\_hessians)
  - demo evaluator function for parallel plug-ins
Additional Inherited Members

13.114.1 Detailed Description

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

The plug-in ParallelDirectApplicInterface resides in namespace SIM and uses a copy of textbook() to perform parallel parameter to response mappings. It is used to demonstrate plugging in a parallel direct analysis driver into Dakota in library mode. Test input files can then use an analysis driver of "plugin_textbook".

13.114.2 Member Function Documentation

void test_local_evaluations ( Dakota::PRPQueue & prp_queue ) [inline],[protected]

invokes wait_local_evaluations() (no special nowait support)

For use by ApplicationInterface::serve_evaluations_asynch(), which can provide a batch processing capability within message passing schedulers (called using chain ApplicationInterface::serve() from Model::serve() from IteratorScheduler::run_iterator()).

References ParallelDirectApplicInterface::wait_local_evaluations().

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

- PluginParallelDirectApplicInterface.hpp
- PluginParallelDirectApplicInterface.cpp

13.115 ParallelLevel Class Reference

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

Public Member Functions

- ParallelLevel ()
  
  default constructor
- ParallelLevel (const ParallelLevel &pl)
  
  copy constructor
- ~ParallelLevel ()
  
  destructor
- ParallelLevel & operator= (const ParallelLevel &pl)
  
  assignment operator
- bool dedicated_master () const
  
  return dedicatedMasterFlag
- bool communicator_split () const
  
  return commSplitFlag
- bool server_master () const
  
  return serverMasterFlag
- bool message_pass () const
  
  return messagePass
- bool idle_partition () const
  
  return idlePartition
- int num_servers () const
return numServers
• int processors_per_server () const
  return procsPerServer
• int processor_remainder () const
  return procRemainder
• const MPI_Comm & server_intra_communicator () const
  return serverIntraComm
• int server_communicator_rank () const
  return serverCommRank
• int server_communicator_size () const
  return serverCommSize
• const MPI_Comm & hub_server_intra_communicator () const
  return hubServerIntraComm
• int hub_server_communicator_rank () const
  return hubServerCommRank
• 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
• 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
• bool idlePartition
  identifies presence of an idle processor < partition at this level
• int numServers
  number of servers
• int procsPerServer
  processors per server
• int procRemainder
  proc remainder after equal distribution
• MPI_Comm serverIntraComm
  intracomm. for each server partition
• int serverCommRank
  rank in serverIntraComm
• int serverCommSize
  size of serverIntraComm
• MPI_Comm hubServerIntraComm
  intracomm for all serverCommRank==0
  < w/i next higher level serverIntraComm
• int hubServerCommRank
  rank in hubServerIntraComm
• int hubServerCommSize
  size of hubServerIntraComm
• MPI_Comm hubServerInterComm
  intercomm. between a server & the hub
  < (on server partitions only)
• MPI_Comm * hubServerInterComms
  intercomm. array on hub processor
• int serverId
  server identifier

Friends
• class ParallelLibrary
  the ParallelLibrary class has special access privileges in order to streamline implementation

13.115.1 Detailed Description

Container class for the data associated with a single level of communicator partitioning.
A list of these levels is contained in ParallelLibrary (ParallelLibrary::parallelLevels), which defines all of the parallelism levels across one or more multilevel parallelism configurations.
The documentation for this class was generated from the following file:
• ParallelLibrary.hpp

13.116 ParallelLibrary Class Reference

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

- **ParallelLibrary ()**
  
  default constructor (used for dummy_lib)

- **ParallelLibrary (const MPIManager &mpi_mgr, ProgramOptions &prog_opts, OutputManager &output_mgr)**
  
  stand-alone and default library mode constructor; don’t require options

- ∼ParallelLibrary ()
  
  destructor

- const ParallelLevel & init_iterator_communicators (int iterator_servers, int procs_per_iterator, int min_procs_per_iterator, int max_procs_per_iterator, int max_iterator_concurrency, short default_config, short iterator_scheduling, bool peer_dynamic_avail)

  split MPI_COMM_WORLD into iterator communicators

- const ParallelLevel & init_evaluation_communicators (int evaluation_servers, int procs_per_evaluation, int min_procs_per_eval, int max_procs_per_eval, int max_evaluation_concurrency, int asynch_local_evaluation_concurrency, short default_config, short evaluation_scheduling, bool peer_dynamic_avail)

  split an iterator communicator into evaluation communicators

- const ParallelLevel & init_analysis_communicators (int analysis_servers, int procs_per_analysis, int min_procs_per_analysis, int max_procs_per_analysis, int max_analysis_concurrency, int asynch_local_analysis_concurrency, short default_config, short analysis_scheduling, bool peer_dynamic_avail)

  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 manage_outputs_restart (const ParallelLevel &pl)

  manage output streams and restart file(s) (both modes)

- void write_restart (const ParamResponsePair &prp)

  write a parameter/response set to the restart file

- ProgramOptions & program_options ()

  return programOptions reference

- OutputManager & output_manager ()

  return outputManager reference

- void terminate_modelcenter ()

  terminate ModelCenter if running

- void abort_helper (int code)

  finalize MPI with correct communicator for abort

- 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 (int &data, const ParallelLevel &pl)
  broadcast an integer across the serverIntraComm of a ParallelLevel
• void bcast_w (int &data)
  broadcast an integer across MPI_COMM_WORLD
• void bcast_i (int &data)
  broadcast an integer across an iterator communicator
• void bcast_e (int &data)
  broadcast an integer across an evaluation communicator
• void bcast_a (int &data)
  broadcast an integer across an analysis communicator
• void bcast_si (int &data)
  broadcast an integer across a strategy-iterator intra communicator
• void bcast_w (MPIPackBuffer &send_buff)
  broadcast a packed buffer across MPI_COMM_WORLD
• void bcast_i (MPIPackBuffer &send_buff)
  broadcast a packed buffer across an iterator communicator
• void bcast_e (MPIPackBuffer &send_buff)
  broadcast a packed buffer across an evaluation communicator
• void bcast_a (MPIPackBuffer &send_buff)
  broadcast a packed buffer across an analysis communicator
• void bcast_si (MPIPackBuffer &send_buff)
  broadcast a packed buffer across a strategy-iterator intra communicator
• void bcast_w (MPIUnpackBuffer &recv_buff)
  matching receive for packed buffer broadcast across MPI_COMM_WORLD
• void bcast_i (MPIUnpackBuffer &recv_buff)
  matching receive for packed buffer bcast across an iterator communicator
• void bcast_e (MPIUnpackBuffer &recv_buff)
  matching receive for packed buffer bcast across an evaluation communicator
• void bcast_a (MPIUnpackBuffer &recv_buff)
  matching receive for packed buffer bcast across an analysis communicator
• void bcast_si (MPIUnpackBuffer &recv_buff)
  matching recv for packed buffer bcast across a strat-iterator intra comm
• void barrier_w ()
  enforce MPI_Barrier on MPI_COMM_WORLD
• void barrier_i ()
  enforce MPI_Barrier on an iterator communicator
• void barrier_e ()
enforce MPI_Barrier on an evaluation communicator

- void barrier_ea ()

enforce MPI_Barrier on an analysis communicator

- void reduce_sum_ea (double *local_vals, double *sum_vals, 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, 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 (int num_recs, MPI_Request *&recv_recs)
  wait for all messages from a series of nonblocking receives

- void waitsome (int num_sends, MPI_Request *&recv_requests, int *num_recs, int *index_array, MPI_Status *&status_array)
  wait for at least one message from a series of nonblocking receives but complete all that are available

- void free (MPI_Request &request)
  free an MPI_Request

- int world_size () const
  return Dakota’s worldSize

- int world_rank () const
  return Dakota’s worldRank

- bool mpirun_flag () const
  return mpirunFlag

- bool is_null () const
  return dummyFlag

- Real parallel_time () const
  return 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
  return the number of entries in parallelConfigurations

- bool parallel_configuration_is_complete ()
  identifies if the current ParallelConfiguration has been fully populated

- void increment_parallel_configuration ()
  add a new node to parallelConfigurations and increment currPCIter

- bool w_parallel_level_defined () const
  test current parallel configuration for definition of world parallel level

- bool si_parallel_level_defined () const
test current parallel configuration for definition of strategy-iterator parallel level

- bool ie_parallel_level_defined() const
  
  test current parallel configuration for definition of iterator-evaluation parallel level

- bool ea_parallel_level_defined() const
  
  test current parallel configuration for definition of evaluation-analysis parallel level

- std::vector<MPI_Comm> analysis_intra_communicators() const
  
  return the set of analysis intra communicators for all parallel configurations (used for setting up direct simulation interfaces prior to execution time).

Private Member Functions

- void init_mpi_comm() const
  
  convenience function for initializing DAKOTA's top-level MPI communicators, based on dakotaMPIComm

- void initialize_timers() const
  
  initialize DAKOTA and UTILIB timers

- void output_timers() const
  
  conditionally output timers in destructor

- void init_communicators(const ParallelLevel &parent_pl, int num_servers, int procs_per_server, int min_procs_per_server, int max_procs_per_server, int max_concurrency, int asynch_local_concurrency, short default_config, short scheduling_override, bool peer_dynamic_avail)
  
  split a parent communicator into child server communicators

- void free_communicators(ParallelLevel &pl)
  
  deallocate intra/inter communicators for a particular ParallelLevel

- void split_communicator_dedicated_master(const ParallelLevel &parent_pl, ParallelLevel &child_pl)
  
  split a parent communicator into a dedicated master processor and num_servers child communicators

- void split_communicator_peer_partition(const ParallelLevel &parent_pl, ParallelLevel &child_pl)
  
  split a parent communicator into num_servers peer child communicators (no dedicated master processor)

- void resolve_inputs(ParallelLevel &child_pl, int avail_procs, int min_procs_per_server, int max_procs_per_server, int max_concurrency, int capacity_multiplier, short default_config, short scheduling_override, bool peer_dynamic_avail, bool print_rank)
  
  resolve user inputs into a sensible partitioning scheme

- void send(MPIPackBuffer &send_buff, int dest, int tag, ParallelLevel &parent_pl, ParallelLevel &child_pl)
  
  blocking buffer send at the current communication level

- void send(int &send_int, int dest, int tag, ParallelLevel &parent_pl, ParallelLevel &child_pl)
  
  blocking integer send at the current communication level

- void isend(MPIPackBuffer &send_buff, int dest, 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, int dest, 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, int source, int tag, MPI_Status &status, ParallelLevel &parent_pl, ParallelLevel &child_pl)
  
  blocking buffer receive at the current communication level
• void recv (int &recv_int, int source, int tag, MPI_Status &status, ParallelLevel &parent_pl, ParallelLevel &child_pl)
  blocking integer receive at the current communication level
• void irecv (MPIUnpackBuffer &recv_buff, int source, 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, int source, 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, int num_vals, const MPI_Comm &comm)
  compute a sum over comm using MPI_Reduce
• void check_error (const String &err_source, int err_code)
  check the MPI return code and abort if error
• void inherit_as_server_comm (const ParallelLevel &parent_pl, ParallelLevel &child_pl)
  convenience function for updating child serverIntraComm from parent serverIntraComm
• void inherit_as_hub_server_comm (const ParallelLevel &parent_pl, ParallelLevel &child_pl)
  convenience function for updating child hubServerIntraComm from parent serverIntraComm

Private Attributes
• const MPIManager & mpiManager
  reference to the MPI manager with Dakota’s MPI options
• ProgramOptions & programOptions
  programOptions is non-const due to updates from broadcast
• OutputManager & outputManager
  Non-const output handler to help with file redirection.
• MPI_Comm dakotaMPIComm
  MPI_Comm on which DAKOTA is running.
• int worldRank
  rank in MPI_Comm in which DAKOTA is running
• int worldSize
  size of MPI_Comm in which DAKOTA is running
• bool mpirunFlag
  flag for a parallel mpirun/yod launch
• bool dummyFlag
  prevents multiple MPI_Finalize calls due to dummy_lib
• bool outputTimings
  timing info only beyond help/version/check
• Real startCPUPTime
  start reference for UTILIB CPU timer
• Real startWCTime
  start reference for UTILIB wall clock timer
• Real startMPITime
  start reference for MPI wall clock timer
• long startClock
  start reference for local clock() timer measuring
  < parent+child CPU
• std::list< ParallelLevel > parallelLevels
  the complete set of parallelism levels for managing multilevel parallelism among one or more configurations
• std::list< ParallelConfiguration > parallelConfigurations
  the set of parallel configurations which manage list iterators for indexing into parallelLevels
• ParLevLIter currPLIter
  list iterator identifying the current node in parallelLevels
• ParConfigLIter currPCIter
  list iterator identifying the current node in parallelConfigurations

13.116.1 Detailed Description

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

The ParallelLibrary class encapsulates all of the details of performing message passing within multiple levels of parallelism. It provides functions for partitioning of levels according to user configuration input and functions for passing messages within and across MPI communicators for each of the parallelism levels. If support for other message-passing libraries beyond MPI becomes needed (PVM, ...), then ParallelLibrary would be promoted to a base class with virtual functions to encapsulate the library-specific syntax.

13.116.2 Constructor & Destructor Documentation

ParallelLibrary ( )

default 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.

ParallelLibrary ( const MPIManager & mpi_mgr, ProgramOptions & prog_opts, OutputManager & output_mgr )

stand-alone and default library mode constructor; don’t require options
  library mode constructor accepting communicator
  TODO: Update comment.
  Same constructor is used for executable and library environments and sequencing of object construction is ordered, so no need to separately get updates off command line (programOptions)
  References ParallelLibrary::init_mpi_comm(), and ParallelLibrary::initialize_timers().
### 13.116.3 Member Function Documentation

**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.

References ParallelLibrary::bcast(), ParallelLevel::dedicatedMasterFlag, OutputManager::file_tag(), ParallelLevel::hubServerCommSize, ParallelLevel::hubServerIntraComm, OutputManager::init_restart(), OutputManager::init_resultsdb(), ParallelLevel::numServers, ParallelLibrary::outputManager, ParallelLibrary::programOptions, OutputManager::redirect_cerr(), OutputManager::redirect_cout(), OutputManager::resultsOutputFile, OutputManager::resultsOutputFlag, ParallelLevel::serverCommRank, ParallelLevel::serverId, ParallelLevel::serverMasterFlag, MPIPackBuffer::size(), OutputManager::tabularDataFile, OutputManager::tabularDataFlag, and ParallelLibrary::worldRank.

Referenced by IteratorScheduler::init_iterator_parallelism(), and IteratorScheduler::init_serial_iterators().

**void terminate_modelcenter** ( )

terminate ModelCenter if running

Close streams associated with manage_outputs and manage_restart and terminate any additional services that may be active.

References Dakota::abort_handler(), Dakota::dc_ptr_int, and Dakota::mc_ptr_int.

Referenced by ParallelLibrary::~ParallelLibrary().

**void increment_parallel_configuration** ( ) [inline]

add a new node to parallelConfigurations and increment currPCIter

Called from the ParallelLibrary ctor and from Model::init_communicators(). An increment is performed for each Model initialization except the first (which inherits the world and strategy-iterator parallel levels from the first partial configuration).

References ParallelLibrary::currPCIter, ParallelConfiguration::eaPLIter, ParallelConfiguration::iePLIter, ParallelConfiguration::numParallelLevels, ParallelLibrary::parallelConfigurations, ParallelLibrary::parallelLevels, ParallelConfiguration::siPLIter, and ParallelConfiguration::wPLIter.

Referenced by Model::init_communicators(), and ParallelLibrary::init_mpi_comm().

**void init_mpi_comm** ( ) [private]

convenience function for initializing DAKOTA's top-level MPI communicators, based on dakotaMPIComm

shared function for initializing based on passed MPI_Comm

References Dakota::abort_handler(), ParallelLibrary::currPLIter, ParallelLibrary::dakotaMPIComm, ParallelLibrary::increment_parallel_configuration(), ParallelLibrary::mpirunFlag, ParallelLibrary::outputManager, ParallelLibrary::parallelLevels, ParallelLevel::serverCommRank, ParallelLevel::serverCommSize, ParallelLevel::serverIntraComm, ParallelLibrary::startMPITime, OutputManager::startup_message(), ParallelLibrary::worldRank, and ParallelLibrary::worldSize.

Referenced by ParallelLibrary::ParallelLibrary().
void init_communicators ( const ParallelLevel & parent_pl, int num_servers, int procs_per_server, int min_procs_per_server, int max_procs_per_server, int max_concurrency, int asynch_local_concurrency, short default_config, short scheduling_override, bool peer_dynamic_avail ) [private]

split a parent communicator into child server communicators

Split parent communicator into concurrent child server partitions as specified by the passed parameters. This constructs new child intra-communicators and parent-child inter-communicators. This function is called from the Strategy constructor for the concurrent iterator level and from ApplicationInterface::init_communicators() for the concurrent evaluation and concurrent analysis levels.

References ParallelLibrary::currPCIter, ParallelLibrary::currPLIter, ParallelLevel::dedicatedMasterFlag, ParallelLevel::messagePass, ParallelLevel::numServers, ParallelLibrary::parallelLevels, ParallelLevel::procsPerServer, ParallelLibrary::resolve_inputs(), ParallelLevel::serverCommRank, ParallelLevel::serverCommSize, ParallelLibrary::split_communicator_dedicated_master(), and ParallelLibrary::split_communicator_peer_partition().

Referenced by ParallelLibrary::init_analysis_communicators(), ParallelLibrary::init_evaluation_communicators(), and ParallelLibrary::init_iterator_communicators().

void resolve_inputs ( ParallelLevel & child_pl, int avail_procs, int min_procs_per_server, int max_procs_per_server, int max_concurrency, int capacity_multiplier, short default_config, short scheduling_override, bool peer_dynamic_avail, bool print_rank ) [private]

resolve user inputs into a sensible partitioning scheme

This function is responsible for the "auto-configure" intelligence of DAKOTA. It resolves a variety of inputs and overrides into a sensible partitioning configuration for a particular parallelism level. It also handles the general case in which a user’s specification request does not divide out evenly with the number of available processors for the level. If num_servers & procs_per_server are both nondefault, then the former takes precedence.

References Dakota::abort_handler(), ParallelLevel::dedicatedMasterFlag, ParallelLevel::numServers, ParallelLevel::procRemainder, and ParallelLevel::procsPerServer.

Referenced by ParallelLibrary::init_communicators().

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

- ParallelLibrary.hpp
- ParallelLibrary.cpp

13.117 ParamResponsePair Class Reference

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

Public Member Functions

- ParamResponsePair ()
  
  default constructor

- ParamResponsePair (const Variables &vars, const String &interface_id, const Response &response, bool deep_copy=false)
  
  alternate constructor for temporaries

- ParamResponsePair (const Variables &vars, const String &interface_id, const Response &response, const int eval_id, bool deep_copy=true)
  
  standard constructor for history uses

- ParamResponsePair (const ParamResponsePair &pair)
  
  copy constructor
• ~ParamResponsePair ()
  destructor
• ParamResponsePair & operator= (const ParamResponsePair &pair)
  assignment operator
• void read (std::istream &s)
  read a ParamResponsePair object from an std::istream
• void write (std::ostream &s) const
  write a ParamResponsePair object to an std::ostream
• void read.annotated (std::istream &s)
  read a ParamResponsePair object in annotated format from an std::istream
• void write.annotated (std::ostream &s) const
  write a ParamResponsePair object in annotated format to an std::ostream
• void write.tabular (std::ostream &s) const
  write a ParamResponsePair object in tabular format to an std::ostream
• void read (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
• void eval.id (int id)
  set 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 Member Functions

• template<class Archive>
  void serialize (Archive &ar, const unsigned int version)
  serialize the PRP: write and read are symmetric for this class
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

- class boost::serialization::access
  *allow boost access to serialize this class*
- **bool operator==(const ParamResponsePair &pair1, const ParamResponsePair &pair2)**
  *equality operator*
- **bool operator!=(const ParamResponsePair &pair1, const ParamResponsePair &pair2)**
  *inequality operator*

13.117.1 Detailed Description

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

*ParamResponsePair* provides a container class for association of the input for a particular function evaluation (a variables object) with the output from this function evaluation (a response object), along with an evaluation identifier. This container defines the basic unit used in the data_pairs cache, in restart file operations, and in a variety of scheduling algorithm queues. With the advent of STL, replacement of arrays of this class with map<> and pair<> template constructs may be possible (using map<pair<int,String>>, pair<Variables,Response>>, for example), assuming that deep copies, I/O, alternate constructors, etc., can be adequately addressed. Boost tuple<> may also be a candidate.

13.117.2 Constructor & Destructor Documentation

**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).

**ParamResponsePair ( const Variables & vars, const String & interface_id, const Response & response, const int eval_id, bool deep_copy = true ) [inline]**

standard constructor for history uses

Uses of this constructor often do not share representations since deep copies are used when history mechanisms (e.g., data_pairs and beforeSynchCorePRPQueue) are involved.
13.117.3 Member Function Documentation

```cpp
void read ( MPIUnpackBuffer & s ) [inline]
```

read a `ParamResponsePair` object from a packed MPI buffer

- interfaceId is omitted since master processor retains interface ids and communicates asv and response data only with slaves.
  - References `ParamResponsePair::evalInterfaceIds`, `ParamResponsePair::prPairParameters`, and `ParamResponsePair::prPairResponse`.

```cpp
void write ( MPIPackBuffer & s ) const [inline]
```

write a `ParamResponsePair` object to a packed MPI buffer

- interfaceId is omitted since master processor retains interface ids and communicates asv and response data only with slaves.
  - References `ParamResponsePair::evalInterfaceIds`, `ParamResponsePair::prPairParameters`, and `ParamResponsePair::prPairResponse`.

13.117.4 Member Data Documentation

```cpp
IntStringPair evalInterfaceIds [private]
```

the `evalInterfaceIds` aggregate

- the function evaluation identifier (assigned from `Interface::evalIdCntr`) is paired with the interface used to generate the response object. Used in PRPCache id_vars_set_compare to prevent duplicate detection on results from different interfaces. `evalInterfaceIds` belongs here rather than in `Response` since some `Response` objects involve consolidation of several fn evals (e.g., `Model::synchronize_derivatives()`) that are not, in total, generated by a single interface. The `prPair`, on the other hand, is used for storage of all low level fn evals that get evaluated in `ApplicationInterface::map()`.

  - Referenced by `ParamResponsePair::eval_id()`, `ParamResponsePair::eval_interface_ids()`, `ParamResponsePair::interface_id()`, `ParamResponsePair::operator=()`, `Dakota::operator==()`, `ParamResponsePair::read()`, `ParamResponsePair::write()`, and `ParamResponsePair::write_tabular()`.

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

- `ParamResponsePair.hpp`

13.118 ParamStudy Class Reference

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

Inheritance diagram for `ParamStudy`:

```
   Iterator
      |
      Analyzer
      |
      PStudyDACE
      |
      ParamStudy
```
Public Member Functions

- **ParamStudy** (ProblemDescDB &problem_db, Model &model)
  
  *constructor*

- **~ParamStudy** ()
  
  *destructor*

- **void pre_run** ()
  
  pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori

- **void extract_trends** ()
  
  Mapping of the *core_run*() virtual function for the PStudy/DACE branch.

- **void post_input** ()
  
  read tabular data for post-run mode

- **void post_run** (std::ostream &s)
  
  post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way

Private Member Functions

- **void sample** ()
  
  performs the parameter study by sampling from a list of points

- **void vector_loop** ()
  
  performs the parameter study by sampling along a vector, starting from an initial point followed by numSteps increments along continuous/discrete step vectors

- **void centered_loop** ()
  
  performs a number of plus and minus offsets for each parameter centered about an initial point

- **void multidim_loop** ()
  
  performs a full factorial combination for all intersections defined by a set of multidimensional partitions

- **bool load_distribute_points** (const String &points_filename, bool annotated)

- **template<typename OrdinalType, typename ScalarTypeA, typename ScalarTypeC, typename ScalarTypeDI, typename ScalarTypeDR>
  
  bool distribute** (const Teuchos::SerialDenseVector<OrdinalType, ScalarTypeA> &all_data, Teuchos::SerialDenseVector<OrdinalType, ScalarTypeC> &c_data, Teuchos::SerialDenseVector<OrdinalType, ScalarTypeDI> &di_data, Teuchos::SerialDenseVector<OrdinalType, ScalarTypeDR> &dr_data)
  
  distributes incoming all vector in standard variable ordering among continuous, discrete int, and discrete real vectors

- **template<typename ScalarType>
  
  bool distribute** (const std::vector<ScalarType> &all_data, std::vector<ScalarType> &c_data, std::vector<ScalarType> &di_data, std::vector<ScalarType> &dr_data)
  
  distributes incoming all array in standard variable ordering among continuous, discrete int, and discrete real arrays

- **bool distribute_list_of_points** (const RealVector &list_of_pts)
  
  distributes incoming list_of_pts among listCVPoints, listDIVPoints, and listDRVPoints

- **void final_point_to_step_vector** ()
  
  compute step vectors from finalPoint, initial points, and numSteps

- **void distribute_partitions** ()
  
  compute step vectors from {cont, discInt, discReal} VarPartitions and global bounds
• bool check_num_steps (int num_steps)
  perform error checks on numSteps
• bool check_final_point (const RealVector &final_pt)
  perform error checks on finalPoint
• bool check_steps_per_variable (const IntVector &steps_per_var)
  perform error checks on stepsPerVariable
• bool check_variable_partitions (const UShortArray &partitions)
  perform error checks on variable partitions
• bool check_finite_bounds ()
  check for finite variable bounds withiniteratedModel, as required for computing partitions of finite ranges
• bool check_ranges_sets (int num_steps)
  sanity check for vector parameter study
• bool check_ranges_sets (const IntVector &c_steps, const IntVector &di_steps, const IntVector &dr_steps)
  sanity check for centered parameter study
• bool check_sets (const IntVector &c_steps, const IntVector &di_steps, const IntVector &dr_steps)
  sanity check for increments along int/real set dimensions
• int integer_step (int range, int num_steps) const
  check for integer remainder and return step
• int index_step (size_t start, size_t end, int num_steps) const
  check for out of bounds and index remainder and return step
• void c_step (size_t c_index, int increment, Variables &vars)
  helper function for performing a continuous step in one variable
• void dri_step (size_t di_index, int increment, Variables &vars)
  helper function for performing a discrete step in an integer range variable
• void dsi_step (size_t di_index, int increment, const IntSet &values, Variables &vars)
  helper function for performing a discrete step in an integer set variable
• void dsr_step (size_t dr_index, int increment, const RealSet &values, Variables &vars)
  helper function for performing a discrete step in a real set variable
• void reset (Variables &vars)
  reset vars to initial point (center)
• void centered_header (const String &type, size_t var_index, int step, size_t hdr_index)
  store a centered parameter study header within allHeaders

Private Attributes
• size_t numEvals
  total number of parameter study evaluations computed from specification
• RealVectorArray listCVPoints
  array of continuous evaluation points for the list parameter study
• IntVectorArray listDIVPoints
  array of discrete int evaluation points for the list parameter study
• RealVectorArray listDRVPoints
  array of discrete real evaluation points for the list parameter study
• RealVector initialCVPoint
  
  the continuous starting point for vector and centered parameter studies

• IntVector initialDIVPoint

  the continuous starting point for vector and centered parameter studies

• RealVector initialDRVPoint

  the continuous starting point for vector and centered parameter studies

• RealVector contStepVector

  the n-dimensional continuous increment

• IntVector discIntStepVector

  the n-dimensional discrete value or index increment

• IntVector discRealStepVector

  the n-dimensional discrete real index increment

• RealVector finalPoint

  the ending point for vector parameter study (a specification option)

• int numSteps

  the number of times continuous/discrete step vectors are applied for vector parameter study (a specification option)

• IntVector contStepsPerVariable

  number of offsets in the plus and the minus direction for each continuous variable in a centered parameter study

• IntVector discIntStepsPerVariable

  number of offsets in the plus and the minus direction for each discrete integer variable in a centered parameter study

• IntVector discRealStepsPerVariable

  number of offsets in the plus and the minus direction for each discrete real variable in a centered parameter study

• UShortArray contVarPartitions

  number of partitions for each continuous variable in a multidim parameter study

• UShortArray discIntVarPartitions

  number of partitions for each discrete integer variable in a multidim parameter study

• UShortArray discRealVarPartitions

  number of partitions for each discrete real variable in a multidim parameter study

### Additional Inherited Members

#### 13.118.1 Detailed Description

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

The ParamStudy class contains several algorithms for performing parameter studies of different types. The vector parameter study steps along an n-dimensional vector from an arbitrary initial point to an arbitrary final point in a specified number of steps. The centered parameter study performs a number of plus and minus offsets in each coordinate direction around a center point. A multidimensional parameter study fills an n-dimensional hypercube based on bounds and a specified number of partitions for each dimension. And the list parameter study provides for a user specification of a list of points to evaluate, which allows general parameter investigations not fitting the structure of vector, centered, or multidim parameter studies.
13.118.2 Member Function Documentation

```cpp
void pre_run() [virtual]
```

pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori

pre-run phase, which a derived iterator may optionally reimplement; when not present, pre-run is likely integrated into the derived run function. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s `pre_run()`, if implemented, typically before performing its own implementation steps.

Reimplemented from Iterator.

References Dakota::abort_handler(), SharedVariablesData::active_components_totals(), Analyzer::allHeaders, Analyzer::allVariables, ParamStudy::centered_loop(), Variables::continuous_variables(), ParamStudy::contSteps-PerVariable, ParamStudy::contStepVector, ParamStudy::contVarPartitions, Dakota::copy_data(), Model::current_variables(), ParamStudy::discIntStepsPerVariable, ParamStudy::discIntStepVector, ParamStudy::discIntVarPartitions, ParamStudy::discRealStepsPerVariable, ParamStudy::discRealStepVector, ParamStudy::discRealVarPartitions, Variables::discrete_int_variables(), Variables::discrete_real_variables(), ParamStudy::distribute_partitions(), ParamStudy::final_point_to_step_vector(), ParamStudy::finalPoint, ParamStudy::initialCVPoint, ParamStudy::initialDIVPoint, ParamStudy::initialDRVPoint, Iterator::iteratedModel, Iterator::methodName, ParamStudy::multidim_loop(), ParamStudy::numEvals, ParamStudy::numSteps, Iterator::outputLevel, Iterator::pre_run(), ParamStudy::sample(), Variables::shared_data(), ParamStudy::vector_loop(), and Dakota::write_ordered().

```cpp
void post_run(std::ostream &s) [virtual]
```

post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way

Post-run phase, which a derived iterator may optionally reimplement; when not present, post-run is likely integrated into run. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s `post_run()`, typically after performing its own implementation steps.

Reimplemented from Analyzer.

References Analyzer::allResponses, Analyzer::allVariables, SensAnalysisGlobal::compute_correlations(), Iterator::methodName, Analyzer::post_run(), PStudyDACE::pStudyDACESensGlobal, and Iterator::subIteratorFlag.

```cpp
bool load_distribute_points(const String &points_filename, bool annotated) [private]
```

Load from file and distribute points; using this function to manage construction of the temporary array

References ParamStudy::distribute_list_of_points(), Analyzer::numContinuousVars, Analyzer::numDiscrete-IntVars, and Analyzer::numDiscreteRealVars.

Referenced by ParamStudy::ParamStudy().

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

- ParamStudy.hpp
- ParamStudy.cpp

13.119 partial_prp_equality Struct Reference

predicate for comparing ONLY the interfaceId and Vars attributes of PRPair

Public Member Functions

- bool operator() (const ParamResponsePair &database_pr, const ParamResponsePair &search_pr) const
  
  access operator
Detailed Description

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

- PRPMultiIndex.hpp

wrapper to delegate to the ParamResponsePair hash_value function

Public Member Functions

- std::size_t operator() (const ParamResponsePair &prp) const

Detailed Description

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

- PRPMultiIndex.hpp

PecosApproximation Class Reference

Derived approximation class for global basis polynomials.

Inheritance diagram for PecosApproximation:

<table>
<thead>
<tr>
<th>Approximation</th>
</tr>
</thead>
<tbody>
<tr>
<td>PecosApproximation</td>
</tr>
</tbody>
</table>

Public Member Functions

- PecosApproximation ()
  default constructor
- PecosApproximation (ProblemDescDB &problem_db, const SharedApproxData &shared_data)
  standard ProblemDescDB-driven constructor
- PecosApproximation (const SharedApproxData &shared_data)
  alternate constructor
- ~PecosApproximation ()
  destructor
- void expansion_coefficient_flag (bool coeff_flag)
  set pcosBasisApprox.configOptions.expansionCoeffFlag
- bool expansion_coefficient_flag () const
CHAPTER 13. CLASS DOCUMENTATION

get pecosBasisApprox.configOptions.expansionCoeffFlag

void expansion_gradient_flag (bool grad_flag)

set pecosBasisApprox.configOptions.expansionGradFlag

bool expansion_gradient_flag () const

get pecosBasisApprox.configOptions.expansionGradFlag

RealVector dense_coefficients () const

return expansion coefficients in a form consistent with the shared multi-index

void compute_component_effects ()

Performs global sensitivity analysis using Sobol' Indices by computing component (main and interaction) effects.

void compute_total_effects ()

Performs global sensitivity analysis using Sobol' Indices by computing total effects.

const Pecos::RealVector & sobol_indices () const

return polyApproxRep->sobolIndices

const Pecos::RealVector & total_sobol_indices () const

return polyApproxRep->totalSobolIndices

Pecos::ULongULongMap sparse_sobol_index_map () const

return RegressOrthogPolyApproximation::sparseSobolIndexMap

const Pecos::RealVector & dimension_decay_rates () const

return OrthogPolyApproximation::decayRates

void allocate_arrays ()

invoke Pecos::PolynomialApproximation::allocate_arrays()

Real mean ()

return the mean of the expansion, treating all variables as random

Real mean (const Pecos::RealVector &x)

return the mean of the expansion for a given parameter vector, treating a subset of the variables as random

const Pecos::RealVector & mean_gradient ()

return the gradient of the expansion mean for a given parameter vector, treating all variables as random

const Pecos::RealVector & mean_gradient (const Pecos::RealVector &x, const Pecos::SizetArray &dvv)

return the gradient of the expansion mean for a given parameter vector and given DVV, treating a subset of the variables as random

Real variance ()

return the variance of the expansion, treating all variables as random

Real variance (const Pecos::RealVector &x)

return the variance of the expansion for a given parameter vector, treating a subset of the variables as random

const Pecos::RealVector & variance_gradient ()

return the gradient of the expansion variance for a given parameter vector, treating all variables as random

const Pecos::RealVector & variance_gradient (const Pecos::RealVector &x, const Pecos::SizetArray &dvv)

return the gradient of the expansion variance for a given parameter vector and given DVV, treating a subset of the variables as random

Real covariance (PecosApproximation *pecos_approx_2)

return the covariance between two response expansions, treating all variables as random

Real covariance (const Pecos::RealVector &x, PecosApproximation *pecos_approx_2)

return the covariance between two response expansions, treating a subset of the variables as random
• Real delta covariance (PecosApproximation *pecos_approx_2)
  return the change in covariance between two response expansions, treating all variables as random
• Real delta covariance (const Pecos::RealVector &x, PecosApproximation *pecos_approx_2)
  return the change in covariance between two response expansions, treating a subset of the variables as random
• Real delta_mean ()
  return the change in mean between two response expansions, treating all variables as random
• Real delta_mean (const RealVector &x)
  return the change in mean between two response expansions, treating a subset of variables as random
• Real delta_std_deviation ()
  return the change in standard deviation between two response expansions, treating all variables as random
• Real delta_std_deviation (const RealVector &x)
  return the change in standard deviation between two response expansions, treating a subset of variables as random
• Real delta_beta (bool cdf_flag, Real z_bar)
  return the change in reliability index (mapped from z_bar) between two response expansions, treating all variables as random
• Real delta_beta (const RealVector &x, bool cdf_flag, Real z_bar)
  return the change in reliability index (mapped from z_bar) between two response expansions, treating a subset of variables as random
• Real delta_z (bool cdf_flag, Real beta_bar)
  return the change in response level (mapped from beta_bar) between two response expansions, treating all variables as random
• Real delta_z (const RealVector &x, bool cdf_flag, Real beta_bar)
  return the change in response level (mapped from beta_bar) between two response expansions, treating a subset of the variables as random
• void compute_moments ()
  compute moments up to the order supported by the Pecos polynomial approximation
• void compute_moments (const Pecos::RealVector &x)
  compute moments in all-variables mode up to the order supported by the Pecos polynomial approximation
• const RealVector & moments () const
  return virtual Pecos::PolynomialApproximation::moments()
• const RealVector & expansion_moments () const
  return Pecos::PolynomialApproximation::expansionMoments
• const RealVector & numerical_moments () const
  return Pecos::PolynomialApproximation::numericalMoments
• void standardize_moments (const Pecos::RealVector &central_moments, Pecos::RealVector &std_moments)
  standardize the central moments returned from Pecos
• Pecos::BasisApproximation & pecos_basis_approximation ()
  return pecosBasisApprox
Protected Member Functions

- Real value (const Variables &vars)
  retrieve the approximate function value for a given parameter vector
- const Pecos::RealVector & gradient (const Variables &vars)
  retrieve the approximate function gradient for a given parameter vector
- const Pecos::RealSymMatrix & hessian (const Variables &vars)
  retrieve the approximate function Hessian for a given parameter vector
- int min_coefficients () const
  return the minimum number of samples (unknowns) required to build the derived class approximation type in num-Vars dimensions
- void build ()
  builds the approximation from scratch
- void rebuild ()
  rebuilds the approximation incrementally
- void pop (bool save_data)
  removes entries from end of SurrogateData::{vars,resp}Data (last points appended, or as specified in args)
- void restore ()
  restores state prior to previous append()
- void finalize ()
  finalize approximation by applying all remaining trial sets
- void store ()
  store current approximation for later combination
- void combine (short corr_type)
  combine current approximation with previously stored approximation
- void print_coefficients (std::ostream &s, bool normalized=false)
  print the coefficient array computed in build()/rebuild()
- const RealVector & approximation_coefficients () const
  return the coefficient array computed by build()/rebuild()
- void approximation_coefficients (const RealVector &approx_coeffs)
  set the coefficient array from external sources, rather than computing with build()/rebuild()
- void coefficient_labels (std::vector<std::string> &coeff_labels) const
  print the coefficient array computed in build()/rebuild()

Private Member Functions

- void approx_type_to_basis_type (const String &approx_type, short &basis_type)
  utility to convert Dakota type string to Pecos type enumeration

Private Attributes

- Pecos::BasisApproximation pecosBasisApprox
  the Pecos basis approximation, encompassing OrthogPolyApproximation and InterpPolyApproximation
- Pecos::PolynomialApproximation * polyApproxRep
  convenience pointer to representation of Pecos polynomial approximation
Additional Inherited Members

13.121.1 Detailed Description

Derived approximation class for global basis polynomials.

The PecosApproximation class provides a global approximation based on basis polynomials. This includes orthogonal polynomials used for polynomial chaos expansions and interpolation polynomials used for stochastic collocation.

13.121.2 Member Function Documentation

void build() [inline], [protected], [virtual]

builds the approximation from scratch

This is the common base class portion of the virtual fn and is insufficient on its own; derived implementations should explicitly invoke (or reimplement) this base class contribution.

Reimplemented from Approximation.
References Approximation::build(), and PecosApproximation::pecosBasisApprox.

void rebuild() [inline], [protected], [virtual]

rebuilds the approximation incrementally

This is the common base class portion of the virtual fn and is insufficient on its own; derived implementations should explicitly invoke (or reimplement) this base class contribution.

Reimplemented from Approximation.
References PecosApproximation::pecosBasisApprox.

void pop(bool save_data) [inline], [protected], [virtual]

removes entries from end of SurrogateData::{vars,resp}Data (last points appended, or as specified in args)

This is the common base class portion of the virtual fn and is insufficient on its own; derived implementations should explicitly invoke (or reimplement) this base class contribution.

Reimplemented from Approximation.
References PecosApproximation::pecosBasisApprox, and Approximation::pop().

void restore() [inline], [protected], [virtual]

restores state prior to previous append()

This is the common base class portion of the virtual fn and is insufficient on its own; derived implementations should explicitly invoke (or reimplement) this base class contribution.

Reimplemented from Approximation.
References PecosApproximation::pecosBasisApprox, and Approximation::restore().

void finalize() [inline], [protected], [virtual]

finalize approximation by applying all remaining trial sets

This is the common base class portion of the virtual fn and is insufficient on its own; derived implementations should explicitly invoke (or reimplement) this base class contribution.

Reimplemented from Approximation.
References Approximation::finalize(), and PecosApproximation::pecosBasisApprox.

The documentation for this class was generated from the following files:
13.122 ProblemDescDB Class Reference

The database containing information parsed from the DAKOTA input file.

Inheritance diagram for ProblemDescDB:

```
ProblemDescDB
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>NIDRProblemDescDB</td>
</tr>
</tbody>
</table>
```

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 parse_inputs (const ProgramOptions &prog_opts, DbCallbackFunctionPtr callback=NULL, void *callback_data=NULL)**
  Parses the input file or input string if present and executes callbacks. Does not perform any validation.
- **void check_and_broadcast (const ProgramOptions &prog_opts)**
  performs check_input, broadcast, and post_process, but for now, allowing separate invocation through the public API as well
- **void check_input ()**
  verifies that there is at least one of each of the required keywords in the dakota input file
- **void broadcast ()**
  invokes send_dbl_buffer() and receive_dbl_buffer() to broadcast DB data across the processor allocation. Used by manage_inputs().
- **void post_process ()**
  post-processes the (minimal) input specification to assign default variables/responses specification arrays. Used by manage_inputs().
- **void lock ()**
  Locks the database in order to prevent data access when the list nodes may not be set properly. Unlocked by a set nodes operation.
- **void unlock ()**
  Explicitly unlocks the database. Use with care.
• void set_db_list_nodes (const String &method_tag)
  set dataMethodIter based on a method identifier string to activate a particular method specification in dataMethodList and use pointers from this method specification to set all other list iterators.
• void set_db_list_nodes (const size_t &method_index)
  set dataMethodIter based on an index within dataMethodList to activate a particular method specification and use pointers from this method specification to set all other list iterators.
• void resolve_top_method ()
  For a (default) environment lacking a top method pointer, this function is used to determine which of several potential method specifications corresponds to the top method and then sets the list nodes accordingly.
• void set_db_method_node (const String &method_tag)
  set dataMethodIter based on a method identifier string to activate a particular method specification (only).
• void set_db_method_node (const size_t &method_index)
  set dataMethodIter based on an index within dataMethodList to activate a particular method specification (only).
• size_t get_db_method_node ()
  return the index of the active node in dataMethodList
• void set_db_model_nodes (const String &model_tag)
  set the model list iterators (dataModelIter, dataVariablesIter, dataInterfaceIter, and dataResponsesIter) based on the model identifier string
• void set_db_model_nodes (const size_t &model_index)
  set the model list iterators (dataModelIter, dataVariablesIter, dataInterfaceIter, and dataResponsesIter) based on an index within dataModelList
• size_t get_db_model_node ()
  return the index of the active node in dataModelList
• void set_db_variables_node (const String &variables_tag)
  set dataVariablesIter based on the variables identifier string
• void set_db_interface_node (const String &interface_tag)
  set dataInterfaceIter based on the interface identifier string
• void set_db_responses_node (const String &responses_tag)
  set dataResponsesIter based on the responses identifier string
• ParallelLibrary & parallel_library () const
  return the parallelLib reference
• IteratorList & iterator_list ()
  return a list of all Iterator objects that have been instantiated
• ModelList & model_list ()
  return a list of all Model objects that have been instantiated
• VariablesList & variables_list ()
  return a list of all Variables objects that have been instantiated
• InterfaceList & interface_list ()
  return a list of all Interface objects that have been instantiated
• ResponseList & response_list ()
  return a list of all Response objects that have been instantiated
• const RealVector & get_rv (const String &entry_name) const
  get a RealVector out of the database based on an identifier string
• const IntVector & get_jv (const String &entry_name) const
get an IntVector out of the database based on an identifier string

- `const SizetArray & get_sza (const String &entry_name) const`

get an SizetArray out of the database based on an identifier string

- `const UShortArray & get_usa (const String &entry_name) const`

get an UShortArray out of the database based on an identifier string

- `const RealSymMatrix & get_rsm (const String &entry_name) const`

get a RealSymMatrix out of the database based on an identifier string

- `const RealVectorArray & get_rva (const String &entry_name) const`

get a RealVectorArray out of the database based on an identifier string

- `const IntVectorArray & get_iva (const String &entry_name) const`

get an IntVectorArray out of the database based on an identifier string

- `const IntSet & get_is (const String &entry_name) const`

get an IntSet out of the database based on an identifier string

- `const IntSetArray & get_isa (const String &entry_name) const`

get an IntSetArray out of the database based on an identifier string

- `const RealSetArray & get_rsa (const String &entry_name) const`

get a RealSetArray out of the database based on an identifier string

- `const IntRealMapArray & get_irma (const String &entry_name) const`

get an IntRealMapArray out of the database based on an identifier string

- `const RealRealMapArray & get_rrma (const String &entry_name) const`

get a RealRealMapArray out of the database based on an identifier string

- `const StringArray & get_sa (const String &entry_name) const`

get a StringArray out of the database based on an identifier string

- `const String2DArray & get_s2a (const String &entry_name) const`

get a String2DArray out of the database based on an identifier string

- `const String & get_string (const String &entry_name) const`

get a String out of the database based on an identifier string

- `const Real & get_real (const String &entry_name) const`

get a Real out of the database based on an identifier string

- `int get_int (const String &entry_name) const`

get an int out of the database based on an identifier string

- `short get_short (const String &entry_name) const`

get a short out of the database based on an identifier string

- `unsigned short get_ushort (const String &entry_name) const`

get an unsigned short out of the database based on an identifier string

- `size_t get_sizet (const String &entry_name) const`

get a size_t out of the database based on an identifier string

- `bool get_bool (const String &entry_name) const`

get a bool out of the database based on an identifier string

- `void ** get_voidss (const String &entry_name) const`

for getting a void**, e.g., `&dlLib`

- `void insert_node (const DataEnvironment &data_env)`

set the DataEnvironment 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 &rv) 
  set a RealVector within the database based on an identifier string
• void set (const String &entry_name, const IntVector &iv) 
  set an IntVector within the database based on an identifier string
• void set (const String &entry_name, const RealSymMatrix &rsm) 
  set a RealSymMatrix within the database based on an identifier string
• void set (const String &entry_name, const RealVectorArray &rva) 
  set a RealVectorArray within the database based on an identifier string
• void set (const String &entry_name, const IntVectorArray &iva) 
  set an IntVectorArray within the database based on an identifier string
• void set (const String &entry_name, const IntSetArray &isa) 
  set an IntSetArray within the database based on an identifier string
• void set (const String &entry_name, const RealSetArray &rsa) 
  set a RealSetArray within the database based on an identifier string
• void set (const String &entry_name, const IntRealMapArray &irma) 
  set an IntRealMapArray within the database based on an identifier string
• void set (const String &entry_name, const RealRealMapArray &rrma) 
  set a RealRealMapArray within the database based on an identifier string
• void set (const String &entry_name, const StringArray &sa) 
  set a StringArray within the database based on an identifier string
• bool is_null () const
  function to check dbRep (does this envelope contain a letter)

Protected Member Functions

• ProblemDescDB (BaseConstructor, ParallelLibrary &parallel_lib) 
  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)
• virtual void derived_parse_inputs (const ProgramOptions &prog_opts)
  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

- **DataEnvironment** environmentSpec
  
  the environment specification (only one allowed) resulting from a call to environment\_kwhandler() or insert\_node()

- std::list< DataMethod > dataMethodList
  
  list of method specifications, one for each call to method\_kwhandler() or insert\_node()

- std::list< DataModel > dataModelList
  
  list of model specifications, one for each call to model\_kwhandler() or insert\_node()

- std::list< DataVariables > dataVariablesList
  
  list of variables specifications, one for each call to variables\_kwhandler() or insert\_node()

- std::list< DataInterface > dataInterfaceList
  
  list of interface specifications, one for each call to interface\_kwhandler() or insert\_node()

- std::list< DataResponses > dataResponsesList
  
  list of responses specifications, one for each call to responses\_kwhandler() or insert\_node()

- size\_t environmentCntr
  
  counter for environment specifications used in check\_input

Private Member Functions

- const Iterator & get\_iterator ()
  
  retrieve an existing Iterator, if it exists, or instantiate a new one

- const Iterator & get\_iterator (Model &model)
  
  retrieve an existing Iterator, if it exists, or instantiate a new one

- const Model & get\_model ()
  
  retrieve an existing Model, if it exists, or instantiate a new one

- const Variables & get\_variables ()
  
  retrieve an existing Variables, if it exists, or instantiate a new one

- const Interface & get\_interface ()
  
  retrieve an existing Interface, if it exists, or instantiate a new one

- const Response & get\_response (const Variables &vars)
  
  retrieve an existing Response, if it exists, or instantiate a new one

- ProblemDescDB * get\_db (ParallelLibrary &parallel\_lib)
  
  Used by the envelope constructor to instantiate the correct letter class.

- void send\_db\_buffer ()
  
  MPI send of a large buffer containing environmentSpec and all objects in dataMethodList, dataModelList, dataVariablesList, dataInterfaceList, and dataResponsesList. Used by manage\_inputs().

- void receive\_db\_buffer ()
  
  MPI receive of a large buffer containing environmentSpec and all objects in dataMethodList, dataModelList, dataVariablesList, dataInterfaceList, and dataResponsesList. Used by manage\_inputs().

- void echo\_input\_file (const ProgramOptions &prog\_opts)
  
  echo the (potentially) specified input file or string to stdout
**Private Attributes**

- **ParallelLibrary & parallelLib**
  reference to the parallel_lib object passed from main
- **std::list< DataMethod >::iterator dataMethodIter**
  iterator identifying the active list node in dataMethodList
- **std::list< DataModel >::iterator dataModelIter**
  iterator identifying the active list node in dataModelList
- **std::list< DataVariables >::iterator dataVariablesIter**
  iterator identifying the active list node in dataVariablesList
- **std::list< DataInterface >::iterator dataInterfaceIter**
  iterator identifying the active list node in dataInterfaceList
- **std::list< DataResponses >::iterator dataResponsesIter**
  iterator identifying the active list node in dataResponsesList
- **IteratorList iteratorList**
  list of iterator objects, one for each method specification
- **ModelList modelList**
  list of model objects, one for each model specification
- **VariablesList variablesList**
  list of variables objects, one for each variables specification
- **InterfaceList interfaceList**
  list of interface objects, one for each interface specification
- **ResponseList responseList**
  list of response objects, one for each responses specification
- **bool methodDBLocked**
  prevents use of get_<typename> retrieval and set_<typename> update functions prior to setting the list node for the active method specification
- **bool modelDBLocked**
  prevents use of get_<typename> retrieval and set_<typename> update functions prior to setting the list node for the active model specification
- **bool variablesDBLocked**
  prevents use of get_<typename> retrieval and set_<typename> update functions prior to setting the list node for the active variables specification
- **bool interfaceDBLocked**
  prevents use of get_<typename> retrieval and set_<typename> update functions prior to setting the list node for the active interface specification
- **bool responsesDBLocked**
  prevents use of get_<typename> retrieval and set_<typename> update functions prior to setting the list node for the active responses specification
- **ProblemDescDB * dbRep**
  pointer to the letter (initialized only for the envelope)
- **int referenceCount**
  number of objects sharing dbRep
Friends

- class Model
  
  Model requires access to get_variables() and get_response()

- class SingleModel
  
  SingleModel requires access to get_interface()

- class HierarchSurrModel
  
  HierarchSurrModel requires access to get_model()

- class DataFitSurrModel
  
  DataFitSurrModel requires access to get_iterator() and get_model()

- class NestedModel
  
  NestedModel requires access to get_interface(), get_response(), get_iterator(), and get_model()

- class Environment
  
  Environment requires access to get_iterator()

- class IteratorScheduler
  
  Environment requires access to get_iterator()

- class Iterator
  
  Iterator requires access to get_model()

- class MetaIterator
  
  Iterator requires access to get_model()

- class SeqHybridMetaIterator
  
  SeqHybridMetaIterator requires access to get_model()

- class CollabHybridMetaIterator
  
  CollabHybridMetaIterator requires access to get_model()

- class ConcurrentMetaIterator
  
  ConcurrentMetaIterator requires access to get_model()

- class SurrBasedLocalMinimizer
  
  SurrBasedLocalMinimizer requires access to get_iterator()

- class SurrBasedGlobalMinimizer
  
  SurrBasedGlobalMinimizer requires access to get_iterator()

13.122.1 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 (DataEnvironment, DataMethod, DataVariables, DataInterface, or DataResponses) and, for all cases except environment, appends the object to a linked list (dataMethodList, dataVariablesList, dataInterfaceList, or dataResponsesList). No environment linked list is used since only one environment specification is allowed.

13.122.2 Constructor & Destructor Documentation

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.
ProblemDescDB ( ParallelLibrary & parallel_lib )

standard constructor

This is the envelope constructor which uses problem_db to build a fully populated db object. It only needs to extract enough data to properly execute get_db(problem_db), since the constructor overloaded with BaseConstructor builds the actual base class data inherited by the derived classes.

References Dakota::abort_handler(), ProblemDescDB::dbRep, and ProblemDescDB::get_db().

ProblemDescDB ( const ProblemDescDB & db )

copy constructor

Copy constructor manages sharing of dbRep and incrementing of referenceCount.

References ProblemDescDB::dbRep, and ProblemDescDB::referenceCount.

~ProblemDescDB ( )
destructor

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

References Dakota::Dakpddb, ProblemDescDB::dbRep, and ProblemDescDB::referenceCount.

ProblemDescDB ( BaseConstructor, ParallelLibrary & parallel_lib ) [protected]

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

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

13.122.3 Member Function Documentation

ProblemDescDB operator= ( const ProblemDescDB & db )

assignment operator


References ProblemDescDB::dbRep, and ProblemDescDB::referenceCount.

void parse_inputs ( const ProgramOptions & prog_opts, DbCallbackFunctionPtr callback = NULL, void * callback_data = NULL )

Parses the input file or input string if present and executes callbacks. Does not perform any validation.

DB setup phase 1: parse the input file and execute callback functions if present. Rank 0 only.

DB setup phase 2: optionally insert additional data via late sets. Rank 0 only.

References Dakota::abort_handler(), ProblemDescDB::dbRep, ProblemDescDB::derived_parse_inputs(), ProgramOptions::echo_input(), ProblemDescDB::echo_input_file(), ProgramOptions::input_file(), ProgramOptions::input_string(), ProblemDescDB::parallelLib, ProblemDescDB::parse_inputs(), and ParallelLibrary::world_rank().

Referenced by Environment::parse(), and ProblemDescDB::parse_inputs().
void check_and_broadcast ( const ProgramOptions & prog_opts )
Performs check_input, broadcast, and post_process, but for now, allowing separate invocation through the public API as well.

DB setup phase 3: perform basic checks on keywords counts in current DB state, then sync to all processors.

References ProblemDescDB::broadcast(), ProblemDescDB::check_and_broadcast(), ProblemDescDB::check_input(), ProblemDescDB::dbRep, ProblemDescDB::parallelLib, ProblemDescDB::post_process(), and ParallelLibrary::world_rank().

Referenced by ProblemDescDB::check_and_broadcast(), LibraryEnvironment::done_modifying_db(), and Environment::parse().

void check_input ( )
Verifies that there is at least one of each of the required keywords in the dakota input file.

NOTE: when using library mode in a parallel application, check_input() should either be called only on world-Rank 0, or it should follow a matched send_db_buffer() / receive_db_buffer() pair.

References Dakota::abort_handler(), ProblemDescDB::check_input(), ParallelLibrary::command_line_post_run_input(), ParallelLibrary::command_line_post_run_output(), ParallelLibrary::command_line_pre_run_input(), ParallelLibrary::command_line_pre_run_output(), ParallelLibrary::command_line_run_input(), ParallelLibrary::command_line_run_output(), ParallelLibrary::command_line_user_modes(), ProblemDescDB::dataInterfaceList, ProblemDescDB::dataMethodList, ProblemDescDB::dataModelList, ProblemDescDB::dataResponsesList, ProblemDescDB::dataVariablesList, ProblemDescDB::dbRep, ProblemDescDB::environmentCntr, ProblemDescDB::parallelLib, and Dakota::strbegins().

Referenced by ProblemDescDB::check_and_broadcast(), and ProblemDescDB::check_input().

void post_process ( )
Post-processes the (minimal) input specification to assign default variables/responses specification arrays. Used by manage_inputs().

When using library mode in a parallel application, post_process() should be called on all processors following broadcast() of a minimal problem specification.

References ProblemDescDB::dbRep, and ProblemDescDB::derived_post_process().

Referenced by ProblemDescDB::check_and_broadcast().

ProblemDescDB * get_db ( ParallelLibrary & parallel_lib ) [private]

Used by the envelope constructor to instantiate the correct letter class.

Initializes dbRep to the appropriate derived type. The standard derived class constructors are invoked.

References Dakota::Dak_pddb.

Referenced by ProblemDescDB::ProblemDescDB().

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

- ProblemDescDB.hpp
- ProblemDescDB.cpp

13.123 ProcessApplicInterface Class Reference

Derived application interface class that spawns a simulation code using a separate process and communicates with it through files.

Inheritance diagram for ProcessApplicInterface:
Public Member Functions

- `ProcessApplicInterface` (const ProblemDescDB &problem_db)
  constructor
- `~ProcessApplicInterface` ()
  destructor

Protected Member Functions

- void `derived_map` (const Variables &vars, const ActiveSet &set, Response &response, int fn_eval_id)
  Called by `map()` and other functions to execute the simulation in synchronous mode. The portion of performing an evaluation that is specific to a derived class.
- void `derived_map_asynch` (const ParamResponsePair &pair)
  Called by `map()` and other functions to execute the simulation in asynchronous mode. The portion of performing an asynchronous evaluation that is specific to a derived class.
- const StringArray & `analysis_drivers` () const
  retrieve the analysis drivers specification for application interfaces
- void `file_cleanup` () const
  clean up any interface parameter/response files when aborting
- virtual void `map_bookkeeping` (pid_t pid, int fn_eval_id=0)
  bookkeeping of process and evaluation ids for asynchronous maps
- virtual pid_t `create_evaluation_process` (bool block_flag)=0
  Spawn the evaluation by managing the input filter, analysis drivers, and output filter. Called from `derived_map()` & `derived_map_asynch()`.
- void `synchronous_local_analyses` (int start, int end, int step)
  execute analyses synchronously on the local processor
- void `define_filenames` (const String &eval_id_tag)
  define modified filenames from user input by handling Unix temp file and optionally tagging with given eval_id_tag
- void `write_parameters_files` (const Variables &vars, const ActiveSet &set, const Response &response, const int id)
  write the parameters data and response request data to one or more parameters files (using one or more invocations of `write_parameters_file()`) in either standard or aprepro format
- void `read_results_files` (Response &response, const int id, const String &eval_id_tag)
  read the response object from one or more results files using full eval_id_tag passed
Protected Attributes

- **bool fileTagFlag**
  
  flags tagging of parameter/results files

- **bool fileSaveFlag**
  
  flags retention of parameter/results files

- **bool commandLineArgs**
  
  flag indicating use of passing of filenames as command line arguments to the analysis drivers and input/output filters

- **bool apreproFlag**
  
  flag indicating use of the APREPRO (the Sandia "A PRE PROcessor" utility) format for parameter files

- **bool multipleParamsFiles**
  
  flag indicating the need for separate parameters files for multiple analysis drivers

- **std::string iFilterName**
  
  the name of the input filter (input filter user specification)

- **std::string oFilterName**
  
  the name of the output filter (output filter user specification)

- **std::vector<String> programNames**
  
  the names of the analysis code programs (analysis drivers user specification)

- **std::string specifiedParamsFileName**
  
  the name of the parameters file from user specification

- **std::string paramsFileName**
  
  the parameters file name actually used (modified with tagging or temp files)

- **std::string specifiedResultsFileName**
  
  the name of the results file from user specification

- **std::string resultsFileName**
  
  the results file name actually used (modified with tagging or temp files)

- **std::string fullEvalId**
  
  complete evalIdTag, possibly including hierarchical tagging and final eval id, but not program numbers, for passing to write parameters files

- **bool allowExistingResults**
  
  by default analysis code interfaces delete results files if they exist; user may override with this flag and we’ll try to gather and only fork if needed

- **std::string curWorkdir**
  
  working directory when useWorkdir is true

- **std::map< int, std::pair< std::string, std::string > > fileNameMap**
  
  stores parameters and results file names used in spawning function evaluations. Map key is the function evaluation identifier.

- **bool useWorkdir**
  
  whether to use a new or specified work directory

- **std::string workDir**
  
  its name, if specified...

- **bool dirTag**
whether to tag the working directory

- bool dirSave
  whether dir_save was specified
- bool dirDel
  whether to delete the directory when Dakota terminates
- bool haveWorkdir
  for dirTag, whether we have workDir
- std::string templateDir
  template directory (if specified)
- StringArray templateFiles
  template files (if specified)
- bool templateCopy
  whether to force a copy (versus link) every time
- bool templateReplace
  whether to replace existing files
- bool haveTemplateDir
  state variable for template directory
- std::string dakDir
  Dakota directory (if needed)

Private Member Functions

- void write_parameters_file (const Variables &vars, const ActiveSet &set, const Response &response, const std::string &prog, const std::vector<String> &ancomps, const std::string &params_fname)
  write the variables, active set vector, derivative variables vector, and analysis components to the specified parameters file in either standard or aprepro format

Private Attributes

- String2DArray analysisComponents
  the set of optional analysis components used by the analysis drivers (from the analysis_components interface specification)

13.123.1 Detailed Description

Derived application interface class that spawns a simulation code using a separate process and communicates with it through files.

ProcessApplicInterface is subclassed for process handles or file completion testing.

13.123.2 Member Function Documentation

void synchronous_local_analyses ( int start, int end, int step ) [inline], [protected]
execute analyses synchronously on the local processor

Execute analyses synchronously in succession on the local processor (start to end in step increments). Modeled after ApplicationInterface::synchronous_local_evaluations().
References ApplicationInterface::synchronous_local_analysis().
CHAPTER 13. CLASS DOCUMENTATION

Referenced by ProcessHandleApplicInterface::create_evaluation_process().
The documentation for this class was generated from the following files:

- ProcessApplicInterface.hpp
- ProcessApplicInterface.cpp

13.124 ProcessHandleApplicInterface Class Reference

Derived application interface class that spawns a simulation code using a separate process, receives a process identifier, and communicates with the spawned process through files.

Inheritance diagram for ProcessHandleApplicInterface:

```
  Interface
   /
  ApplicationInterface
   /
ProcessApplicInterface
   /
ProcessHandleApplicInterface
      /
ForkApplicInterface  SpawnApplicInterface
```

Public Member Functions

- ProcessHandleApplicInterface (const ProblemDescDB &problem_db)
  constructor
- ~ProcessHandleApplicInterface ()
  destructor

Protected Member Functions

- int synchronous_local_analysis (int analysis_id)
- void init_communicators_checks (int max_eval_concurrency)
- void set_communicators_checks (int max_eval_concurrency)
- void map_bookkeeping (pid_t pid, int fn_eval_id)
  bookkeeping of process and evaluation ids for asynchronous maps
- pid_t create_evaluation_process (bool block_flag)
- virtual pid_t create_analysis_process (bool block_flag, bool new_group)=0
  spawn a child process for an analysis component within an evaluation
- virtual size_t wait_local_analyses ()=0
  wait for asynchronous analyses on the local processor, completing at least one job
- virtual size_t test_local_analyses_send (int analysis_id)=0
  test for asynchronous analysis completions on the local processor and return results for any completions by sending messages
13.124. PROCESSHANDLEAPPLICINTERFACE CLASS REFERENCE

• virtual void join_evaluation_process_group (bool new_group)
  create (if new_group) and join the process group for asynch evaluations
• virtual void join_analysis_process_group (bool new_group)
  create (if new_group) and join the process group for asynch analyses
• virtual void evaluation_process_group_id (pid_t pgid)
  set evalProcGroupId
• virtual pid_t evaluation_process_group_id () const
  return evalProcGroupId
• virtual void analysis_process_group_id (pid_t pgid)
  set analysisProcGroupId
• virtual pid_t analysis_process_group_id () const
  return analysisProcGroupId
• void process_local_evaluation (PRPQueue &prp_queue, const pid_t pid)
  Common processing code used by \{wait,test\}_local_evaluations.
• void check_wait (pid_t pid, int status)
  check the exit status of a forked process and abort if an error code was returned
• void asynchronous_local_analyses (int start, int end, int step)
  execute analyses asynchronously on the local processor
• void serve_analyses_asynch ()
  serve the analysis scheduler and execute analysis jobs asynchronously
• void ifilter_argument_list ()
  set argList for execution of the input filter
• void ofilter_argument_list ()
  set argList for execution of the output filter
• void driver_argument_list (int analysis_id)
  set argList for execution of the specified analysis driver

Protected Attributes

• std::map< pid_t, int > evalProcessIdMap
  map of fork process id’s to function evaluation id’s for asynchronous evaluations
• std::map< pid_t, int > analysisProcessIdMap
  map of fork process id’s to analysis job id’s for asynchronous analyses
• std::vector< std::string > argList
  an array of strings for use with execvp(const char *, char * const *). These are converted to an array of const char*’s in fork_program().

13.124.1 Detailed Description

Derived application interface class that spawns a simulation code using a separate process, receives a process identifier, and communicates with the spawned process through files.

ProcessHandleApplicInterface is subclassed for fork/execvp/waitpid (Unix) and spawnvp (Windows).
13.124.2 Member Function Documentation

```cpp
int synchronous_local_analysis ( int analysis_id ) [inline], [protected], [virtual]
```

This code provides the derived function used by ApplicationInterface::serve_analyses_synch() as well as a convenience function for ProcessHandleApplicInterface::synchronous_local_analyses() below.

Reimplemented from ApplicationInterface.

References ProcessHandleApplicInterface::create_analysis_process(), and ProcessHandleApplicInterface::driver_argument_list().

```cpp
void init_communicators_checks ( int max_eval_concurrency ) [inline], [protected], [virtual]
```

No derived interface plug-ins, so perform construct-time checks. However, process init issues as warnings since some contexts (e.g., HierarchSurrModel) initialize more configurations than will be used.

Reimplemented from ApplicationInterface.

References ApplicationInterface::check_multiprocessor_analysis(), and ApplicationInterface::check_multiprocessor_asynchronous().

```cpp
void set_communicators_checks ( int max_eval_concurrency ) [inline], [protected], [virtual]
```

Process run-time issues as hard errors.

Reimplemented from ApplicationInterface.

References Dakota::abort_handler(), ApplicationInterface::check_multiprocessor_analysis(), and ApplicationInterface::check_multiprocessor_asynchronous().

```cpp
pid_t create_evaluation_process ( bool block_flag ) [protected], [virtual]
```

Manage the input filter, 1 or more analysis programs, and the output filter in blocking or nonblocking mode as governed by block_flag. In the case of a single analysis and no filters, a single fork is performed, while in other cases, an initial fork is reforked multiple times. Called from derived_map() with block_flag == BLOCK and from derived_map_asynch() with block_flag == FALL_THROUGH. Uses create_analysis_process() to spawn individual program components within the function evaluation.

Implements ProcessApplicInterface.

References Dakota::abort_handler(), ProcessHandleApplicInterface::analysis_process_group_id(), ApplicationInterface::analysisServerId, ApplicationInterface::asyncLocalAnalysisConcurrency, ApplicationInterface::asyncLocalAnalysisFlag, ProcessHandleApplicInterface::asyncLocalAnalyses(), ParallelLibrary::barrier_e(), ProcessApplicInterface::commandLineArgs, ProcessHandleApplicInterface::create_analysis_process(), ProcessHandleApplicInterface::driver_argument_list(), ApplicationInterface::eaDedMasterFlag, ApplicationInterface::evalCommRank, ApplicationInterface::evalCommSize, ProcessHandleApplicInterface::evalProcessIdMap, ProcessHandleApplicInterface::evaluation_process_group_id(), ProcessHandleApplicInterface::ifilter_argument_list(), ProcessApplicInterface::iFilterName, ProcessHandleApplicInterface::join_evaluation_process_group(), ApplicationInterface::master_dynamic_schedule_analyses(), ProcessApplicInterface::multipleParamsFiles, ApplicationInterface::numAnalysisDrivers, ApplicationInterface::numAnalysisServers, ProcessHandleApplicInterface::oFilterName, ApplicationInterface::parallelLib, ProcessApplicInterface::paramsFileName, ProcessApplicInterface::programNames, ProcessApplicInterface::resultsFileName, ProcessHandleApplicInterface::serve_analyses_asynch(), ApplicationInterface::serve_analyses_synch(), ApplicationInterface::suppressOutput, and ProcessApplicInterface::synchronous_local_analyses().

```cpp
void check_wait ( pid_t pid, int status ) [protected]
```

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

References Dakota::abort_handler().

Referenced by ForkApplicInterface::create_analysis_process(), SpawnApplicInterface::test_local_analyses_send(), SpawnApplicInterface::test_local_evaluations(), ForkApplicInterface::wait(), SpawnApplicInterface::wait_local_analyses(), and SpawnApplicInterface::wait_local_evaluations().

```c
void asynchronous_local_analyses ( int start, int end, int step ) [protected]
```

execute analyses asynchronously on the local processor

Schedule analyses asynchronously on the local processor using a dynamic scheduling approach (start to end in step increments). Concurrency is limited by asynchLocalAnalysisConcurrency. Modeled after ApplicationInterface::asynchronous_local_evaluations(). NOTE: This function should be elevated to ApplicationInterface if and when another derived interface class supports asynchronous local analyses.

References Dakota::abort_handler(), ProcessHandleApplicInterface::analysisProcessIdMap, ApplicationInterface::asynchLocalAnalysisConcurrency, ProcessHandleApplicInterface::create_analysis_process(), ProcessHandleApplicInterface::driver_argument_list(), ApplicationInterface::numAnalysisDrivers, and ProcessHandleApplicInterface::wait_local_analyses().

Referenced by ProcessHandleApplicInterface::create_evaluation_process().

```c
void serve_analyses_async ( ) [protected]
```

serve the analysis scheduler and execute analysis jobs asynchronously

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

References Dakota::abort_handler(), ProcessHandleApplicInterface::analysisProcessIdMap, ApplicationInterface::asynchLocalAnalysisConcurrency, ProcessHandleApplicInterface::create_analysis_process(), ProcessHandleApplicInterface::driver_argument_list(), ParallelLibrary::irecv_ea(), ApplicationInterface::numAnalysisDrivers, ApplicationInterface::parallelLib, ParallelLibrary::recv_ea(), ParallelLibrary::test(), and ProcessHandleApplicInterface::test_local_analyses_send().

Referenced by ProcessHandleApplicInterface::create_evaluation_process().

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

- ProcessHandleApplicInterface.hpp
- ProcessHandleApplicInterface.cpp

### 13.125 ProgramOptions Class Reference

**ProgramOptions** stores options whether from the CLH or from library user; initially valid only on worldRank = 0, but then broadcast in ParallelLibrary::manage_outputs_restart.

**Public Member Functions**

- **ProgramOptions ()**
default constructor (needed for default environment ctors and could be used by library clients to late update data)

- **ProgramOptions** (int argc, char ∗argv[, int world_rank])
  - standard constructor that uses a CommandLineHandler to parse user options

- const String & **input_file** () const
  - Dakota input file base name (no tag)

- const String & **input_string** () const
  - alternate Dakota input string literal

- bool **echo_input** () const
  - is input echo specified?

- const String & **parser_options** () const
  - (deprecated) NIDR parser options

- String **output_file** () const
  - output (user-provided or default) file base name (no tag)

- const String & **error_file** () const
  - error file base name (no tag)

- const String & **read_restart_file** () const
  - restart file base name (no tag)

- size_t **stop_restart_evals** () const
  - eval ID at which to stop reading restart

- String **write_restart_file** () const
  - write restart (user-provided or default) file base name (no tag)

- bool **help** () const
  - is help mode active?

- bool **version** () const
  - is version mode active?

- bool **check** () const
  - is check mode active?

- bool **pre_run** () const
  - is pre-run mode active?

- bool **run** () const
  - is run mode active?

- bool **post_run** () const
  - is post-run mode active?

- bool **user_modes** () const
  - are any non-default, user-specified run modes active?

- const String & **pre_run_input** () const
  - filename for pre-run input

- const String & **pre_run_output** () const
  - filename for pre-run output

- const String & **run_input** () const
  - filename for run input

- const String & **run_output** () const
  - filename for run output
• const String & post_run_input () const
  filename for post-run input
• const String & post_run_output () const
  filename for post-run output
• bool proceed_to_instantiate () const
  whether steps beyond help/version are requested (instantiation required)
• bool proceed_to_run () const
  Whether steps beyond check are requested.
• bool user_stdoutredirect () const
  whether the user/client code requested a redirect of stdout
• void input_file (const String &in_file)
  set Dakota input file base name (no tag)
• void input_string (const String &in_string)
  set alternate Dakota input string literal
• void echo_input (bool echo_flag)
  set whether to echo input to output
• void output_file (const String &out_file)
  set base file name for Dakota output
• void error_file (const String &err_file)
  set base file name for Dakota errors
• void read_restart_file (const String &read_rst)
  set base file name for restart file from which to read
• void stop_restart_evals (size_t stop_rst)
  set eval ID at which to stop reading restart
• void write_restart_file (const String &write_rst)
  set base file name for restart file to write
• void help (bool help_flag)
  set true to print help information and exit
• void version (bool version_flag)
  set true to print version information and exit
• void check (bool check_flag)
  set true to check input and instantiate objects, then exit
• void pre_run (bool pre_run_flag)
  set to enable/disable pre-run phase
• void run (bool run_flag)
  set to enable/disable run phase
• void post_run (bool post_run_flag)
  set to enable/disable post-run phase
• void pre_run_input (const String &pre_run_in)
  Specify the pre-run phase input filename.
• void pre_run_output (const String &pre_run_out)
  Specify the pre-run phase output filename.
• void run_input (const String &run_in)
CHAPTER 13. CLASS DOCUMENTATION

Specify the run phase input filename.
• void run_output (const String &run_out)
  Specify the run phase output filename.
• void post_run_input (const String &post_run_in)
  Specify the post-run phase input filename.
• void post_run_output (const String &post_run_out)
  Specify the post-run phase output filename.
• void read (MPIUnpackBuffer &s)
  helper function for reading some class data from MPI buffer
• void write (MPIPackBuffer &s) const
  helper function for writing some class data to MPI buffer

Private Member Functions
• void parse_environment_options ()
  any environment variables affecting global behavior get read here
• void manage_run_modes (const CommandLineHandler &clh)
  retrieve run mode options from the CLH
• void split_filenames (const char *filenames, std::string &input_filename, std::string &output_filename)
  manage pre/run/post filenames
• void validate ()
  verify consistency of user settings (helpful for library mode especially)

Private Attributes
• String inputFile
  Dakota input file name, e.g., "dakota.in".
• String inputString
  alternate input means for library clients: input string (mutually exclusive with input file)
• bool echoInput
  whether to echo client's input file at parse
• String parserOptions
  Deprecated option for NIDR parser options.
• String outputFile
  Dakota output base file name, e.g., "dakota.out".
• String errorFile
  Dakota error base file name, e.g., "dakota.err".
• String readRestartFile
  e.g., "dakota.old.rst"
• size_t stopRestartEvals
  eval number at which to stop restart read
• String writeRestartFile
  e.g., "dakota.new.rst"
• bool helpFlag
whether to print help message and exit
• bool versionFlag
  whether to print version message and exit
• 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 any user run modes 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

13.125.1 Detailed Description

ProgramOptions stores options whether from the CLH or from library user; initially valid only on worldRank = 0, but then broadcast in ParallelLibrary::manage_outputs_restart.

13.125.2 Member Function Documentation

void split_filenames ( const char *filenames, std::string &input_filename, std::string &output_filename )
[private]
manage pre/run/post filenames
Tokenize colon-delimited input and output filenames, returns unchanged strings if tokens not found.
Referenced by ProgramOptions::manage_run_modes().
The documentation for this class was generated from the following files:
• ProgramOptions.hpp
• ProgramOptions.cpp
13.126 PStudyDACE Class Reference

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

Inheritance diagram for PStudyDACE:

```
  Iterator
  Analyzer
  PStudyDACE
    DDACEDesignCompExp
    FSUDesignCompExp
    ParamStudy
    PSUADEDesignCompExp
```

Protected Member Functions

- **PStudyDACE (ProblemDescDB &problem_db, Model &model)**
  constructor
- **PStudyDACE (unsigned short method_name, Model &model)**
  alternate constructor for instantiations "on the fly"
- **~PStudyDACE ()**
  destructor
- **void core_run ()**
  core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post
- **void print_results (std::ostream &s)**
  print the final iterator results
- **virtual void extract_trends ()=0**
  Mapping of the core_run() virtual function for the PStudy/DACE branch.
- **void volumetric_quality (int ndim, int num_samples, double *sample_points)**
  Calculation of volumetric quality measures.

Protected Attributes

- **SensAnalysisGlobal pStudyDACESensGlobal**
  initialize statistical post processing
- **bool volQualityFlag**
  flag which specifies evaluation of volumetric quality measures
- **bool varBasedDecompFlag**
  flag which specifies calculating variance based decomposition sensitivity analysis metrics

Private Attributes

- **double chiMeas**
  quality measure
- **double dMeas**
13.126. PSTUDYDACE CLASS REFERENCE

quality measure
- double hMeas

quality measure
- double tauMeas

Additional Inherited Members

13.126.1 Detailed Description

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

The PSTudyDACE base class manages common data and functions, such as those involving the best solutions located during the parameter set evaluations or the printing of final results.

13.126.2 Member Function Documentation

void core_run ( ) [inline], [protected], [virtual]

core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post
Virtual run function for the iterator class hierarchy. All derived classes need to redefine it.
Reimplemented from Iterator.
References Analyzer::bestVarsRespMap, and PSTudyDACE::extract_trends().

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

print the final iterator results
This virtual function provides additional iterator-specific final results outputs beyond the function evaluation summary printed in finalize_run().
Reimplemented from Analyzer.
References PSTudyDACE::chiMeas, Analyzer::compactMode, Model::continuous_variable_labels(), SensAnalysisGlobal::correlations_computed(), Model::discrete_int_variable_labels(), Model::discrete_real_variable_labels(), PSTudyDACE::dMeas, PSTudyDACE::hMeas, Iterator::iteratedModel, Analyzer::numLSqTerms, Analyzer::num_ObjFns, SensAnalysisGlobal::print_correlations(), Analyzer::print_results(), Analyzer::print_sobol_indices(), PSTudyDACE::pStudyDACE::SensGlobal, Model::response_labels(), PSTudyDACE::tauMeas, PSTudyDACE::var-BasedDecompFlag, and PSTudyDACE::volQualityFlag.

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

Calculation of volumetric quality measures.
Calculation of volumetric quality measures developed by FSU.
References PSTudyDACE::chiMeas, PSTudyDACE::dMeas, PSTudyDACE::hMeas, and PSTudyDACE::tauMeas.
Referenced by FSUDesignCompExp::get_parameter_sets(), and DDACEDesignCompExp::get_parameter_sets().
The documentation for this class was generated from the following files:

- DakotaPStudyDACE.hpp
- DakotaPStudyDACE.cpp
13.127 PSUADEDesignCompExp Class Reference

Wrapper class for the PSUADE library.

Inheritance diagram for PSUADEDesignCompExp:

```
   Iterator
     |
     V
  Analyzer
     |
     V
PStudyDACE
     |
     V
PSUADEDesignCompExp
```

Public Member Functions

- `PSUADEDesignCompExp (ProblemDescDB &problem_db, Model &model)`
  - primary constructor for building a standard DACE iterator
- `~PSUADEDesignCompExp ()`
  - destructor
- `void pre_run ()`
  - pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori
- `void post_input ()`
  - read tabular data for post-run mode
- `void extract_trends ()`
  - Mapping of the `core_run()` virtual function for the PStudy/DACE branch.
- `void post_run (std::ostream &s)`
  - post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way
- `int num_samples () const`
- `void sampling_reset (int min_samples, bool all_data_flag, bool stats_flag)`
  - reset sampling iterator to use at least min_samples
- `unsigned short sampling_scheme () const`
  - return sampling name
- `void vary_pattern (bool pattern_flag)`
  - sets varyPattern in derived classes that support it
- `void get_parameter_sets (Model &model)`
  - Returns one block of samples (ndim * num_samples)

Private Member Functions

- `void enforce_input_rules ()`
  - enforce sanity checks/modifications for the user input specification
Private Attributes

- int samplesSpec
  initial specification of number of samples
- int numSamples
  current number of samples to be evaluated
- const UShortArray & varPartitionsSpec
  number of partitions in each variable direction
- int numPartitions
  number of partitions to pass to PSUADE (levels = partitions + 1)
- bool allDataFlag
  flag which triggers the update of allVars/allResponses for use by Iterator::all_variables() and Iterator::all_responses()
- size_t numDACERuns
  counter for number of executions for this object
- bool varyPattern
  flag for generating a sequence of seed values within multiple get_parameter_sets() calls so that the sample sets are not repeated, but are still repeatable
- const int seedSpec
  the user seed specification for the random number generator (allows repeatable results)
- int randomSeed
  current seed for the random number generator

Additional Inherited Members

13.127.1 Detailed Description

Wrapper class for the PSUADE library.

The PSUADEDesignCompExp class provides a wrapper for PSUADE, a C++ design of experiments library from Lawrence Livermore National Laboratory. Currently this class only includes the PSUADE Morris One-at-a-time (MOAT) method to uniformly sample the parameter space spanned by the active bounds of the current Model. It returns all generated samples and their corresponding responses as well as the best sample found.

13.127.2 Constructor & Destructor Documentation

PSUADEDesignCompExp( ProblemDescDB & problem_db, Model & model )

primary constructor for building a standard DACE iterator

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

References Dakota::abort_handler(), Iterator::maxEvalConcurrency, Iterator::methodName, and PSUADE-DesignCompExp::numSamples.

13.127.3 Member Function Documentation

void pre_run( ) [virtual]

pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori
pre-run phase, which a derived iterator may optionally reimplement; when not present, pre-run is likely inte-
grated into the derived run function. This is a virtual function; when re-implementing, a derived class must call
its nearest parent’s pre_run(), if implemented, typically before performing its own implementation steps.
Reimplemented from Iterator.
References PSUADEDesignCompExp::get_parameter_sets(), and Iterator::iteratedModel.

```cpp
void post_run ( std::ostream & s ) [virtual]
```
post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/
Responses and perform final analysis phase in a standalone way

Post-run phase, which a derived iterator may optionally reimplement; when not present, post-run is likely
integrated into run. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s
post_run(), typically after performing its own implementation steps.
Reimplemented from Analyzer.
References Dakota::abort_handler(), Analyzer::allResponses, Analyzer::allSamples, Model::continuous_lower
bounds(), Model::continuous_upper_bounds(), Iterator::iteratedModel, Analyzer::numContinuousVars, Analyzer::
umFunctions, PSUADEDesignCompExp::numSamples, and Analyzer::post_run().

```cpp
int num_samples ( ) const [inline], [virtual]
```
Return current number of evaluation points. Since the calculation of samples, collocation points, etc. might be
costly, provide a default implementation here that backs out from the maxEvalConcurrency.
Reimplemented from Analyzer.
References PSUADEDesignCompExp::numSamples.

```cpp
void enforce_input_rules ( ) [private]
```
enforce sanity checks/modifications for the user input specification

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

References Dakota::abort_handler(), Iterator::methodName, Analyzer::numContinuousVars, PSUADEDesign-
CompExp::numPartitions, PSUADEDesignCompExp::numSamples, and PSUADEDesignCompExp::varPartitions-
Spec.

Referenced by PSUADEDesignCompExp::get_parameter_sets().
The documentation for this class was generated from the following files:

- PSUADEDesignCompExp.hpp
- PSUADEDesignCompExp.cpp

### 13.128 PythonInterface Class Reference

Inheritance diagram for PythonInterface:
Public Member Functions

- **PythonInterface** (const `ProblemDescDB` &problem_db)
  
  Constructor
- **~PythonInterface** ()
  
  Destructor

Protected Member Functions

- virtual int **derived_map_ac** (const String &ac_name)
  
  Execute an analysis code portion of a direct evaluation invocation
- **int python_run** (const String &ac_name)
  
  Direct interface to Python via API, BMA 07/02/07
- template<class ArrayT , class Size >
  
  bool **python_convert_int** (const ArrayT &src, Size size, PyObject **dst)
  
  Convert arrays of integer types to Python list or numpy array
- bool **python_convert** (const RealVector &src, PyObject **dst)
  
  Convert RealVector to Python list or numpy array
- bool **python_convert** (const RealVector &c_src, const IntVector &di_src, const RealVector &dr_src, PyObject **dst)
  
  Convert RealVector + IntVector + RealVector to Python mixed list or numpy double array
- template<class StringArrayT >
  
  bool **python_convert_strlist** (const StringArrayT &src, PyObject **dst)
  
  Convert labels
- bool **python_convert** (const StringMultiArray &c_src, const StringMultiArray &di_src, const StringMultiArray &dr_src, PyObject **dst)
  
  Convert all labels to single list
- bool **python_convert** (PyObject *pyv, RealVector &rv, const int &dim)
  
  Convert python [list of int or float] or [numpy array of double] to RealVector (for funs)
- bool **python_convert** (PyObject *pyv, double *rv, const int &dim)
  
  Convert python [list of int or float] or [numpy array of double] to double[], for use as helper in converting gradients
- bool **python_convert** (PyObject *pym, RealMatrix &rm)
  
  Convert python [list of lists of int or float] or [numpy array of dbl] to RealMatrix (for gradients)
- bool **python_convert** (PyObject *pym, RealSymMatrix &rm)
convert python [list of lists of int or float] or [numpy array of dbl] to RealMatrix (used as helper in Hessian conversion)

• bool python_convert (PyObject *pyma, RealSymMatrixArray &rma)
  convert python [list of lists of lists of int or float] or [numpy array of double] to RealSymMatrixArray (for Hessians)

Protected Attributes

• bool userNumpyFlag
  whether the user requested numpy data structures in the input file

13.128.1 Detailed Description

Specialization of DirectApplicInterface to link to Python analysis drivers. Includes convenience functions to map data to/from Python

13.128.2 Member Function Documentation

int derived_map_ac ( const String & ac_name ) [protected], [virtual]
  execute an analysis code portion of a direct evaluation invocation
  Python specialization of derived analysis components.
  Reimplemented from DirectApplicInterface.
  References ApplicationInterface::analysisServerId, and PythonInterface::python_run().

bool python_convert_int ( const ArrayT & src, Size sz, PyObject ** dst ) [protected]
  convert arrays of integer types to Python list or numpy array
  convert all integer array types including IntVector, ShortArray, and SizetArray to Python list of ints or numpy array of ints
  References PythonInterface::userNumpyFlag.
  Referenced by PythonInterface::python_run().
  The documentation for this class was generated from the following files:
  • PythonInterface.hpp
  • PythonInterface.cpp

13.129 RecastBaseConstructor Struct Reference

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

Public Member Functions

• RecastBaseConstructor (int=0)
  C++ structs can have constructors.
13.130. **RecastModel Class Reference**

Derived model class which provides a thin wrapper around a sub-model in order to recast the form of its inputs and/or outputs.

Inheritance diagram for RecastModel:

```
Model
    RecastModel
```

**Public Member Functions**

- **RecastModel** (const `Model` &sub_model, const `Sizet2DArray` &vars_map_indices, const `SizetArray` &vars_comps_totals, `size_t` num_recast_primary_fns, `size_t` num_recast_secondary_fns, `size_t` recast_secondary_offset)
  - *standard constructor*

- **RecastModel** (const `Model` &sub_model, const `SizetArray` &vars_comps_totals, `size_t` num_recast_primary_fns, `size_t` num_recast_secondary_fns, `size_t` recast_secondary_offset)
  - *alternate constructor*

- **~RecastModel** ()
  - *destructor*

  - *completes initialization of the RecastModel after alternate construction*

---

13.129.1 **Detailed Description**

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

`RecastBaseConstructor` is used to overload the constructor used for on-the-fly `Model` instantiations. Putting this struct here avoids circular dependencies.

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

- dakota.global_defs.hpp
• void inverse_mappings (void(*inv_vars_map)(const Variables &sub_model_vars, Variables &recast_vars),
  void(*inv_set_map)(const Variables &sub_model_vars, const ActiveSet &sub_model_set, ActiveSet &recast_set),
  void(*inv_pri_resp_map)(const Variables &recast_vars, const Variables &sub_model_vars, const Response &recast RESP, Response &sub_model_resp),
  void(*inv_sec_resp_map)(const Variables &recast_vars, const Variables &sub_model_vars, const Response &recast RESP, Response &sub_model_resp))
  
  provide optional inverse mappings

• void transform_variables (const Variables &recast_vars, Variables &sub_model_vars)
  perform transformation of Variables (recast -> sub-model)

• void transform_set (const Variables &recast_vars, const ActiveSet &recast_set, ActiveSet &sub_model_set)
  into sub_model_set for use with subModel.

• void transform_response (const Variables &recast_vars, const Variables &sub_model_vars, const Response &sub_model_resp, Response &recastResp)
  perform transformation of Response (sub-model -> recast)

• void inverse_transform_variables (const Variables &sub_model_vars, Variables &recast_vars)
  perform inverse transformation of Variables (sub-model -> recast)

• void inverse_transform_set (const Variables &sub_model_vars, const ActiveSet &sub_model_set, ActiveSet &recast_set)
  into sub_model_set for use with subModel.

• void inverse_transform_response (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_resp)
  perform inverse transformation of Response (recast -> sub-model)

• void submodel_supports_derivative_estimation (bool sed_flag)
  override the submodel's derivative estimation behavior

Protected Member Functions

• void derived_compute_response (const ActiveSet &set)
  portion of compute_response() specific to RecastModel (forward to subModel.compute_response())

• void derived_asynch_compute_response (const ActiveSet &set)
  portion of asynch_compute_response() specific to RecastModel (forward to subModel.asynch_compute_response())

• const IntResponseMap & derived_synchronize ()
  portion of synchronize() specific to RecastModel (forward to subModel.synchronize())

• const IntResponseMap & derived_synchronize_nowait ()
  portion of synchronize_nowait() specific to RecastModel (forward to subModel.synchronize_nowait())

• Iterator & subordinate_iterator ()
  return sub-iterator, if present, within subModel

• Model & subordinate_model ()
  return subModel

• Model & surrogate_model ()
  return surrogate model, if present, within subModel

• Model & truth_model ()
  return truth model, if present, within subModel

• void derived_subordinate_models (ModelList &ml, bool recurse_flag)
  add subModel to list and recurse into subModel
void update_from_subordinate_model (bool recurse_flag=true)
  pass request to subModel if recursing and then update from it

Interface & derived interface ()
  return subModel interface

void primary_response_fn_weights (const RealVector &wts, bool recurse_flag=true)
  set the relative weightings for multiple objective functions or least squares terms and optionally recurses into sub-
  Model

void surrogate_function_indices (const IntSet &surr_fn_indices)
  update the subModel’s surrogate response function indices (DataFitSurrModel::surrogateFnIndices)

void surrogate_response_mode (short mode)
  update the subModel’s surrogate response mode (SurrogateModel::responseMode)

void build_approximation ()
  builds the subModel approximation

bool build_approximation (const Variables &vars, const IntResponsePair &response_pr)
  builds the subModel approximation

void update_approximation (bool rebuild_flag)
  replaces data in the subModel approximation

void update_approximation (const Variables &vars, const IntResponsePair &response_pr, bool rebuild_flag)
  replaces data in the subModel approximation

void update_approximation (const VariablesArray &vars_array, const IntResponseMap &resp_map, bool rebuild_flag)
  replaces data in the subModel approximation

void append_approximation (bool rebuild_flag)
  appends data to the subModel approximation

void append_approximation (const Variables &vars, const IntResponsePair &response_pr, bool rebuild_flag)
  appends data to the subModel approximation

void append_approximation (const VariablesArray &vars_array, const IntResponseMap &resp_map, bool rebuild_flag)
  appends data to the subModel approximation

void pop_approximation (bool save_surr_data, bool rebuild_flag=false)
  remove the previous data set addition to a surrogate (e.g., due to a previous append_approximation() call); flag
  manages storing of surrogate data for use in a subsequent restore_approximation()

void restore_approximation ()
  restore a previous approximation data state within a surrogate

bool restore_available ()
  query for whether a trial increment is restorable within a surrogate

void finalize_approximation ()
  finalize an approximation by applying all previous trial increments

void store_approximation ()
  move the current approximation into storage for later combination

void combine_approximation (short corr_type)
  combine the current approximation with one previously stored

std::vector< Approximation > & approximations ()
Retrieve the set of Approximations from the subModel

- `const RealVectorArray & approximation_coefficients()` retrieve the approximation coefficients from the subModel

- `void approximation_coefficients(const RealVectorArray &approx_coeffs)` set the approximation coefficients within the subModel

- `const RealVector & approximation_variances(const Variables &vars)` retrieve the approximation variances from the subModel

- `const Pecos::SurrogateData & approximation_data(size_t index)` retrieve the approximation data from the subModel

- `void component_parallel_mode(short mode)`
  
  *RecastModel* only supports parallelism in subModel, so this virtual function redefinition is simply a sanity check.

- `String local_eval_synchronization()` return subModel local synchronization setting

- `int local_eval_concurrency()` return subModel local evaluation concurrency

- `bool derived_master_overload()` const
  
  flag which prevents overloading the master with a multiprocessor evaluation (request forwarded to subModel)

- `void derived_init_communicators(int max_eval_concurrency, bool recurse_flag=true)` set up *RecastModel* for parallel operations (request forwarded to subModel)

- `void derived_init_serial()`
  
  set up *RecastModel* for serial operations (request forwarded to subModel).

- `void derived_set_communicators(int max_eval_concurrency, bool recurse_flag=true)` set active parallel configuration within subModel

- `void derived_free_communicators(int max_eval_concurrency, bool recurse_flag=true)`
  deallocate communicator partitions for the *RecastModel* (request forwarded to subModel)

- `void serve(int max_eval_concurrency)`
  
  Service subModel job requests received from the master. Completes when a termination message is received from `stop_servers()`.

- `void stop_servers()` executed by the master to terminate subModel server operations when *RecastModel* iteration is complete.

- `void inactive_view(short view, bool recurse_flag=true)` update the Model’s inactive view based on higher level (nested) context and optionally recurse into subModel

- `const String & interface_id()` const
  
  return the subModel interface identifier

- `int evaluation_id()` const
  
  return the current evaluation id for the *RecastModel* (request forwarded to subModel)

- `void set_evaluation_reference()`
  
  set the evaluation counter reference points for the *RecastModel* (request forwarded to subModel)

- `void fine_grained_evaluation_counters()` request fine-grained evaluation reporting within subModel

- `void print_evaluation_summary(std::ostream &s, bool minimal_header=false, bool relative_count=true)` const
  
  print the evaluation summary for the *RecastModel* (request forwarded to subModel)

- `virtual void eval_tag_prefix(const String &eval_id_str)`
  
  set the hierarchical eval ID tag prefix
Private Member Functions

- void initialize_data_from_submodel ()
  
  code shared among constructors to initialize base class data from submodel

- void update_from_sub_model ()
  
  update current variables/labels/bounds/targets from subModel

Private Attributes

- Model subModel
  
  the sub-model underlying the function pointers

- Sizet2DArray varsMapIndices
  
  For each subModel variable, identifies the indices of the recast variables used to define it (maps RecastModel variables to subModel variables; data is packed with only the variable indices employed rather than a sparsely filled N_sm x N_r matrix)

- bool nonlinearVarsMapping
  
  boolean set to true if the variables mapping involves a nonlinear transformation. Used in transform_set() to manage the requirement for gradients within the Hessian transformations. This does not require a BoolDeque for each individual variable, since response gradients and Hessians are managed per function, not per variable.

- bool respMapping
  
  set to true if non-NULL primaryRespMapping or secondaryRespMapping are supplied

- Sizet2DArray primaryRespMapIndices
  
  For each recast primary function, identifies the indices of the subModel functions used to define it (maps subModel response to RecastModel Response).

- Sizet2DArray secondaryRespMapIndices
  
  For each recast secondary function, identifies the indices of the subModel functions used to define it (maps subModel response to RecastModel response).

- BoolDequeArray nonlinearRespMapping
  
  array of BoolDeques, one for each recast response function. Each BoolDeque defines which subModel response functions contribute to the recast function using a nonlinear mapping. Used in transform_set() to augment the subModel function value/gradient requirements.

- IntActiveSetMap recastSetMap
  
  map of recast active set passed to derived_asynch_compute_response(). Needed for currentResponse update in synchronization routines.

- IntVariablesMap recastVarsMap
  
  map of recast variables used by derived_asynch_compute_response(). Needed for primaryRespMapping() and secondaryRespMapping() in synchronization routines.

- IntVariablesMap subModelVarsMap
  
  map of subModel variables used by derived_asynch_compute_response(). Needed for primaryRespMapping() and secondaryRespMapping() in synchronization routines.

- IntResponseMap recastResponseMap
  
  map of recast responses used by RecastModel::derived_synchronize() and RecastModel::derived_synchronize_nowait()

- void(* variablesMapping )(const Variables &recast_vars, Variables &sub_model_vars)
  
  holds pointer for variables mapping function passed in ctor/initilize

- void(* setMapping )(const Variables &recast_vars, const ActiveSet &recast_set, ActiveSet &sub_model_set)
CHAPTER 13. CLASS DOCUMENTATION

holds pointer for set mapping function passed in ctor/initialize
- void(* primaryRespMapping )(const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)

holds pointer for primary response mapping function passed in ctor/initialize

holds pointer for secondary response mapping function passed in ctor/initialize
- void(* invVarsMapping )(const Variables &sub_model_vars, Variables &recast_vars)

holds pointer for optional inverse variables mapping function passed in inverse_mappings()
- void(* invSetMapping )(const Variables &sub_model_vars, const ActiveSet &sub_model_set, ActiveSet &recast_set)

holds pointer for optional inverse set mapping function passed in inverse_mappings()
- void(* invPriRespMapping )(const Variables &recast_vars, const Variables &sub_model_vars, const Response &recast_resp, Response &sub_model_resp)

holds pointer for optional inverse primary response mapping function passed in inverse_mappings()
- void(* invSecRespMapping )(const Variables &recast_vars, const Variables &sub_model_vars, const Response &recast_resp, Response &sub_model_resp)

holds pointer for optional inverse secondary response mapping function passed in inverse_mappings()

Additional Inherited Members

13.130.1 Detailed Description

Derived model class which provides a thin wrapper around a sub-model in order to recast the form of its inputs and/or outputs.

The RecastModel class uses function pointers to allow recasting of the subModel input/output into new problem forms. This is currently used to recast SBO approximate subproblems, but can be used for multiobjective, input/output scaling, and other problem modifications in the future.

13.130.2 Constructor & Destructor Documentation

RecastModel ( const Model &sub_model, const Sizet2DArray &vars_map_indices, const SizetArray &vars_comps_totals, bool nonlinear_vars_mapping, void(*)(const Variables &recast_vars, Variables &sub_model_vars) variables_map, void(*)(const Variables &recast_vars, const ActiveSet &sub_model_set) set_map, const Sizet2DArray &primary_resp_map_indices, const Sizet2DArray &secondary_resp_map_indices, size_t recast_secondary_offset, const BoolDequeArray &nonlinear_resp_mapping, void(*)(const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response) primary_resp_map, void(*)(const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response) secondary_resp_map )

standard constructor

Default recast model constructor. Requires full definition of the transformation. Parameter vars_comps_totals indicates the number of each type of variable \{ 4 types \} x \{ 3 domains \} in the recast variable space. Note: recast_secondary_offset is the start index for equality constraints, typically num nonlinear ineq constraints.

References Dakota::abort_handler(), Constraints::copy(), Variables::copy(), Response::copy(), Model::current_response(), Model::current_variables(), Model::currentResponse, Model::currentVariables, Variables::cv(), Response::function_gradients(), Response::function_hessians(), RecastModel::initialize_data_from_submodel(), RecastModel::nonlinearRespMapping, Response::num_functions(), Model::num_functions(), Constraints::num_linear_eq_constraints(),
Constraints::num_linear_ineq_constraints, Constraints::num_nonlinear_eq_constraints, Constraints::num_nonlinear_ineq_constraints, Model::numDerivVars, Model::numFns, RecastModel::primaryRespMapIndices, RecastModel::primaryRespMapping, Constraints::reshape, Response::reshape, RecastModel::respMapping, RecastModel::secondaryRespMapIndices, RecastModel::secondaryRespMapping, Model::userDefinedConstraints, Model::userDefinedConstraints, Variables::variables_components_totals, RecastModel::variablesMapping, and Variables::view().

RecastModel (const Model & sub_model, const SizetArray & vars_comps_totals, size_t num_recast_primary_fns, size_t num_recast_secondary_fns, size_t recast_secondary_offset)

alternate constructor

This alternate constructor defers initialization of the function pointers until a separate call to initialize(), and accepts the minimum information needed to construct currentVariables, currentResponse, and userDefinedConstraints. The resulting model is sufficiently complete for passing to an Iterator. Parameter vars_comps_totals indicates the number of each type of variable {4 types} x {3 domains} in the recast variable space. Note: recast_secondary_offset is the start index for equality constraints, typically num nonlinear ineq constraints.

References Constraints::copy(), Variables::copy(), Response::copy(), Model::current_response, Model::currentVariables, Variables::cv(), Response::function_gradients(), Response::function_hessians(), RecastModel::initialize_data_from_submodel, Model::num_functions, Constraints::num_linear_eq_constraints, Constraints::num_linear_ineq_constraints, Constraints::num_nonlinear_eq_constraints, Constraints::num_nonlinear_ineq_constraints, Model::numDerivVars, Model::numFns, Constraints::reshape, Response::reshape, RecastModel::subModel, Model::userDefinedConstraints, Variables::variables_components_totals, RecastModel::variablesMapping, and Variables::view().

13.130.3 Member Function Documentation

void initialize (const Sizet2DArray & vars_map_indices, bool nonlinear_vars_mapping, void(*)(const Variables & recast_vars, Variables & sub_model_vars) variables_map, void(*)(const Variables & recast_vars, const ActiveSet & recast_set, ActiveSet & sub_model_set) set_map, const Sizet2DArray & primaryResp_map_indices, const Sizet2DArray & secondaryResp_map_indices, const BoolDequeArray & nonlinearResp_mapping, void(*)(const Variables & sub_model_vars, const Variables & recast_vars, const Response & sub_model_response, Response & recast_response) primaryResp_map, void(*)(const Variables & sub_model_vars, const Variables & recast_vars, const Response & sub_model_response, Response & recast_response) secondaryResp_map)

completes initialization of the RecastModel after alternate construction

This function is used for late initialization of the recasting functions. It is used in concert with the alternate constructor.


Referenced by EffGlobalMinimizer::minimize_surrogates_on_model, NonDLocalReliability::mpp_search, NonDGlobalReliability::optimize_gaussian_process, NonDLocalInterval::quantify_uncertainty, NonDGlobalInterval::quantify_uncertainty, and Minimizer::scale_model().

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

portion of compute_response() specific to RecastModel (forward to subModel.compute_response())
void eval_tag_prefix ( const String & eval_id_str ) [inline], [protected], [virtual]

set the hierarchical eval ID tag prefix

RecastModel just forwards any tags to its subModel
Reimplemented from Model.
References Model::eval_tag_prefix(), and RecastModel::subModel.

void update_from_sub_model ( ) [private]

update current variables/labels/bounds/targets from subModel
Update inactive values and labels in currentVariables and inactive bound constraints in userDefinedConstraints from variables and constraints data within subModel.
References Model::aleatDistParams, Model::aleatory_distribution_parameters(), Constraints::continuous_lower_bounds(), Model::continuous_lower_bounds(), Constraints::continuous_upper_bounds(), Model::continuous_upper_bounds(), Variables::continuous_variable_labels(), Model::continuous_variable_labels(), Variables::continuous_variables(), Model::currentResponse, Model::currentVariables, Model::discrete_design_set_int_values(), Model::discrete_design_set_real_values(), Constraints::discrete_int_lower_bounds(), Model::discrete_int_lower_bounds(), Constraints::discrete_int_upper_bounds(), Model::discrete_int_upper_bounds(), Variables::discrete_int_variable_labels(), Model::discrete_int_variable_labels(), Variables::discrete_int_variables(), Model::discrete_real_lower_bounds(), Model::discrete_real_lower_bounds(), Variables::discrete_real_variable_labels(), Model::discrete_real_variable_labels(), Variables::discrete_real_variables(), Model::discrete_real_upper_bounds(), Model::discrete_real_upper_bounds(), Variables::discrete_real_variable_labels(), Model::discrete_real_variable_labels(), Variables::discrete_real_variables(), Model::discrete_state_set_int_values(), Model::discrete_state_set_real_values(), Model::discreteDesignSetIntValues, Model::discreteDesignSetRealValues, Model::discreteStateSetIntValues, Model::discreteStateSetRealValues, Model::epistDistParams, Model::epistemic_distribution_parameters(), Response::function_label(), Constraints::inactive_continuous_lower_bounds(), Model::inactive_continuous_lower_bounds(), Constraints::inactive_continuous_upper_bounds(), Model::inactive_continuous_upper_bounds(), Variables::inactive_continuous_variable_labels(), Model::inactive_continuous_variable_labels(), Variables::inactive_continuous_variables(), Model::inactive_discrete_int_lower_bounds(), Model::inactive_discrete_int_lower_bounds(), Variables::inactive_discrete_int_variable_labels(), Model::inactive_discrete_int_variable_labels(), Variables::inactive_discrete_int_variables(), Constraints::inactive_discrete_real_lower_bounds(), Model::inactive_discrete_real_lower_bounds(), Variables::inactive_discrete_real_variable_labels(), Model::inactive_discrete_real_variable_labels(), Variables::inactive_discrete_real_variables(), Constraints::linear_eq_constraint_coeffs(), Model::linear_eq_constraint_coeffs(), Constraints::linear_eq_constraint_targets(), Model::linear_eq_constraint_targets(), Constraints::linear_ineq_constraint_coeffs(), Model::linear_ineq_constraint_coeffs(), Constraints::linear_ineq_constraint_lower_bounds(), Model::linear_ineq_constraint_lower_bounds(), Constraints::linear_ineq_constraint_upper_bounds(), Model::linear_ineq_constraint_upper_bounds(), Constraints::nonlinear_eq_constraint_targets(), Model::nonlinear_eq_constraint_targets(), Constraints::nonlinear_ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_lower_bounds(), Constraints::nonlinear_ineq_constraint_upper_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Constraints::nonlinear_ineq_constraint_targets(), Model::nonlinear_ineq_constraint_targets(), Constraints::nonlinear_eq_constraint_targets(), Model::nonlinear_eq_constraint_targets(), Constraints::nonlinear_ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_lower_bounds(), Constraints::nonlinear_ineq_constraint_upper_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Constraints::nonlinear_ineq_constraint_targets(), Model::nonlinear_ineq_constraint_targets(), Constraints::nonlinear_eq_constraint_targets(), Model::nonlinear_eq_constraint_targets(), Constraints::nonlinear_ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_lower_bounds(), Constraints::nonlinear_ineq_constraint_upper_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Constraints::nonlinear_ineq_constraint_targets(), Model::nonlinear_ineq_constraint_targets(), Constraints::nonlinear_eq_constraint_targets(), Model::nonlinear_eq_constraint_targets(), Constraints::nonlinear_ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_lower_bounds(), Constraints::nonlinear_ineq_constraint_upper_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Constraints::nonlinear_ineq_constraint_targets(), Model::nonlinear_ineq_constraint_targets(), Constraints::nonlinear_eq_constraint_targets(), Model::nonlinear_eq_constraint_targets(), Constraints::nonlinear_ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_lower_bounds(), Constraints::nonlinear_ineq_constraint_upper_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Constraints::nonlinear_ineq_constraint_targets(), Model::nonlinear_ineq_constraint_targets(), Constraints::nonlinear_eq_constraint_targets(), Model::nonlinear_eq_constraint_targets(), Constraints::nonlinear_ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_lower_bounds(), Constraints::nonlinear_ineq_constraint_upper_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Constraints::nonlinear_ineq_constraint_targets(), Model::nonlinear_ineq_constraint_targets(), Constraints::nonlinear_eq_constraint_targets(), Model::nonlinear_eq_constraint_targets(), Constraints::nonlinear_ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_lower_bounds(), Constraints::nonlinear_ineq_constraint_upper_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Constraints::nonlinear_ineq_constraint_targets(), Model::nonlinear_ineq_constraint_targets(), Constraints::nonlinear_eq_constraint_targets(), Model::nonlinear_eq_constraint_targets(), Constraints::nonlinear_ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_lower_bounds(), Constraints::nonlinear_ineq_constraint_upper_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Constraints::nonlinear_ineq_constraint_targets(), Model::nonlinear_ineq_constraint_targets(), Constraints::nonlinear_eq_constraint_targets(), Model::nonlinear_eq_constraint_targets(), Constraints::nonlinear_ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_lower_bounds(), Constraints::nonlinear_ineq_constraint_upper_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Constraints::nonlinear_ineq_constraint_targets(), Model::nonlinear_ineq_constraint_targets(), Constraints::nonlinear_eq_constraint_targets(), Model::nonlinear_eq_constraint_targets(), Constraints::nonlinear_ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_lower_bounds(), Constraints::nonlinear_ineq_constraint_upper_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Constraints::nonlinear_ineq_constraint_targets(), Model::nonlinear_ineq_constraint_targets(), Constraints::nonlinear_eq_constraint_targets(), Model::nonlinear_eq_constraint_targets(), Constraints::nonlinear_ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_lower_bounds(), Constraints::nonlinear_ineq_constraint_upper_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Constraints::nonlinear_ineq_constraint_targets(), Model::nonlinear_ineq_constraint_targets(), Constraints::nonlinear_eq_constraint_targets(), Model::nonlinear_eq_constraint_targets(), Constraints::nonlinear_ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_lower_bounds(), Constraints::nonlinear_ineq_constraint_upper_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Constraints::nonlinear_ineq_constraint_targets(), Model::nonlinear_ineq_constraint_targets(), Constraints::nonlinear_eq_constraint_targets(), Model::nonlinear_eq_constraint_targets(), Constraints::nonlinear_ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_lower_bounds(), Constraints::nonlinear_ineq_constraint_upper_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Constraints::nonlinear_ineq_constraint_targets(), Model::nonlinear_ineq_constraint_targets(), Constraints::nonlinear_eq_constraint_targets(), Model::nonlinear_eq_constraint_targets(), Constraints::nonlinear_ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_lower_bounds(), Constraints::nonlinear_ineq_constraint_upper_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Constraints::nonlinear_ineq_constraint_targets()
13.131 RelaxedVarConstraints Class Reference

Derived class within the Constraints hierarchy which employs relaxation of discrete variables.

Inheritance diagram for RelaxedVarConstraints:

```
Constraints
    RelaxedVarConstraints
```

Public Member Functions

- **RelaxedVarConstraints** (const SharedVariablesData &svd)
  lightweight constructor
- **RelaxedVarConstraints** (const ProblemDescDB &problem_db, const SharedVariablesData &svd)
  standard constructor
- **~RelaxedVarConstraints** ()
  destructor
- void **write** (std::ostream &s) const
  write a variable constraints object to an std::ostream
- void **read** (std::istream &s)
  read a variable constraints object from an std::istream

Protected Member Functions

- void **reshape** (const SizetArray &vc_totals)
  reshape the lower/upper bound arrays within the Constraints hierarchy
- void **build_active_views** ()
  construct active views of all variables bounds arrays
- void **build_inactive_views** ()
  construct inactive views of all variables bounds arrays
Additional Inherited Members

13.131.1 Detailed Description

Derived class within the `Constraints` hierarchy which employs relaxation of discrete variables.

Derived variable constraints classes take different views of the design, uncertain, and state variable types and the continuous and discrete domain types. The `RelaxedVarConstraints` derived class combines continuous and discrete domain types through integer relaxation. The branch and bound method uses this approach (see Variables-::get_variables(problem_db) for variables type selection; variables type is passed to the `Constraints` constructor in `Model`).

13.131.2 Constructor & Destructor Documentation

```cpp
RelaxedVarConstraints ( const ProblemDescDB & problem_db, const SharedVariablesData & svd )
```

standard constructor

In this class, a relaxed data approach is used in which continuous and discrete arrays are combined into a single continuous array (integrality is relaxed; the converse of truncating reals is not currently supported but could be in the future if needed). Iterators which use this class include: BranchBndOptimizer.

References `Constraints::allContinuousLowerBnds`, `Constraints::allContinuousUpperBnds`, `Constraints::build_views()`, `SharedVariablesData::components_totals()`, `Dakota::copy_data_partial()`, `ProblemDescDB::get_iv()`, `ProblemDescDB::get_rv()`, `Constraints::manage_linear_constraints()`, `Dakota::merge_data_partial()`, `Constraints::sharedVarsData`, `SharedVariablesData::vc_lookup()`, and `SharedVariablesData::view()`.

13.131.3 Member Function Documentation

```cpp
void reshape ( const SizetArray & vc_totals ) [protected], [virtual]
```

resizes the lower/upper bound arrays within the `Constraints` hierarchy

Reimplemented from `Constraints`.

References `Constraints::allContinuousLowerBnds`, `Constraints::allContinuousUpperBnds`, `Constraints::build_views()`, and `Constraints::reshape()`.

Referenced by `RelaxedVarConstraints::RelaxedVarConstraints()`.

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

- `RelaxedVarConstraints.hpp`
- `RelaxedVarConstraints.cpp`

13.132 RelaxedVariables Class Reference

Derived class within the `Variables` hierarchy which employs the relaxation of discrete variables.

Inheritance diagram for `RelaxedVariables`:

```
  Variables
  ↓
RelaxedVariables
```
13.132. RELAXEDVARIABLES CLASS REFERENCE

Public Member Functions

- `RelaxedVariables (const ProblemDescDB &problem_db, const std::pair<short, short> &view)`
  standard constructor
- `RelaxedVariables (const SharedVariablesData &svd)`
  lightweight constructor
- `~RelaxedVariables ()`
  destructor

Protected Member Functions

- `void read (std::istream &s)`
  read a variables object from an std::istream
- `void write (std::ostream &s) const`
  write a variables object to an std::ostream
- `void write_aprepro (std::ostream &s) const`
  write a variables object to an std::ostream in aprepro format
- `void read_tabular (std::istream &s)`
  write a variables object in tabular format to an std::ostream
- `void write_tabular (std::ostream &s) const`
- `void reshape (const SizetArray &vc_totals)`
  reshapes an existing Variables object based on the incoming variablesComponents
- `void build_active_views ()`
  construct active views of all variables arrays
- `void build_inactive_views ()`
  construct inactive views of all variables arrays

Additional Inherited Members

13.132.1 Detailed Description

Derived class within the Variables hierarchy which employs the relaxation of discrete variables.

Derived variables classes take different views of the design, uncertain, and state variable types and the continuous and discrete domain types. The RelaxedVariables derived class combines continuous and discrete domain types but separates design, uncertain, and state variable types. The branch and bound method uses this approach (see Variables::get_variables(problem_db)).

13.132.2 Constructor & Destructor Documentation

RelaxedVariables ( const ProblemDescDB & problem_db, const std::pair<short, short> & view )

standard constructor

In this class, a relaxed data approach is used in which continuous and discrete arrays are combined into a single continuous array (integrality is relaxed; the converse of truncating reals is not currently supported but could be in the future if needed). Iterators/strategies which use this class include: BranchBndOptimizer. Extract fundamental variable types and labels and merge continuous and discrete domains to create aggregate arrays and views.

References Variables::allContinuousVars, Variables::build_views(), SharedVariablesData::components_totals(), Dakota::copy_data_partial(), ProblemDescDB::get_iv(), ProblemDescDB::get_rv(), Dakota::merge_data_partial(), Variables::sharedVarsData, SharedVariablesData::vc_lookup(), and SharedVariablesData::view().
13.132.3 Member Function Documentation

```cpp
void read_tabular ( std::istream & s ) [protected], [virtual]
```

Presumes variables object is appropriately sized to receive data
Reimplemented from Variables.
References Variables::allContinuousVars, SharedVariablesData::components_totals(), and Variables::shared- VarsData.
The documentation for this class was generated from the following files:

- RelaxedVariables.hpp
- RelaxedVariables.cpp

13.133 Response Class Reference

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

Public Member Functions

- **Response ()**
  default constructor
- **Response (const Variables &vars, const ProblemDescDB &problem_db)**
  standard constructor built from problem description database
- **Response (const ActiveSet &set)**
  alternate constructor using limited data
- **Response (const Response &response)**
  copy constructor
- **~Response ()**
  destructor
- **Response operator= (const Response &response)**
  assignment operator
- **size_t num_functions () const**
  return the number of response functions
- **const ActiveSet & active_set () const**
  return the active set
- **void active_set (const ActiveSet &set)**
  set the active set
- **const ShortArray & active_set_request_vector () const**
  return the active set request vector
- **void active_set_request_vector (const ShortArray &asrv)**
  set the active set request vector
- **const SizetArray & active_set_derivative_vector () const**
  return the active set derivative vector
- **void active_set_derivative_vector (const SizetArray &asdv)**
  set the active set derivative vector
- **const String & responses_id () const**
return the response identifier

- const String & function_label (size_t i) const
  return a response function identifier string

- const StringArray & function_labels () const
  return the response function identifier strings

- void function_label (const String &label, size_t i)
  set a response function identifier string

- void function_labels (const StringArray &labels)
  set the response function identifier strings

- const Real & function_value (size_t i) const
  return a function value

- Real & function_value_view (size_t i)
  return a "view" of a function value for updating in place

- const RealVector & function_values () const
  return all function values

- RealVector function_values_view ()
  return all function values as a view for updating in place

- void function_value (const Real &function_val, size_t i)
  set a function value

- void function_values (const RealVector &function_vals)
  set all function values

- const Real * function_gradient (const int &i) const
  return the i-th function gradient as a const Real*

- RealVector function_gradient_view (const int &i)
  return the i-th function gradient as a SerialDenseVector Teuchos::View (shallow copy) for updating in place

- RealVector function_gradient_copy (const int &i)
  return the i-th function gradient as a SerialDenseVector Teuchos::Copy (deep copy)

- const RealMatrix & function_gradients () const
  return all function gradients

- RealMatrix function_gradients_view ()
  return all function gradients as a view for updating in place

- void function_gradient (const RealVector &function_grad, const int &i)
  set a function gradient

- void function_gradients (const RealMatrix &function_grads)
  set all function gradients

- const RealSymMatrix & function_hessian (size_t i) const
  return the i-th function Hessian

- RealSymMatrix function_hessian_view (size_t i)
  return the i-th function Hessian as a Teuchos::View (shallow copy) for updating in place

- const RealSymMatrixArray & function_hessians () const
  return all function Hessians

- RealSymMatrixArray function_hessians_view ()
  return all function Hessians as Teuchos::Views (shallow copies) for updating in place
• void function_hessian (const RealSymMatrix &function_hessian, size_t i)
  set a function Hessian
• void function_hessians (const RealSymMatrixArray &function_hessians)
  set all function Hessians
• void read (std::istream &s)
  read a response object from an std::istream
• void write (std::ostream &s) const
  write a response object to an std::ostream
• void read.annotated (std::istream &s)
  read a response object in annotated format from an std::istream
• void write.annotated (std::ostream &s) const
  write a response object in annotated format to an std::ostream
• void read.tabular (std::istream &s)
  read responseRep::functionValues in tabular format from an std::istream
• void write.tabular (std::ostream &s) const
  write responseRep::functionValues in tabular format to an std::ostream
• void read (MPIUnpackBuffer &s)
  read a response object from a packed MPI buffer
• void write (MPIPackBuffer &s) const
  write a response object to a packed MPI buffer
• Response copy () const
  a deep copy for use in history mechanisms
• int data.size ()
  handle class forward to corresponding body class member function
• void read.data (double ∗response_data)
  handle class forward to corresponding body class member function
• void write.data (double ∗response_data)
  handle class forward to corresponding body class member function
• void overlay (const Response &response)
  handle class forward to corresponding body class member function
• void update (const Response &response)
  Used in place of operator= when only results data updates are desired (functionValues/functionGradients/function-
Hessians are updated, ASV/labels/id’s/etc. are not). Care is taken to allow different derivative array sizing between
the two response objects.
• void update (const RealVector &source_fn_vals, const RealMatrix &source_fn_grads, const RealSymMatrix-
Array &source_fn_hessians, const ActiveSet &source_set)
  Overloaded form which allows update from components of a response object. Care is taken to allow different
derivative array sizing.
• void update.partial (size_t start_index_target, size_t num_items, const Response &response, size_t start_-
index_source)
  partial update of this response object from another response object. The response objects may have different
numbers of response functions.
• void update.partial (size_t start_index_target, size_t num_items, const RealVector &source_fn_vals, const
RealMatrix &source_fn_grads, const RealSymMatrixArray &source_fn_hessians, const ActiveSet &source-
_set, size_t start_index_source)
Overloaded form which allows partial update from components of a response object. The response objects may have different numbers of response functions.

- **void reshape**(size_t num_fns, size_t num_params, bool grad_flag, bool hess_flag)
  - reshapes 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)

- **template<class Archive>** void **load**(Archive &ar, const unsigned int version)
- **template<class Archive>** void **save**(Archive &ar, const unsigned int version) const

**Private Member Functions**

- **template<class Archive>** void **load**(Archive &ar, const unsigned int version)
  - read a Response from an archive

- **template<class Archive>** void **save**(Archive &ar, const unsigned int version) const
  - write a Response to an archive

- **BOOST_SERIALIZATION_SPLIT_MEMBER()** ResponseRep &responseRep
  - pointer to the body (handle-body idiom)

**Friends**

- **class boost:serialization::access**
- **bool operator==(const Response &resp1, const Response &resp2)**
  - equality operator

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

13.133.1 Detailed Description

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

The **Response** class is a container class for an abstract set of functions (functionValues) and their first (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.
CHAPTER 13. CLASS DOCUMENTATION

13.133.2 Member Function Documentation

void load ( Archive & ar, const unsigned int version )
Implementation of serialization load for the Response handle

void save ( Archive & ar, const unsigned int version ) const
Implementation of serialization save for the Response handle

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

• DakotaResponse.hpp
• DakotaResponse.cpp

13.134 ResponseRep Class Reference

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

Public Member Functions

• template<class Archive , typename OrdinalType , typename ScalarType >
  void write_sdm_col (Archive &ar, int col, const Teuchos::SerialDenseMatrix< OrdinalType, ScalarType >&sdm) const
  convenience function to write a serial dense matrix column to an Archive

• template<class Archive , typename OrdinalType , typename ScalarType >
  void read_sdm_col (Archive &ar, int col, Teuchos::SerialDenseMatrix< OrdinalType, ScalarType >&sdm)
  convenience function to read a serial dense matrix column from an Archive

• template<class Archive >
  void load (Archive &ar, const unsigned int version)
• template<class Archive >
  void save (Archive &ar, const unsigned int version) const

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 (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

template<class Archive , typename OrdinalType , typename ScalarType >
void write.sdm.col (Archive &ar, int col, const Teuchos::SerialDenseMatrix< OrdinalType, ScalarType > &sdm) const
write a column of a SerialDenseMatrix

template<class Archive , typename OrdinalType , typename ScalarType >
void read.sdm.col (Archive &ar, int col, Teuchos::SerialDenseMatrix< OrdinalType, ScalarType > &sdm)
read a column of a SerialDenseMatrix

void load (Archive &ar, const unsigned int version)
read a ResponseRep from an archive

void save (Archive &ar, const unsigned int version) const
write a ResponseRep to an archive

BOOST_SERIALIZATION_SPLIT_MEMBER () int data.size()
return the number of doubles active in response. Used for sizing double* response.data arrays passed into read.data and write.data.

void read.data (double *response.data)
read from an incoming double* array

void write.data (double *response.data)
write to an incoming double* array

void overlay (const Response &response)
add incoming response to functionValues/Gradients/Hessians

void update (const RealVector &source_fn vals, const RealMatrix &source_fn grads, const RealSymMatrix-Array &source_fn hessians, const ActiveSet &source_set)
update this response object from components of another response object

void update.partial (size_t start_index_target, size_t num_items, const RealVector &source_fn vals, const RealMatrix &source_fn grads, const RealSymMatrixArray &source_fn hessians, const ActiveSet &source_set, size_t start_index_source)
partially update this response object partial components of another response object

void reshape (size_t num_fns, size_t num.params, bool grad_flag, bool hess_flag)
rephases response data arrays

void reset ()
resets all response data to zero

void reset.inactive ()
resets all inactive response data to zero

- void active_set_request_vector (const ShortArray &asrv)
  
  set the active set request vector and verify consistent number of response functions

- void active_set_derivative_vector (const SizetArray &asdv)
  
  set the active set derivative vector and reshape functionGradients/functionHessians if needed

Private Attributes

- int referenceCount
  
  number of handle objects sharing responseRep

- RealVector functionValues
  
  abstract set of response functions

- RealMatrix functionGradients
  
  first derivatives of the response functions

- RealSymMatrixArray functionHessians
  
  second derivatives of the response functions

- ActiveSet responseActiveSet
  
  copy of the ActiveSet used by the Model to generate a Response instance

- StringArray functionLabels
  
  response function identifiers used to improve output readability

- String responsesId
  
  response identifier string from the input file

Friends

- class boost::serialization::access
  
  for serializing private data members

- class Response
  
  the handle class can access attributes of the body class directly

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

13.134.1 Detailed Description

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

The ResponseRep class is the "representation" of the response container class. It is the "body" portion of the "handle-body idiom" (see Coplien "Advanced C++", p. 58). The handle class (Response) provides for memory efficiency in management of multiple response objects through reference counting and representation sharing. The body class (ResponseRep) actually contains the response data (functionValues, functionGradients, functionHessians, etc.). The representation is hidden in that an instance of ResponseRep may only be created by Response. Therefore, programmers create instances of the Response handle class, and only need to be aware of the handle/body mechanisms when it comes to managing shallow copies (shared representation) versus deep copies (separate representation used for history mechanisms).
13.134.2 Constructor & Destructor Documentation

**ResponseRep ( const Variables & vars, const ProblemDescDB & problem_db ) [private]**

standard constructor built from problem description database

The standard constructor used by Dakota::ModelRep.

References Variables::continuous_variable_ids(), Variables::cv(), ActiveSet::derivative_vector(), ResponseRep::functionGradients, ResponseRep::functionHessians, ResponseRep::functionValues, ProblemDescDB::get sizet(), ProblemDescDB::get_string(), ActiveSet::request_vector(), and ResponseRep::responseActiveSet.

**ResponseRep ( const ActiveSet & set ) [private]**

alternate constructor using limited data

Used for building a response object of the correct size on the fly (e.g., by slave analysis servers performing execute() on a local_response). functionLabels is not needed for this purpose since it’s not passed in the MPI send/recv buffers. However, NPSOLOptimizer’s user-defined functions option uses this constructor to build best-ResponseArray.front() and bestResponseArray.front() needs functionLabels for I/O, so construction of function-Labels has been added.

References Dakota::build_labels(), ActiveSet::derivative_vector(), ResponseRep::functionGradients, ResponseRep::functionHessians, ResponseRep::functionLabels, and ActiveSet::request_vector().

13.134.3 Member Function Documentation

**void read ( std::istream & s ) [private]**

read a responseRep object from an std::istream

ASCII version of read needs capabilities for capturing data omissions or formatting errors (resulting from user error or asynch race condition) and analysis failures (resulting from nonconvergence, instability, etc.).

References ResponseRep::functionGradients, ResponseRep::functionHessians, ResponseRep::functionLabels, Dakota::re_match(), ActiveSet::request_vector(), ResponseRep::reset(), and ResponseRep::responseActiveSet.

**void read.annotated ( std::istream & s ) [private]**

read a responseRep object from an std::istream (annotated format)

read.annotated() is used for neutral file translation of restart files. Since objects are built solely from this data, annotations are used. This version closely mirrors the BiStream version.

References ResponseRep::functionGradients, ResponseRep::functionHessians, ResponseRep::functionLabels, ResponseRep::functionValues, ActiveSet::request_vector(), ResponseRep::reset(), ActiveSet::reshape(), ResponseRep::reshape(), and ResponseRep::responseActiveSet.

**void write.annotated ( std::ostream & s ) const [private]**

write a responseRep object to an std::ostream (annotated format)

write.annotated() is used for neutral file translation of restart files. Since objects need to be build solely from this data, annotations are used. This version closely mirrors the BoStream version, with the exception of the use of white space between fields.

References ActiveSet::derivative_vector(), ResponseRep::functionGradients, ResponseRep::functionHessians, ResponseRep::functionValues, ActiveSet::request_vector(), ResponseRep::responseActiveSet, and ActiveSet::write.annotated().
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 Approximation-
   Interfaces in reading samples from a file. There is insufficient data in a tabular file to build complete response
   objects; rather, the response object must be constructed a priori and then its functionValues can be set.
   References ResponseRep::functionValues.

void write_tabular ( std::ostream & s ) const [private]
write functionValues to an std::ostream (tabular format)
   write_tabular is used for output of functionValues in a tabular format for convenience in post-processing/plotting
   of DAKOTA results.
   References ResponseRep::functionValues, ActiveSet::request_vector(), ResponseRep::responseActiveSet, and
   Dakota::write_precision.

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.
   References ResponseRep::functionGradients, ResponseRep::functionHessians, ResponseRep::functionValues,
   ResponseRep::reset(), ResponseRep::reshape(), and ResponseRep::responseActiveSet.

void update ( const RealVector & source_fn_vals, const RealMatrix & source_fn_grads, const
   RealSymMatrixArray & source_fn_hessians, const ActiveSet & source_set ) [private]
update this response object from components of another response object
   Copy function values/gradients/Hessians data only. Prevents unwanted overwriting of responseActiveSet,
   functionLabels, etc. Also, care is taken to account for differences in derivative variable matrix sizing.
   References Dakota::abort_handler(), ActiveSet::derivative_vector(), ResponseRep::functionGradients, Response-
   Rep::functionHessians, ActiveSet::request_vector(), and ResponseRep::reset_inactive().

void update_partial ( size_t start_index_target, size_t num_items, const RealVector & source_fn_vals, const
   RealMatrix & source_fn_grads, const RealSymMatrixArray & source_fn_hessians, const ActiveSet &
   source_set, size_t start_index_source ) [private]
partially update this response object partial components of another response object
   Copy function values/gradients/Hessians data only. Prevents unwanted overwriting of responseActiveSet,
   functionLabels, etc. Also, care is taken to account for differences in derivative variable matrix sizing.
   References Dakota::abort_handler(), ActiveSet::derivative_vector(), ResponseRep::functionGradients, Response-
   Rep::functionHessians, ActiveSet::request_vector(), and ResponseRep::reset_inactive().

void reshape ( size_t num_fns, size_t num_params, bool grad_flag, bool hess_flag ) [private]
rehapes response data arrays
   Reshape functionValues, functionGradients, and functionHessians according to num_fns, num_params, grad-_flag, and hess_flag.
   References Dakota::build_labels(), ResponseRep::functionGradients, and ResponseRep::functionHessians.
   Referenced by ResponseRep::active_set_derivative_vector(), ResponseRep::load(), ResponseRep::read(), and
   ResponseRep::read_annotated().
void reset() [private]
reset all response data to zero
    Reset all numerical response data (not labels, ids, or active set) to zero.
    References ResponseRep::functionGradients, and ResponseRep::functionHessians.
    Referenced by ResponseRep::load(), ResponseRep::read(), and ResponseRep::read.annotated().

void reset inactive() [private]
resets all inactive response data to zero
    Used to clear out any inactive data left over from previous evaluations.
    References ResponseRep::functionGradients, and ResponseRep::functionHessians.
    Referenced by ResponseRep::update(), and ResponseRep::update_partial().

void load(Archive & ar, const unsigned int version)
    Binary version differs from ASCII version in 2 primary ways: (1) it lacks formatting. (2) the Response has not
    been sized a priori. In reading data from the binary restart file, a ParamResponsePair was constructed with its
    default constructor which called the Response default constructor. Therefore, we must first read sizing data and
    resize all of the arrays.
    References ResponseRep::functionGradients, ResponseRep::functionHessians, ResponseRep::functionLabels,
    ResponseRep::read_sdm_col(), ResponseRep::reset(), ResponseRep::reshape(), and ResponseRep::responseActive-
    Set.

void save(Archive & ar, const unsigned int version) const
    Binary version differs from ASCII version in 2 primary ways: (1) It lacks formatting. (2) In reading data from the
    binary restart file, ParamResponsePairs are constructed with their default constructor which calls the Response
    default constructor. Therefore, we must first write sizing data so that ResponseRep::read(BoStream& s) can resize
    the arrays.
    References ResponseRep::functionGradients, ResponseRep::functionHessians, ResponseRep::functionLabels,
    ResponseRep::responseActiveSet, and ResponseRep::write_sdm_col().

13.134.4 Member Data Documentation

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).
    Referenced by ResponseRep::active_set_derivative_vector(), ResponseRep::load(), ResponseRep::overlay(),
    ResponseRep::read(), ResponseRep::read_annotated(), ResponseRep::read_data(), ResponseRep::reset(), Response-
    Rep::reset_inactive(), ResponseRep::reshape(), ResponseRep::ResponseRep(), ResponseRep::save(), Response-
    Rep::update(), ResponseRep::update_partial(), ResponseRep::write_annotated(), and ResponseRep::write_data().
    The documentation for this class was generated from the following files:
    • DakotaResponse.hpp
    • DakotaResponse.cpp
13.135 ResultsDBAny Class Reference

Public Member Functions

- template<typename StoredType>
  void array_allocate (const StrStrSizet &iterator_id, const std::string &data_name, size_t array_size, const MetaDataType &metadata)
  allocate an entry with sized array of the StoredType, e.g., array across response functions or optimization results sets

- template<typename StoredType>
  void array_insert (const StrStrSizet &iterator_id, const std::string &data_name, size_t index, const StoredType &sent_data)
  insert sent_data in specified position in previously allocated array

- template<typename StoredType>
  StoredType get_data (const StrStrSizet &iterator_id, const std::string &data_name) const
  return requested data by value in StoredType

- template<typename StoredType>
  StoredType *get_data_ptr (const StrStrSizet &iterator_id, const std::string &result_key) const
  return pointer to stored data entry

- template<typename StoredType>
  const StoredType *get_array_data_ptr (const StrStrSizet &iterator_id, const std::string &data_name, size_t index) const
  return pointer to stored data at given array location

- void insert (const StrStrSizet &iterator_id, const std::string &data_name, const boost::any &result, const MetaDataType &metadata)
  record addition with metadata map

- void dump_data (std::ostream &output_stream)
  coarsely dump the data to the passed output stream

- void print_data (std::ostream &output_stream)
  pretty print the data to the passed output stream

Private Member Functions

- const ResultsValueType &lookup_data (const StrStrSizet &iterator_id, const std::string &data_name) const
  attempt to find the requested data, erroring if not found

- template<typename StoredType>
  StoredType cast_data (const boost::any &dataholder) const
  cast the reference to the any data to the requested type

- template<typename StoredType>
  const StoredType *cast_data_ptr (const boost::any *dataholder) const
  cast the pointer to the any data to the requested type
13.135. RESULTSDBANY CLASS REFERENCE

- void print_metadata (std::ostream &os, const MetaDataType &md)
  - print metadata to ostream
- void extract_data (const boost::any &dataholder, std::ostream &os)
  - determine the type of contained data and output it to ostream
- void output_data (const std::vector<double> &data, std::ostream &os)
  - output data to ostream
- void output_data (const std::vector<RealVector> &data, std::ostream &os)
  - output data to ostream
- void output_data (const std::vector<std::string> &data, std::ostream &os)
  - output data to ostream
- void output_data (const std::vector<std::vector<std::string>> &data, std::ostream &os)
  - output data to ostream
- void output_data (const std::vector<RealMatrix> &data, std::ostream &os)
  - output data to ostream
- void output_data (const RealMatrix &data, std::ostream &os)
  - output data to ostream

Private Attributes

- std::map<ResultsKeyType, ResultsValueType> iteratorData
  - core data storage (map from key to value type)

13.135.1 Detailed Description

Class: ResultsDBAny Description: A map-based container to store DAKOTA Iterator results in underlying boost::any, with optional metadata

13.135.2 Member Function Documentation

void array_insert (const StrStrSizet &iterator_id, const std::string &data_name, size_t index, const StoredType &sent_data)

insert sent_data in specified position in previously allocated array

- insert requires previous allocation, and does not allow metadata update
- References Dakota::abort_handler(), ResultsDBAny::iteratorData, and Dakota::make_key().

void insert (const StrStrSizet &iterator_id, const std::string &data_name, const boost::any &result, const MetaDataType &metadata)

record addition with metadata map

- Add or update existing entry
- References ResultsDBAny::iteratorData, and Dakota::make_key().
- Referenced by ResultsDBAny::array_allocate().
void extract_data ( const boost::any & dataholder, std::ostream & os ) [private]

determine the type of contained data and output it to ostream

  Extract the data from the held any and map to supported concrete types int double RealVector (Teuchos::-
  SerialDenseVector<int,double) RealMatrix (Teuchos::SerialDenseMatrix<int,double)
  References ResultsDBAny::output_data().
  Referenced by ResultsDBAny::dump_data(), and ResultsDBAny::print_data().
  The documentation for this class was generated from the following files:
  
  • ResultsDBAny.hpp
  • ResultsDBAny.cpp

13.136 ResultsEntry< StoredType > Class Template Reference

Class to manage in-core vs. file database lookups.

Public Member Functions

• ResultsEntry (const ResultsManager &results_mgr, const StrStrSizet &iterator_id, const std::string &data_name)
  Construct ResultsEntry containing retrieved item of StoredType.

• ResultsEntry (const ResultsManager &results_mgr, const StrStrSizet &iterator_id, const std::string &data_name, size_t array_index)
  Construct ResultsEntry to retrieve item array_index from array of StoredType.

Private Member Functions

• ResultsEntry ()
  return a reference to the stored data, whether from core or file

Private Attributes

• bool coreActive
  whether the ResultsManager has an active in-core database

• StoredType dbData
  data retrieved from file data base

• const StoredType * dbDataPtr
  non-const pointer to const data we don’t own in the core case

13.136.1 Detailed Description

template<typename StoredType> class Dakota::ResultsEntry< StoredType >

Class to manage in-core vs. file database lookups.

ResultsEntry manages database lookups. If a core database is available, will return a reference directly to the
stored data; if disk, will return reference to a local copy contained in this class. Allows disk-stored data to persist
for minimum time during lookup to support true out-of-core use cases.
13.136.2 Constructor & Destructor Documentation

ResultsEntry ( ) [private]

return a reference to the stored data, whether from core or file
default construction disallowed: data must be initialized from DB lookup if needed
The documentation for this class was generated from the following file:

- ResultsManager.hpp

13.137 ResultsID Class Reference

Get a globally unique 1-based execution number for a given iterator name (combination of methodName and methodID) for use in results DB. Each Iterator::run() call creates or increments this count for its string identifier.

Public Member Functions

- size_t increment_id (const std::string &methodName, const std::string &methodID)
  explicitly increment the iterator results ID, init to 1 if needed
- size_t get_id (const std::string &methodName, const std::string &methodID)
  get (possibly creating) a unique iterator results ID for the passed name
- size_t get_id (const std::string &methodName, const std::string &methodID) const
  get a unique iterator results ID for the passed name (const version errors if not found)

Static Public Member Functions

- static ResultsID & instance ()
  get the single unique instance of ResultsID

Private Member Functions

- ResultsID ()
  Private constructor for ResultsID.
- ~ResultsID ()
  Private destructor for ResultsID.
- ResultsID (ResultsID const &)
  Private copy constructor for ResultsID.
- ResultsID & operator= (ResultsID const &)
  Private assignment operator for ResultsID.

Private Attributes

- std::map<std::pair<std::string, std::string>, size_t> idMap
  storage for the results IDs
CHAPTER 13. CLASS DOCUMENTATION

13.137.1 Detailed Description

Get a globally unique 1-based execution number for a given iterator name (combination of methodName and methodID) for use in results DB. Each Iterator::run() call creates or increments this count for its string identifier.

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

- ResultsManager.hpp
- ResultsManager.cpp

13.138 ResultsManager Class Reference

Results manager for iterator final data.

Public Member Functions

- ResultsManager ()
  default constructor: no databases active until initialize called
- void initialize (const std::string &base_filename)
  initialize the results manager to manage an in-core database, writing to the specified file name
- bool active () const
  whether any databases are active
- void write_databases ()
  Write in-core databases to file.
- template<typename StoredType>
  void insert (const StrStrSizet &iterator_id, const std::string &data_name, const StoredType &sent_data, const MetaDataType metadata=MetaDataType())
  insert data
- void insert (const StrStrSizet &iterator_id, const std::string &data_name, StringMultiArrayConstView sma_labels, const MetaDataType metadata=MetaDataType())
- template<typename StoredType>
  void array_allocate (const StrStrSizet &iterator_id, const std::string &data_name, size_t array_size, const MetaDataType metadata=MetaDataType())
  allocate an entry with array of StoredType of array_size for future insertion; likely move to non-templated accessors for these
- template<typename StoredType>
  void array_insert (const StrStrSizet &iterator_id, const std::string &data_name, size_t index, const StoredType &sent_data)
  insert into a previously allocated array of StoredType at index specified; metadata must be specified at allocation

Public Attributes

- ResultsNames results_names
  Copy of valid results names for when manager is passed around.
Private Member Functions

- template<typename StoredType>
  StoredType core_lookup (const StrStrSizet &iterator_id, const std::string &data_name) const
  retrieve in-core entry given by id and name

- template<typename StoredType>
  StoredType * core_lookup_ptr (const StrStrSizet &iterator_id, const std::string &data_name) const
  retrieve data via pointer to avoid copy; work-around for Boost any use of pointer (could use utilib::Any)

- template<typename StoredType>
  StoredType core_lookup (const StrStrSizet &iterator_id, const std::string &data_name, size_t index) const
  retrieve data from in-core array of StoredType at given index

- template<typename StoredType>
  const StoredType * core_lookup_ptr (const StrStrSizet &iterator_id, const std::string &data_name, size_t index) const
  retrieve data via pointer to entry in in-core array

- template<typename StoredType>
  void file_lookup (StoredType &db_data, const StrStrSizet &iterator_id, const std::string &data_name) const
  retrieve requested data into provided db_data StoredType

Private Attributes

- bool coreDBActive
  whether the in-core database is active

- std::string coreDBFilename
  filename for the in-core database

- bool hdf5DBActive
  whether the file database is active

- boost::scoped_ptr<ResultsDBAny> coreDB
  In-core database, with option to flush to file at end.

- boost::shared_ptr<ResultsDBHDF5> hdf5DB
  File-based database; using shared_ptr due to potentially incomplete type and requirements for checked_delete in debug builds.

Friends

- template<typename StoredType>
  class ResultsEntry
  ResultsEntry is a friend of ResultsManager.

13.138.1 Detailed Description

Results manager for iterator final data.

The results manager provides the API for posting and retrieving iterator results data (and eventually run config/statistics). It can manage a set of underlying results databases, in or out of core, depending on configuration.

The key for a results entry is documented in results_types.hpp, e.g., tuple<std::string, std::string, size_t, std::string>
For now, using concrete types for most insertion, since underlying databases like HDF5 might need concrete
types; though template parameter for array allocation and retrieval.
All insertions overwrite any previous data.
The documentation for this class was generated from the following files:

- ResultsManager.hpp
- ResultsManager.cpp

### 13.139 ResultsNames Class Reference

List of valid names for iterator results.

**Public Member Functions**

- **ResultsNames ()**
  
  Default constructor initializes all valid names.

**Public Attributes**

- size_t namesVersion
- std::string best_cv
- std::string best_div
- std::string best_drv
- std::string best_fns
- std::string moments_std
- std::string moments_central
- std::string moments_std_num
- std::string moments_central_num
- std::string moments_std_exp
- std::string moments_central_exp
- std::string moment_cis
- std::string extreme_values
- std::string map_resp_prob
- std::string map_resp_rel
- std::string map_resp_genrel
- std::string map_prob Resp
- std::string map_rel Resp
- std::string map_genrel Resp
- std::string pdf_histograms
- std::string correl_simple_all
- std::string correl_simple_io
- std::string correl_partial_io
- std::string correl_simple_rank_all
- std::string correl_simple_rank_io
- std::string correl_partial_rank_io
- std::string pce_coeffs
- std::string pce_coeff_labels
• std::string \texttt{cv\_labels}
• std::string \texttt{div\_labels}
• std::string \texttt{drv\_labels}
• std::string \texttt{fn\_labels}

### 13.139.1 Detailed Description

List of valid names for iterator results.

All data in the \texttt{ResultsNames} class is public, basically just a struct

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

- ResultsManager.hpp

### 13.140 RichExtrapVerification Class Reference

Class for Richardson extrapolation for code and solution verification.

Inheritance diagram for RichExtrapVerification:

```
RichExtrapVerification
   |   
   v   
Verification
    |   
    v   
Analyzer
     |   
     v   
Iterator
```

**Public Member Functions**

- \texttt{RichExtrapVerification (ProblemDescDB &problem\_db, Model &model)}
  
  \textit{constructor}

- \texttt{\~ RichExtrapVerification ()}
  
  \textit{destructor}

- void \texttt{perform\_verification ()}
  
  \textit{Redefines the core\_run() virtual function for the Verification branch.}

- void \texttt{print\_results (std::ostream &s)}
  
  \textit{print the final iterator results}

**Private Member Functions**

- void \texttt{estimate\_order ()}
  
  \textit{perform a single estimation of convOrder using extrapolation()}

- void \texttt{converge\_order ()}
  
  \textit{iterate using extrapolation() until convOrder stabilizes}

- void \texttt{converge\_qoi ()}
iterate using \texttt{extrapolation()} until QOIs stabilize

- \texttt{void extrapolation} (const RealVector &refine\_triple, RealMatrix &qoi\_triples) estimate convOrder from refinement and quantity of interest (QOI) triples

- \texttt{void extrapolate\_result} (const RealVector &refine\_triple, const RealMatrix &qoi\_triples) predict the converged value based on the convergence rate and the value of Phi

Private Attributes

- unsigned short \texttt{studyType} internal code for extrapolation study type: \texttt{SUBMETHOD\_\{CONVERGE\_ORDER,CONVERGE\_QOI,ESTIMATE\_ORDER\}}

- size_t \texttt{numFactors} number of refinement factors defined from active state variables

- RealVector \texttt{initialCVars} initial reference values for refinement factors

- size_t \texttt{factorIndex} the index of the active factor

- Real \texttt{refinementRate} rate of mesh refinement (default = 2.)

- RealMatrix \texttt{convOrder} the orders of convergence of the QOIs (numFunctions by numFactors)

- RealMatrix \texttt{extrapQOI} the extrapolated value of the QOI (numFunctions by numFactors)

- RealMatrix \texttt{numErrorQOI} the numerical uncertainty associated with level of refinement (numFunctions by numFactors)

- RealVector \texttt{refinementRefPt} This is a reference point reported for the converged extrapQOI and numErrorQOI. It currently corresponds to the coarsest mesh in the final refinement triple.

Additional Inherited Members

13.140.1 Detailed Description

Class for Richardson extrapolation for code and solution verification.

The \texttt{RichExtrapVerification} class contains several algorithms for performing Richardson extrapolation.

13.140.2 Member Function Documentation

\texttt{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 \texttt{finalize\_run()}.

Reimplemented from \texttt{Verification}.

References Model::continuous\_variable\_labels(), RichExtrapVerification::convOrder, Dakota::copy\_data(), RichExtrapVerification::extrapQOI, Iterator::iteratedModel, RichExtrapVerification::numErrorQOI, Verification::print\_results(), RichExtrapVerification::refinementRate, RichExtrapVerification::refinementRefPt, and Model::response\_labels().
void estimate_order() [private]
perform a single estimation of convOrder using extrapolation()
This algorithm executes a single refinement triple and returns convergence order estimates.
Referenced by RichExtrapVerification::perform_verification().

void converge_order() [private]
iterate using extrapolation() until convOrder stabilizes
This algorithm continues to refine until the convergence order estimate converges.
Referenced by RichExtrapVerification::perform_verification().

void converge_qoi() [private]
iterate using extrapolation() until QOIs stabilize
This algorithm continues to refine until the discretization error lies within a prescribed tolerance.
Referenced by RichExtrapVerification::perform_verification().
The documentation for this class was generated from the following files:
- RichExtrapVerification.hpp
- RichExtrapVerification.cpp

13.141 ScilabInterface Class Reference

Inheritance diagram for ScilabInterface:

```
interface
  ApplicationInterface
  DirectApplicInterface
  ScilabInterface
```
Public Member Functions
- ScilabInterface (const ProblemDescDB &problem_db)
  Constructor: start Matlab engine.
- ~ScilabInterface()
  Destructor: close Matlab engine.

Protected Member Functions
- virtual int derived_map_ac (const String &ac_name)
  execute an analysis code portion of a direct evaluation invocation
- int scilab_engine_run (const String &ac_name)
  principal Scilab execute function

Protected Attributes
- int scilabEngine
  identifier for the running Scilab engine

13.141.1 Detailed Description
Specialization of DirectApplicInterface to link to Scilab analysis drivers. Includes convenience functions to map data to/from Scilab
The documentation for this class was generated from the following files:
- ScilabInterface.hpp
- ScilabInterface.cpp

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

Public Member Functions
- SensAnalysisGlobal()
  constructor
- ~SensAnalysisGlobal()
  destructor
- void compute_correlations (const VariablesArray &vars_samples, const IntResponseMap &resp_samples)
  computes four correlation matrices for input and output data simple, partial, simple rank, and partial rank
- void compute_correlations (const RealMatrix &vars_samples, const IntResponseMap &resp_samples)
  computes four correlation matrices for input and output data simple, partial, simple rank, and partial rank
- void archive_correlations (const StrStrSizet &run_identifier, ResultsManager &iterator_results, StringMultiArrayConstView cv_labels, StringMultiArrayConstView div_labels, StringMultiArrayConstView drv_labels, const StringArray &resp_labels) const
  save correlations to database
- bool correlations_computed() const
returns corrComputed to indicate whether compute_correlations() has been invoked

- void print_correlations (std::ostream &s, StringMultiArrayConstView cv_labels, StringMultiArrayConstView div_labels, StringMultiArrayConstView drv_labels, const StringArray &resp_labels) const
  prints the correlations computed in compute_correlations()

Private Member Functions

- void simple_corr (RealMatrix &total_data, bool rank_on, const int &num_in)
  computes simple correlations
- void partial_corr (RealMatrix &total_data, bool rank_on, const int &num_in)
  computes partial correlations

Static Private Member Functions

- static bool rank_sort (const int &x, const int &y)
  sort algorithm to compute ranks for rank correlations

Private Attributes

- RealMatrix simpleCorr
  matrix to hold simple raw correlations
- RealMatrix simpleRankCorr
  matrix to hold simple rank correlations
- RealMatrix partialCorr
  matrix to hold partial raw correlations
- RealMatrix partialRankCorr
  matrix to hold partial rank correlations
- size_t numFns
  number of responses
- size_t numVars
  number of inputs
- bool numericalIssuesRaw
  flag indicating numerical issues in partial raw correlation calculations
- bool numericalIssuesRank
  flag indicating numerical issues in partial rank correlation calculations
- bool corrComputed
  flag indicating whether correlations have been computed

Static Private Attributes

- static RealArray rawData = RealArray()
  array to hold temporary data before sort
13.142.1 Detailed Description

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

This class provides code for several of the sampling methods both in the NonD branch and in the PStudyD-ACE branch. Currently, the utility functions provide global sensitivity analysis through correlation calculations (e.g. simple, partial, rank, raw) as well as variance-based decomposition.

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

- SensAnalysisGlobal.hpp
- SensAnalysisGlobal.cpp

13.143 SeqHybridMetaIterator Class Reference

Method for sequential hybrid iteration using multiple optimization and nonlinear least squares methods on multiple models of varying fidelity.

Inheritance diagram for SeqHybridMetaIterator:

```
Iterator

MetaIterator

SeqHybridMetaIterator
```

Public Member Functions

- **SeqHybridMetaIterator** (ProblemDescDB &problem_db)
  
  *standard constructor*

- **SeqHybridMetaIterator** (ProblemDescDB &problem_db, Model &model)
  
  *alternate constructor*

- **~SeqHybridMetaIterator** ()
  
  *destructor*

Protected Member Functions

- void **core_run** ()
  
  *Performs the hybrid iteration by executing a sequence of iterators, using a similar sequence of models that may vary in fidelity.*

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

- 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)
13.143. SEQHYBRIDMETAITERATOR CLASS REFERENCE

used by IteratorScheduler to set the starting data for a run
• void pack_parameters_buffer (MPIPackBuffer &send_buffer, int job_index)
  used by IteratorScheduler to pack starting data for an iterator run
• void unpack_parameters_buffer (MPIUnpackBuffer &recv_buffer)
  used by IteratorScheduler to unpack starting data for an iterator run
• void pack_results_buffer (MPIPackBuffer &send_buffer, int job_index)
  used by IteratorScheduler to pack results data from an iterator run
• void unpack_results_buffer (MPIUnpackBuffer &recv_buffer, int job_index)
  used by IteratorScheduler to unpack results data from an iterator run
• void update_local_results (int job_index)
  used by IteratorScheduler to update local results arrays

Private Member Functions

• void run_sequential ()
  run a sequential hybrid
• void run_sequential_adaptive ()
  run a sequential adaptive hybrid
• void partition_sets (size_t num_sets, int job_index, size_t &start_index, size_t &job_size)
  convert num_sets and job_index into a start_index and job_size for extraction from parameterSets
• void extract_parameter_sets (int job_index, VariablesArray &partial_param_sets)
  extract partial_param_sets from parameterSets based on job_index
• void update_local_results (PRPArray &prp_results, int job_id)
  update the partial set of final results from the local iterator execution
• void initialize_iterator (const VariablesArray &param_sets)
  called by unpack_parameters_buffer(MPIUnpackBuffer) and initialize_iterator(int) to update the active Model and Iterator

Private Attributes

• String seqHybridType
  empty (default) or "adaptive"
• StringArray methodList
  the list of method name identifiers
• bool lightwtCtor
  indicates use of lightweight Iterator ctors
• IteratorArray selectedIterators
  the set of iterators, one for each entry in methodList
• ModelArray selectedModels
  the set of models, one for each iterator (if not lightweight construction)
• size_t seqCount
  hybrid sequence counter: 0 to numIterators-1
• Real progressMetric
  the amount of progress made in a single iterator++ cycle within a sequential adaptive hybrid
• Real progressThreshold
  when the progress metric falls below this threshold, the sequential adaptive hybrid switches to the next method
• PRP2DArray prpResults
  2-D array of results corresponding to numIteratorJobs, one set of results per job (iterators may return multiple final solutions)
• VariablesArray parameterSets
  1-D array of variable starting points for the iterator jobs

Additional Inherited Members

13.143.1 Detailed Description

Method for sequential hybrid iteration using multiple optimization and nonlinear least squares methods on multiple models of varying fidelity.

Sequential hybrid meta-iteration supports 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 iterator), and (2) the adaptive sequential hybrid uses adaptive stopping rules for the iterators that are controlled externally by this method. Any iterator may be used so long as it defines the notion of a final solution which can be passed as starting data for subsequent iterators.

13.143.2 Member Function Documentation

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.
  References Response::function_values(), Variables::is_null(), Response::is_null(), MetaIterator::iterSched, IteratorScheduler::messagePass, and SeqHybridMetaIterator::prpResults.

void run_sequential ( ) [private]

run a sequential hybrid
  In the sequential nonadaptive case, there is no interference with the iterators. Each runs until its own convergence criteria is satisfied. Status: fully operational.
  References Iterator::accepts_multiple_points(), ParallelLibrary::bcast_i(), ParallelLibrary::bcast_si(), Iterator::initialize_graphics(), Model::interface_id(), Iterator::iteratedModel, IteratorScheduler::iteror_message_lengths(), IteratorScheduler::iteratorCommRank, IteratorScheduler::iteratorCommSize, IteratorScheduler::iteratorScheduling, IteratorScheduler::iteratorServerId, MetaIterator::iterSched, SeqHybridMetaIterator::lightwtCtor, IteratorScheduler::messagePass, SeqHybridMetaIterator::methodList, Iterator::num_final_solutions(), IteratorScheduler::numIteratorJobs, IteratorScheduler::numIteratorServers, SeqHybridMetaIterator::pack_parameters_buffer(), IteratorScheduler::parallelLib, SeqHybridMetaIterator::parameterSets, SeqHybridMetaIterator::prpResults, ParallelLibrary::recv_s(), Iterator::response_results(), IteratorScheduler::schedule_iterators(), SeqHybridMetaIterator::selectedIterators, SeqHybridMetaIterator::selectedModels, ParallelLibrary::send_si(), SeqHybridMetaIterator::seqCount, MPIPackBuffer::size(), Iterator::summaryOutputFlag, and Iterator::variables_results().
  Referenced by SeqHybridMetaIterator::core_run().
void run_sequential_adaptive ( ) [private]
run a sequential adaptive hybrid

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

References: Iterator::finalize_run(), Iterator::initialize_graphics(), Iterator::initialize_run(), IteratorScheduler::iteratorCommRank, IteratorScheduler::iteratorServerId, MetaIterator::iterSched, SeqHybridMetaIterator::methodList, IteratorScheduler::numIteratorServers, ParallelLibrary::parallel_configuration(), IteratorScheduler::parallelLib, SeqHybridMetaIterator::progressMetric, SeqHybridMetaIterator::progressThreshold, Iterator::response_results(), IteratorScheduler::run_iterator(), SeqHybridMetaIterator::selectedIterators, SeqHybridMetaIterator::selectedModels, SeqHybridMetaIterator::seqCount, ParallelConfiguration::si_parallel_level(), Iterator::summaryOutputFlag, and Iterator::variables_results().

Referenced by SeqHybridMetaIterator::core_run().

void extract_parameter_sets ( int job_index, VariablesArray & partial_param_sets ) [inline], [private]
extract partial_param_sets from parameterSets based on job_index

This convenience function is executed on an iterator master (static scheduling) or a meta-iterator master (self scheduling) at run initialization time and has access to the full parameterSets array (this is All-Reduced for all peers at the completion of each cycle in run_sequential()).

References: SeqHybridMetaIterator::parameterSets, and SeqHybridMetaIterator::partition_sets().

Referenced by SeqHybridMetaIterator::initialize_iterator(), and SeqHybridMetaIterator::pack_parameters_buffer().

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

- SeqHybridMetaIterator.hpp
- SeqHybridMetaIterator.cpp

13.144 SerialDirectApplicInterface Class Reference

Sample derived interface class for testing serial simulator 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
- void derived_map_asynch(const Dakota::ParamResponsePair &pair)  
  no-op hides base error; job batching occurs within wait_local_evaluations()
- void wait_local_evaluations(Dakota::PRPQueue &prp_queue)  
  evaluate the batch of jobs contained in prp_queue
- void test_local_evaluations(Dakota::PRPQueue &prp_queue)  
  invokes wait_local_evaluations() (no special nowait support)
- void set_communicators_checks(int max_eval_concurrency)  
  no-op hides default run-time error checks at DirectApplicInterface level

Private Member Functions

- int rosenbrock(const Dakota::RealVector &c_vars, short asv, Dakota::Real &fn_val, Dakota::RealVector &fn_grad, Dakota::RealSymMatrix &fn_hess)  
  Rosenbrock plug-in test function.

Additional Inherited Members

13.144.1 Detailed Description

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

The plug-in SerialDirectApplicInterface resides in namespace SIM and uses a copy of rosenbrock() to perform serial parameter to response mappings. It is used to demonstrate plugging in a serial direct analysis driver into Dakota in library mode. Test input files can then use an analysis_driver of ”plugin_rosenbrock”.

13.144.2 Member Function Documentation

void test_local_evaluations(Dakota::PRPQueue & prp_queue)  [inline], [protected]  

invokes wait_local_evaluations() (no special nowait support)

For use by ApplicationInterface::serve_evaluations_asynch(), which can provide a batch processing capability within message passing schedulers (called using chain IteratorScheduler::run_iterator() – Model::serve() – ApplicationInterface::serve_evaluations() – ApplicationInterface::serve_evaluations_asynch()).

References SerialDirectApplicInterface::wait_local_evaluations().

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

- PluginSerialDirectApplicInterface.hpp
- PluginSerialDirectApplicInterface.cpp
13.145 SharedApproxData Class Reference

Base class for the shared approximation data class hierarchy.

Inheritance diagram for SharedApproxData:

```
SharedApproxData
    |   |
    v   v
SharedPecosApproxData  SharedSurfpackApproxData
```

Public Member Functions

- **SharedApproxData ()**
  
  default constructor

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

- **SharedApproxData (const String &approx_type, const UShortArray &approx_order, size_t num_vars, short data_order, short output_level)**
  
  alternate constructor for envelope

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

- **virtual ~SharedApproxData ()**
  
  destructor

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

- **virtual void build ()**
  
  builds the shared approximation data from scratch

- **virtual void rebuild ()**
  
  rebuilds the shared approximation data incrementally

- **virtual void pop (bool save_surr_data)**
  
  back out the previous increment to the shared approximation data

- **virtual bool restore_available ()**
  
  queries availability of restoration for trial set

- **virtual size_t restoration_index ()**
  
  return index of trial set within restorable bookkeeping sets

- **virtual void pre_restore ()**
  
  restore a previous state of the shared approximation data

- **virtual void post_restore ()**
  
  clean up saved storage following restoration

- **virtual size_t finalization_index (size_t i)**
  
  return index of i-th trailing trial set within restorable bookkeeping sets

- **virtual void pre_finalize ()**
  
  finalize the shared approximation data following a set of increments
• virtual void post_finalize()
  clean up saved storage following aggregation
• virtual void store()
  store the current state of the shared approximation data for later combination
• virtual void pre_combine (short corr_type)
  aggregate the shared approximation data from current and saved states
• virtual void post_combine (short corr_type)
  clean up saved storage after aggregation
• void set_bounds (const RealVector &c_lbnds, const RealVector &c_ubnds, const IntVector &di_lbnds,
  const IntVector &di_ubnds, const RealVector &dr_lbnds, const RealVector &dr_ubnds)
  set approximation lower and upper bounds (currently only used by graphics)
• SharedApproxData * data_rep () const
  returns dataRep for access to derived class member functions that are not mapped to the top SharedApproxData
  level

Protected Member Functions

• SharedApproxData (BaseConstructor, ProblemDescDB &problem_db, size_t num_vars)
  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion
  in the derived class constructors - Coplien, p. 139)
• SharedApproxData (NoDBBaseConstructor, const String &approx_type, size_t num_vars, short data_order,
  short output_level)
  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion
  in the derived class constructors - Coplien, p. 139)

Protected Attributes

• size_t numVars
  number of variables in the approximation
• String approxType
  approximation type identifier
• short buildDataOrder
  order of the data used for surrogate construction, in ActiveSet request vector 3-bit format.
• short outputLevel
  output verbosity level: {SILENT,QUIET,NORMAL,VERBOSE,DEBUG}.OUTPUT
• RealVector approxCLowerBnds
  approximation continuous lower bounds (used by 3D graphics and Surfpack KrigingModel)
• RealVector approxCUpperBnds
  approximation continuous upper bounds (used by 3D graphics and Surfpack KrigingModel)
• IntVector approxDLowerBnds
  approximation continuous lower bounds
• IntVector approxDUpperBnds
  approximation continuous upper bounds
• RealVector approxDLowerBnds
  approximation continuous lower bounds
• RealVector approxDUpperBnds
  approximation continuous upper bounds
13.145. SHAREDPAPPROXDATA CLASS REFERENCE

Private Member Functions

- **SharedApproxData * get_shared_data (ProblemDescDB & problem_db, size_t num_vars)**
  
  Used only by the standard envelope constructor to initialize dataRep to the appropriate derived type.

- **SharedApproxData * get_shared_data (const String & approx_type, const UShortArray & approx_order, size_t num_vars, short data_order, short output_level)**
  
  Used only by the alternate envelope constructor to initialize dataRep to the appropriate derived type.

Private Attributes

- **SharedApproxData * dataRep**
  
  pointer to the letter (initialized only for the envelope)

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

Friends

- class Approximation
- class TaylorApproximation
- class TANA3Approximation
- class GaussProcApproximation
- class SurfpackApproximation
- class PecosApproximation

13.145.1 Detailed Description

Base class for the shared approximation data class hierarchy.

The **SharedApproxData** class is the base class for the shared approximation data class hierarchy in DAKOTA. 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 (**SharedApproxData**) serves as the envelope and one of the derived classes (selected in **SharedApproxData::get_shared_data()**) serves as the letter.

13.145.2 Constructor & Destructor Documentation

**SharedApproxData ()**

default constructor

For the default constructor, dataRep is NULL. This makes it necessary to check for NULL in the copy constructor, assignment operator, and destructor.

Referenced by SharedApproxData::get_shared_data().

**SharedApproxData ( ProblemDescDB & problem_db, size_t num_vars )**

standard constructor for envelope

Envelope constructor only needs to extract enough data to properly execute get_shared_data, since SharedApproxData(BaseConstructor, problem_db) builds the actual base class data for the derived approximations.

References Dakota::abort_handler(), SharedApproxData::dataRep, and SharedApproxData::get_shared_data().
**SharedApproxData** (const String & *approx_type*, const UShortArray & *approx_order*, size_t *num_vars*, short *data_order*, short *output_level*)

alternate constructor for envelope

This is the alternate envelope constructor for instantiations on the fly. Since it does not have access to problem_db, it utilizes the NoDBBaseConstructor constructor chain.

References Dakota::abort_handler(), SharedApproxData::dataRep, and SharedApproxData::get_shared_data().

**SharedApproxData** (const SharedApproxData & *shared_data*)

copy constructor

Copy constructor manages sharing of dataRep and incrementing of referenceCount.

References SharedApproxData::dataRep, and SharedApproxData::referenceCount.

**~SharedApproxData** () [virtual]

destructor

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

References SharedApproxData::dataRep, and SharedApproxData::referenceCount.

**SharedApproxData** (BaseConstructor , ProblemDescDB & *problem_db*, size_t *num_vars*) [protected]

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

This constructor is the one which must build the base class data for all derived classes. get_shared_data() 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_shared_data() again). Since the letter IS the representation, its rep pointer is set to NULL (an uninitialized pointer causes problems in ~SharedApproxData).

References SharedApproxData::approxType, SharedApproxData::buildDataOrder, ProblemDescDB::get_bool(), ProblemDescDB::get_db_model_node(), ProblemDescDB::get_string(), ProblemDescDB::set_db_model_nodes(), Dakota::strbegins(), and Dakota::strends().

**SharedApproxData** (NoDBBaseConstructor , const String & *approx_type*, size_t *num_vars*, short *data_order*, short *output_level*) [protected]

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

This constructor is the one which must build the base class data for all derived classes. get_shared_data() 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_shared_data() again). Since the letter IS the representation, its rep pointer is set to NULL (an uninitialized pointer causes problems in ~SharedApproxData).

References SharedApproxData::approxType, SharedApproxData::buildDataOrder, Dakota::strbegins(), and Dakota::strends().

### 13.145.3 Member Function Documentation

**SharedApproxData operator=** (const SharedApproxData & *shared_data*)

assignment operator

References SharedApproxData::dataRep, and SharedApproxData::referenceCount.

\texttt{SharedApproxData * get\_shared\_data ( ProblemDescDB & problem\_db, size\_t num\_vars ) [private]}

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

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

References ProblemDescDB::get\_string(), SharedApproxData::SharedApproxData(), and Dakota::strends().

Referenced by SharedApproxData::SharedApproxData().

\texttt{SharedApproxData * get\_shared\_data ( const String & approx\_type, const UShortArray & approx\_order, size\_t num\_vars, short data\_order, short output\_level ) [private]}

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

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

References SharedApproxData::SharedApproxData(), and Dakota::strends().

13.145.4 Member Data Documentation

\texttt{short buildDataOrder [protected]}

order of the data used for surrogate construction, in ActiveSet request vector 3-bit format.

This setting distinguishes derivative data intended for use in construction (includes derivatives w.r.t. the build variables) from derivative data that may be approximated separately (excludes derivatives w.r.t. auxiliary variables). This setting should also not be inferred directly from the responses specification, since we may need gradient support for evaluating gradients at a single point (e.g., the center of a trust region), but not require gradient evaluations at every point.

Referenced by SharedSurfpackApproxData::add\_sd\_to\_surfdata(), TaylorApproximation::build(), TaylorApproximation::gradient(), TaylorApproximation::hessian(), TaylorApproximation::min\_coefficients(), Approximation::min\_points(), Approximation::recommended\_points(), SharedApproxData::SharedApproxData(), SharedPecosApproxData::SharedPecosApproxData(), SurfpackApproximation::SurfpackApproximation(), SurfpackApproximation::surrogates\_to\_surfdata(), TANA3Approximation::TANA3Approximation(), and TaylorApproximation::value().

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

- SharedApproxData.hpp
- SharedApproxData.cpp

13.146 SharedPecosApproxData Class Reference

Derived approximation class for global basis polynomials.

Inheritance diagram for SharedPecosApproxData:

```
SharedApproxData
  SharedPecosApproxData
```
Public Member Functions

- `SharedPecosApproxData ()`
  default constructor
- `SharedPecosApproxData (const String &approx_type, const UShortArray &approx_order, size_t num_vars, short data_order, short output_level)`
  alternate constructor
- `SharedPecosApproxData (ProblemDescDB &problem_db, size_t num_vars)`
  standard ProblemDescDB-driven constructor
- `~SharedPecosApproxData ()`
  destructor
- `void random_variables_key (const Pecos::BitArray &random_vars_key)`
  set pecosBasisApprox.randomVarsKey
- `void integration_iterator (const Iterator &iterator)`
  set pecosBasisApprox.driverRep
- `void construct_basis (const Pecos::ShortArray &u_types, const Pecos::AleatoryDistParams &adp)`
  invoke Pecos::SharedOrthogPolyApproxData::construct_basis()
- `void polynomial_basis (const std::vector<Pecos::BasisPolynomial> &poly_basis)`
  set Pecos::SharedOrthogPolyApproxData::polynomialBasis
- `const std::vector<Pecos::BasisPolynomial> & polynomial_basis () const`
  get Pecos::SharedOrthogPolyApproxData::polynomialBasis
- `void allocate (const UShort2DArray &mi)`
  set Pecos::SharedOrthogPolyApproxData::multiIndex and allocate associated arrays
- `const UShort2DArray & multi_index () const`
  get Pecos::SharedOrthogPolyApproxData::multiIndex
- `const Pecos::BitArrayULongMap & sobol_index_map () const`
  return Pecos::SharedPolyApproxData::sobolIndexMap
- `void cross_validation (bool flag)`
  invoke Pecos::SharedOrthogPolyApproxData::cross_validation()
- `void coefficients_norms_flag (bool flag)`
  invoke Pecos::SharedOrthogPolyApproxData::coefficients_norms_flag()
- `size_t expansion_terms () const`
  return Pecos::SharedOrthogPolyApproxData::expansion_terms()
- `const UShortArray & expansion_order () const`
  return Pecos::SharedOrthogPolyApproxData::expansion_order()
- `void expansion_order (const UShortArray &order)`
  invokes Pecos::SharedOrthogPolyApproxData::expansion_order(UShortArray&)
- `void increment_order ()`
  invokes Pecos::SharedOrthogPolyApproxData::increment_order()
- `void configuration_options (const Pecos::ExpansionConfigOptions &ec_options)`
  set the expansion configuration options within Pecos::SharedPolyApproxData
- `void configuration_options (const Pecos::BasisConfigOptions &bc_options)`
  set the basis configuration options within Pecos::SharedPolyApproxData
• void noise_tolerance (const RealVector &noise_tol)
  set the noise tolerance(s) for compressed sensing algorithms
• void l2_penalty (Real l2_pen)
  set the L2 penalty parameter for LASSO (elastic net variant)

Protected Member Functions

• void build ()
  builds the shared approximation data from scratch
• void rebuild ()
  rebuilds the shared approximation data incrementally
• void pop (bool save_surr_data)
  back out the previous increment to the shared approximation data
• bool restore_available ()
  queries availability of restoration for trial set
• size_t restoration_index ()
  return index of trial set within restorable bookkeeping sets
• void pre_restore ()
  restore a previous state of the shared approximation data
• void post_restore ()
  clean up saved storage following restoration
• size_t finalization_index (size_t i)
  return index of i-th trailing trial set within restorable bookkeeping sets
• void pre_finalize ()
  finalize the shared approximation data following a set of increments
• void post_finalize ()
  clean up saved storage following aggregation
• void store ()
  store the current state of the shared approximation data for later combination
• void pre_combine (short corr_type)
  aggregate the shared approximation data from current and saved states
• void post_combine (short corr_type)
  clean up saved storage after aggregation

Private Member Functions

• Pecos::SharedBasisApproxData & pecos_shared_data ()
  return pecosSharedData
• void approx_type_to_basis_type (const String &approx_type, short &basis_type)
  utility to convert Dakota type string to Pecos type enumeration
Private Attributes

- Pecos::SharedBasisApproxData pecosSharedData
  the Pecos shared approximation data
- Pecos::SharedPolyApproxData * pecosSharedDataRep
  convenience pointer to derived letter within pecosSharedData

Friends

- class PecosApproximation

Additional Inherited Members

13.146.1 Detailed Description

Derived approximation class for global basis polynomials. The SharedPecosApproxData 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.

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

- SharedPecosApproxData.hpp
- SharedPecosApproxData.cpp

13.147 SharedSurfpackApproxData Class Reference

Derived approximation class for Surfpack approximation classes. Interface between Surfpack and Dakota. Inheritance diagram for SharedSurfpackApproxData:

```
SharedApproxData
 SharedSurfpackApproxData
```

Public Member Functions

- SharedSurfpackApproxData ()
  default constructor
- SharedSurfpackApproxData (const String &approx_type, const UShortArray &approx_order, size_t num_vars, short data_order, short output_level)
  alternate constructor
- SharedSurfpackApproxData (ProblemDescDB &problem_db, size_t num_vars)
  standard constructor: Surfpack surface of appropriate type will be created
- ~SharedSurfpackApproxData ()
  destructor
Private Member Functions

- void add_sd_to_surfdata (const Pecos::SurrogateDataVars &sdv, const Pecos::SurrogateDataResp &sdr, short fail_code, SurfData &surf_data)
  
  *add Pecos::SurrogateData::SurrogateData{Vars,Resp} to SurfData, accounting for buildDataOrder available*

- void copy_matrix (const RealSymMatrix &rsm, SurfpackMatrix<Real> &surfpack_matrix)
  
  *copy RealSymMatrix to SurfpackMatrix (Real type only)*

- void merge_variable_arrays (const RealVector &cv, const IntVector &div, const RealVector &drv, RealArray &ra)
  
  *merge cv, div, and drv vectors into a single ra array*

- void sdv_to_realarray (const Pecos::SurrogateDataVars &sdv, RealArray &ra)
  
  *aggregate {continuous,discrete int,discrete real} variables from SurrogateDataVars into ra*

- void vars_to_realarray (const Variables &vars, RealArray &ra)
  
  *aggregate {active,all} {continuous,discrete int,discrete real} variables into ra*

Private Attributes

- unsigned short approxOrder
  
  *order of polynomial approximation*

- String exportModelName
  
  *A Surfpack model name for saving the surrogate model.*

- StringArray diagnosticSet
  
  *set of diagnostic metrics*

- bool crossValidateFlag
  
  *whether to perform cross validation*

- unsigned numFolds
  
  *number of folds for CV*

- Real percentFold
  
  *percentage of data for CV*

- bool pressFlag
  
  *whether to perform PRESS*

Friends

- class SurfpackApproximation

Additional Inherited Members

13.147.1 Detailed Description

Derived approximation class for Surfpack approximation classes. Interface between Surfpack and Dakota.

The SharedSurfpackApproxData class is the interface between Dakota and Surfpack. Based on the information in the ProblemDescDB that is passed in through the constructor, SharedSurfpackApproxData 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 (M-ARS).
13.147.2 Constructor & Destructor Documentation

SharedSurfpackApproxData ( const String & approx_type, const UShortArray & approx_order, size_t num_vars, short data_order, short output_level )

alternate constructor

On-the-fly constructor which uses mostly Surfpack model defaults.

References Dakota::abort_handler(), SharedSurfpackApproxData::approxOrder, and SharedApproxData::approxType.

SharedSurfpackApproxData ( ProblemDescDB & problem_db, size_t num_vars )

standard constructor: Surfpack surface of appropriate type will be created

Initialize the embedded Surfpack surface object and configure it using the specifications from the input file. Data for the surface is created later.

References SharedSurfpackApproxData::approxOrder, SharedApproxData::approxType, ProblemDescDB::get_short(), and ProblemDescDB::get_string().

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

- SharedSurfpackApproxData.hpp
- SharedSurfpackApproxData.cpp

13.148 SharedVariablesData Class Reference

Container class encapsulating variables data that can be shared among a set of Variables instances.

Public Member Functions

- SharedVariablesData ()
  default constructor
- SharedVariablesData (const ProblemDescDB & problem_db, const std::pair< short, short > & view)
  standard constructor
- SharedVariablesData (const std::pair< short, short > & view, const SizetArray & vars_comps_totals)
  lightweight constructor
- SharedVariablesData (const SharedVariablesData & svd)
  copy constructor
- ~SharedVariablesData ()
  destructor
- SharedVariablesData & operator= (const SharedVariablesData & svd)
  assignment operator
- SharedVariablesData copy () const
  create a deep copy of the current object and return by value
- void size_all_continuous_labels (bool relax)
  size labels for all of the continuous variables, with or without discrete relaxation
- void initialize_all_continuous_types (bool relax)
  initialize types for all of the continuous variables, with or without discrete relaxation
- void initialize_all_continuous_ids (bool relax)
initialize ids for all of the continuous variables, with or without discrete relaxation

- void `size_all_discrete_int_labels()`
  size labels for all of the discrete integer variables

- void `initialize_all_discrete_int_types()`
  initialize types for all of the discrete integer variables

- void `size_all_discrete_real_labels()`
  size labels for all of the discrete real variables

- void `initialize_all_discrete_real_types()`
  initialize types for all of the discrete real variables

- void `initialize_active_components()`
  initialize the active components totals given active variable counts

- void `initialize_inactive_components()`
  initialize the inactive components totals given inactive variable counts

- `StringMultiArrayView all_continuous_labels(size_t start, size_t num_items)` const
  get `num_items` continuous labels beginning at index `start`

- void `all_continuous_labels(StringMultiArrayConstView cv_labels, size_t start, size_t num_items)`
  set `num_items` continuous labels beginning at index `start`

- void `all_continuous_label(const String &cv_label, size_t index)`
  set continuous label at index `start`

- `StringMultiArrayView all_discrete_int_labels(size_t start, size_t num_items)` const
  get `num_items` discrete integer labels beginning at index `start`

- void `all_discrete_int_labels(StringMultiArrayConstView div_labels, size_t start, size_t num_items)`
  set `num_items` discrete integer labels beginning at index `start`

- void `all_discrete_int_label(const String &div_label, size_t index)`
  set discrete integer label at index `start`

- `StringMultiArrayView all_discrete_real_labels(size_t start, size_t num_items)` const
  get `num_items` discrete real labels beginning at index `start`

- void `all_discrete_real_labels(StringMultiArrayConstView drv_labels, size_t start, size_t num_items)`
  set `num_items` discrete real labels beginning at index `start`

- void `all_discrete_real_label(const String &drv_label, size_t index)`
  set discrete real label at index `start`

- `UShortMultiArrayConstView all_continuous_types(size_t start, size_t num_items)` const
  get `num_items` continuous types beginning at index `start`

- void `all_continuous_types(UShortMultiArrayConstView cv_types, size_t start, size_t num_items)`
  set `num_items` continuous types beginning at index `start`

- void `all_continuous_type(unsigned short cv_type, size_t index)`
  set continuous type at index `start`

- `UShortMultiArrayConstView all_discrete_int_types(size_t start, size_t num_items)` const
  get `num_items` discrete integer types beginning at index `start`

- void `all_discrete_int_types(UShortMultiArrayConstView div_types, size_t start, size_t num_items)`
  set `num_items` discrete integer types beginning at index `start`

- void `all_discrete_int_type(unsigned short div_type, size_t index)`
  set discrete integer type at index `start`
CHAPTER 13. CLASS DOCUMENTATION

- UShortMultiArrayConstView all_discrete_real_types (size_t start, size_t num_items) const
  get num_items discrete real types beginning at index start
- void all_discrete_real_types (UShortMultiArrayConstView drv_types, size_t start, size_t num_items)
  set num_items discrete real types beginning at index start
- void all_discrete_real_type (unsigned short drv_type, size_t index)
  set discrete real type at index
- SizetMultiArrayConstView all_continuous_ids (size_t start, size_t num_items) const
  get num_items continuous ids beginning at index start
- void all_continuous_ids (SizetMultiArrayConstView cv_ids, size_t start, size_t num_items)
  set num_items continuous ids beginning at index start
- void all_continuous_id (size_t id, size_t index)
  set num_items continuous ids beginning at index start
- const SizetArray & relaxed_discrete_ids () const
  get ids of discrete variables that have been relaxed into continuous variable arrays
- const String & id () const
  return the user-provided or default Variables identifier
- const SizetArray & components_totals () const
  return variable type counts for continuous,discrete integer,discrete real
  design,aleatory uncertain,epistemic uncertain,state
- const SizetArray & active_components_totals () const
  return active variable type counts for continuous,discrete integer,discrete real
  design,aleatory uncertain,epistemic uncertain,state
- const SizetArray & inactive_components_totals () const
  return inactive variable type counts for continuous,discrete integer,discrete real
  design,aleatory uncertain,epistemic uncertain,state
- size_t vc_lookup (unsigned short key) const
  retrieve the variables type count within svdRep->variablesComponents corresponding to (a fine-grain variables type) key
- const std::pair< short, short > & view () const
  retrieve the Variables view
- void inactive_view (short view2)
  set the inactive Variables view
- size_t cv () const
  get number of active continuous vars
- size_t cv_start () const
  get start index of active continuous vars
- size_t div () const
  get number of active discrete int vars
- size_t div_start () const
  get start index of active discrete int vars
- size_t drv () const
  get number of active discrete real vars
- size_t drv_start () const
  get start index of active discrete real vars
- size_t icv() const
  get number of inactive continuous vars
- size_t icv_start() const
  get start index of inactive continuous vars
- size_t idiv() const
  get number of inactive discrete int vars
- size_t idiv_start() const
  get start index of inactive discrete int vars
- size_t idrv() const
  get number of inactive discrete real vars
- size_t idrv_start() const
  get start index of inactive discrete real vars
- void cv(size_t ncv)
  set number of active continuous vars
- void cv_start(size_t cvs)
  set start index of active continuous vars
- void div(size_t ndiv)
  set number of active discrete int vars
- void div_start(size_t divs)
  set start index of active discrete int vars
- void drv(size_t ndrv)
  set number of active discrete real vars
- void drv_start(size_t drvs)
  set start index of active discrete real vars
- void icv(size_t ncv)
  set number of inactive continuous vars
- void icv_start(size_t icvs)
  set start index of inactive continuous vars
- void idiv(size_t nidiv)
  set number of inactive discrete int vars
- void idiv_start(size_t idivs)
  set start index of inactive discrete int vars
- void idrv(size_t nidrv)
  set number of inactive discrete real vars
- void idrv_start(size_t idrvs)
  set start index of inactive discrete real vars

Private Attributes

- SharedVariablesDataRep * svdRep
  pointer to the body (handle-body idiom)
13.148.1 Detailed Description

Container class encapsulating variables data that can be shared among a set of Variables instances.

An array of Variables objects (e.g., Analyzer::allVariables) contains repeated configuration data (id’s, labels, counts). SharedVariablesData employs a handle-body idiom to allow this shared data to be managed in a single object with many references to it, one per Variables object in the array. This allows scaling to larger sample sets.

13.148.2 Member Function Documentation

SharedVariablesData copy ( ) const

create a deep copy of the current object and return by value

Deep copies are used when recasting changes the nature of a Variables set.

References SharedVariablesDataRep::activeVarsCompsTotals, SharedVariablesDataRep::allContinuousIds, SharedVariablesDataRep::allContinuousLabels, SharedVariablesDataRep::allContinuousTypes, SharedVariablesDataRep::allDiscreteIntLabels, SharedVariablesDataRep::allDiscreteIntTypes, SharedVariablesDataRep::allDiscreteRealLabels, SharedVariablesDataRep::allDiscreteRealTypes, SharedVariablesDataRep::cvStart, SharedVariablesDataRep::divStart, SharedVariablesDataRep::drvStart, SharedVariablesDataRep::icvStart, SharedVariablesDataRep::idivStart, SharedVariablesDataRep::idrvStart, SharedVariablesDataRep::inactiveVarsCompsTotals, SharedVariablesDataRep::numCV, SharedVariablesDataRep::numDIV, SharedVariablesDataRep::numDRV, SharedVariablesDataRep::numICV, SharedVariablesDataRep::numIDIV, SharedVariablesDataRep::numIDRV, SharedVariablesDataRep::relaxedDiscreteIds, SharedVariablesData::svdRep, SharedVariablesDataRep::variablesComponents, SharedVariablesDataRep::variablesCompsTotals, SharedVariablesDataRep::variablesId, and SharedVariablesDataRep::variablesView.

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

- SharedVariablesData.hpp
- SharedVariablesData.cpp

13.149 SharedVariablesDataRep Class Reference

The representation of a SharedVariablesData instance. This representation, or body, may be shared by multiple SharedVariablesData handle instances.

Private Member Functions

- SharedVariablesDataRep (const ProblemDescDB &problem_db, const std::pair< short, short > &view)
  standard constructor
- SharedVariablesDataRep (const std::pair< short, short > &view, const std::map< unsigned short, size_t > &vars_comps)
  medium weight constructor
- SharedVariablesDataRep (const std::pair< short, short > &view, const SizetArray &vars_comps_totals)
  lightweight constructor
- SharedVariablesDataRep ()
  default constructor
- ~SharedVariablesDataRep ()
  destructor
- void components_to_totals ()
void do_all_continuous_labels (bool relax)
size allContinuousLabels, with or without discrete relaxation
void do_all_continuous_types (bool relax)
initialize allContinuousTypes, with or without discrete relaxation
void do_all_continuous_ids (bool relax)
initialize allContinuousIds, with or without discrete relaxation

void do_all_discrete_int_labels ()
size allDiscreteIntLabels
void do_all_discrete_int_types ()
initialize allDiscreteIntTypes
void do_all_discrete_real_labels ()
size allDiscreteRealLabels
void do_all_discrete_real_types ()
initialize allDiscreteRealTypes

void do_active_components ()
initialize activeVarsCompsTotals given {c,di,dr}\{vStart and num\{C,DI,DR\}\}
void do_inactive_components ()
initialize inactiveVarsCompsTotals given {c,di,dr}\{vStart and num\{C,DI,DR\}\}

size\_t vc\_lookup (unsigned short key) const
retrieve the count within variablesComponents corresponding to key

Private Attributes

- String variablesId
  variables identifier string from the input file
- std::map\< unsigned short, size\_t \> \ variablesComponents
  map linking variable types to counts
- SizetArray variablesCompsTotals
  totals for variable type counts for \{continuous,discrete integer,discrete real\} \{design,aleatory uncertain,epistemic uncertain,state\}
- SizetArray activeVarsCompsTotals
  totals for active variable type counts for \{continuous,discrete integer,discrete real\} \{design,aleatory uncertain,epistemic uncertain,state\}
- SizetArray inactiveVarsCompsTotals
  totals for inactive variable type counts for \{continuous,discrete integer,discrete real\} \{design,aleatory uncertain,epistemic uncertain,state\}
- std::pair\< short, short \> \ variablesView
  the variables view pair containing active (first) and inactive (second) view enumerations
- size\_t cv\_Start
  start index of active continuous variables within allContinuousVars
- size\_t div\_Start
  start index of active discrete integer variables within allDiscreteIntVars
CHAPTER 13. CLASS DOCUMENTATION

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

• StringMultiArray allContinuousLabels
  array of variable labels for all of the continuous variables

• StringMultiArray allDiscreteIntLabels
  array of variable labels for all of the discrete integer variables

• StringMultiArray allDiscreteRealLabels
  array of variable labels for all of the discrete real variables

• UShortMultiArray allContinuousTypes
  array of variable types for all of the continuous variables

• UShortMultiArray allDiscreteIntTypes
  array of variable types for all of the discrete integer variables

• UShortMultiArray allDiscreteRealTypes
  array of variable types for all of the discrete real variables

• SizetMultiArray allContinuousIds
  array of 1-based position identifiers for the all continuous variables array

• SizetArray relaxedDiscreteIds
  array of discrete variable identifiers for which the discrete requirement is relaxed by merging them into a continuous array

• int referenceCount
  number of handle objects sharing svdRep

Friends

• class SharedVariablesData
13.149.1 Detailed Description

The representation of a SharedVariablesData instance. This representation, or body, may be shared by multiple SharedVariablesData handle instances.

The SharedVariablesData/SharedVariablesDataRep pairs utilize a handle-body idiom (Coplien, Advanced C++).

13.149.2 Constructor & Destructor Documentation

SharedVariablesDataRep (const ProblemDescDB &problem_db, const std::pair<short, short> &view) [private]

standard constructor

This constructor is the one which must build the base class data for all derived classes. get_variables() instantiates a derived class 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).

References SharedVariablesDataRep::allContinuousLabels, SharedVariablesDataRep::allDiscreteIntLabels, SharedVariablesDataRep::allDiscreteRealLabels, Dakota::copy_data_partial(), ProblemDescDB::get_sa(), ProblemDescDB::get_sizet(), SharedVariablesDataRep::initialize_all_continuous_ids(), SharedVariablesDataRep::initialize_all_continuous_types(), SharedVariablesDataRep::initialize_all_discrete_int_types(), SharedVariablesDataRep::initialize_all_discrete_real_types(), SharedVariablesDataRep::variablesComponents, SharedVariablesDataRep::variablesCompsTotals, and SharedVariablesDataRep::variablesView.

13.149.3 Member Data Documentation

SizetMultiArray allContinuousIds [private]

array of 1-based position identifiers for the all continuous variables array

These identifiers define positions of the all continuous variables array within the total variable sequence.

Referenced by SharedVariablesData::all_continuous_id(), SharedVariablesData::all_continuous_ids(), SharedVariablesData::copy(), and SharedVariablesDataRep::initialize_all_continuous_ids().

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

- SharedVariablesData.hpp
- SharedVariablesData.cpp

13.150 SingleModel Class Reference

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

Inheritance diagram for SingleModel:

```
   Model
      ▼
       SingleModel
```
Public Member Functions

- **SingleModel** *(ProblemDescDB &problem_db)*
  constructor
- **~SingleModel** ()
  destructor

Protected Member Functions

- **Interface & derived_interface ()**
  return userDefinedInterface
- void **derived_compute_response** (const ActiveSet &set)
  portion of compute_response() specific to SingleModel (invokes a synchronous map() on userDefinedInterface)
- void **derived_async_compute_response** (const ActiveSet &set)
  portion of async_compute_response() specific to SingleModel (invokes an asynchronous map() on userDefinedInterface)
- const IntResponseMap & **derived_synchronize ()**
  portion of synchronize() specific to SingleModel (invokes synch() on userDefinedInterface)
- const IntResponseMap & **derived_synchronize_nowait ()**
  portion of synchronize_nowait() specific to SingleModel (invokes synch_nowait() on userDefinedInterface)
- void **component_parallel_mode** (short mode)
  SingleModel only supports parallelism in userDefinedInterface, so this virtual function redefinition is simply a sanity check.
- String **local_eval_synchronization ()**
  return userDefinedInterface synchronization setting
- int **local_eval_concurrency ()**
  return userDefinedInterface asynchronous evaluation concurrency
- bool **derived_master_overload ()** const
  flag which prevents overloading the master with a multiprocessor evaluation (request forwarded to userDefinedInterface)
- void **derived_init_communicators** (int max_eval_concurrency, bool recurse_flag=true)
  set up SingleModel for parallel operations (request forwarded to userDefinedInterface)
- void **derived_init_serial ()**
  set up SingleModel for serial operations (request forwarded to userDefinedInterface).
- void **derived_set_communicators** (int max_eval_concurrency, bool recurse_flag=true)
  set active parallel configuration for the SingleModel (request forwarded to userDefinedInterface)
- void **derived_free_communicators** (int max_eval_concurrency, bool recurse_flag=true)
  deallocate communicator partitions for the SingleModel (request forwarded to userDefinedInterface)
- void **serve** (int max_eval_concurrency)
  Service userDefinedInterface job requests received from the master. Completes when a termination message is received from stop_servers().
- void **stop_servers ()**
  executed by the master to terminate userDefinedInterface server operations when SingleModel iteration is complete.
- const String & **interface_id ()** const
  return the userDefinedInterface identifier
13.151. **SNLLBase Class Reference**

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

Inheritance diagram for SNLLBase:
Public Member Functions

- **SNLLBase ()**
  - default constructor

- **SNLLBase (ProblemDescDB &problem db)**
  - standard constructor

- **~SNLLBase ()**
  - destructor

Protected Member Functions

- **void copy_con_vals_dak_to_optpp** (const RealVector &local_fn_vals, RealVector &g, size_t offset)
  - convenience function for copying local_fn_vals to g; used by constraint evaluator functions

- **void copy_con_vals_optpp_to_dak** (const RealVector &g, RealVector &local_fn_vals, size_t offset)
  - convenience function for copying g to local_fn_vals; used in final solution logging

- **void copy_con_grad** (const RealMatrix &local_fn_grads, RealMatrix &grad_g, size_t offset)
  - convenience function for copying local_fn_grads to grad_g; used by constraint evaluator functions

- **void copy_con_hess** (const RealSymMatrixArray &local_fn_hessians, OPTPP::OptppArray<RealSymMatrix> &hess_g, size_t offset)
  - convenience function for copying local_fn_hessians to hess_g; used by constraint evaluator functions

- **void snll_pre_instantiate** (bool bound_constr_flag, int num_constr)
  - convenience function for setting OPT++ options prior to the method instantiation

- **void snll_post_instantiate** (int num_cv, bool vendor_num_grad_flag, const String &finite_diff_type, const RealVector &fdss, int max_iter, int max_fn_evals, Real conv_tol, Real grad_tol, Real max_step, bool bound_constr_flag, int num_constr, short output Lev, OPTPP::OptimizeClass &the_optimizer, OPTPP::NLP0 *nlf_objective, OPTPP::FDNLF1 *fd_nlf1, OPTPP::FDNLF1 *fd_nlf1_con)
  - convenience function for setting OPT++ options after the method instantiation

- **void snll_initialize_run** (OPTPP::NLP0 *nlf_objective, OPTPP::NLP *nlp_constraint, const RealVector &init_pt, bool bound_constr_flag, const RealVector &lower_bnds, const RealVector &upper_bnds, const RealMatrix &lin_ineq_coeffs, const RealVector &lin_ineq_l_bnds, const RealVector &lin_ineq_u_bnds, const RealMatrix &lin_eq_coeffs, const RealVector &lin_eq_targets, const RealVector &nlm_ineq_l_bnds, const RealVector &nlm_ineq_u_bnds, const RealVector &nlm_eq_targets)
  - convenience function for OPT++ configuration prior to the method invocation

- **void snll_post_run** (OPTPP::NLP0 *nlf_objective)
  - convenience function for setting OPT++ options after the method instantiations

Static Protected Member Functions

- **static void init_fn** (int n, RealVector &x)
  - An initialization mechanism provided by OPT++ (not currently used).
Protected Attributes

- String searchMethod
  value based line_search, gradient based line_search, tr
- OPTPP::SearchStrategy searchStrat
  enum: LineSearch, TrustRegion, TrustPDS
- OPTPP::MeritFcn meritFn
  enum: NormFmu, ArgaezTapia, VanShanno
- Real maxStep
  value from max_step specification
- Real stepLenToBndry
  value from steplength_to_boundary specification
- Real centeringParam
  value from centering_parameter specification
- bool constantASVFlag
  flags a user selection of active_set_vector == constant. By mapping this into mode override, reliance on duplicate detection can be avoided.

Static Protected Attributes

- static Minimizer * optLSqInstance
  pointer to the active base class object instance used within the static evaluator functions in order to avoid the need for static data
- static bool modeOverrideFlag
  flags OPT++ mode override (for combining value, gradient, and Hessian requests)
- static EvalType lastFnEvalLocn
  an enum used to track whether an nlf evaluator or a constraint evaluator was the last location of a function evaluation
- static int lastEvalMode
  copy of mode from constraint evaluators
- static RealVector lastEvalVars
  copy of variables from constraint evaluators

13.151.1 Detailed Description

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

The SNLLBase class provides a common base class for SNLLOptimizer and SNLLLeastSq, both of which are wrappers for OPT++, a C++ optimization library from the Computational Sciences and Mathematics Research (CSMR) department at Sandia’s Livermore CA site.

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

- SNLLBase.hpp
- SNLLBase.cpp
13.152 SNLLLeastSq Class Reference

Wrapper class for the OPT++ optimization library.

Inheritance diagram for SNLLLeastSq:

```
SNLLLeastSq
  |     |
  v     v
LeastSq SNLLBase
  |     |
  v     v
Minimizer
  |
Iterator
```

Public Member Functions

- `SNLLLeastSq (ProblemDescDB &problem_db, Model &model)`
  standard constructor
- `SNLLLeastSq (const String &method_name, Model &model)`
  alternate constructor for instantiations without ProblemDescDB support
- `~SNLLLeastSq ()`
  destructor
- `void minimize_residuals ()`
  Performs the iterations to determine the least squares solution.

Protected Member Functions

- `void initialize_run ()`
  invokes LeastSq::initialize_run(), SNLLBase::snll_initialize_run(), and performs other set-up
- `void post_run (std::ostream &s)`
  invokes snll_post_run and re-implements post_run (does not call parent) and performs other solution processing
- `void finalize_run ()`
  restores instances

Static Private Member Functions

- `static void nlf2_evaluator_gn (int mode, int n, const RealVector &x, double &f, RealVector &grad_f, RealSymMatrix &hess_f, int &result_mode)`
  objective function evaluator function which obtains values and gradients for least square terms and computes objective function value, gradient, and Hessian using the Gauss-Newton approximation.
- `static void constraint1_evaluator_gn (int mode, int n, const RealVector &x, RealVector &g, RealMatrix &grad_g, int &result_mode)`
  constraint evaluator function which provides constraint values and gradients to OPT++ Gauss-Newton methods.
- `static void constraint2_evaluator_gn (int mode, int n, const RealVector &x, RealVector &g, RealMatrix &grad_g, OPTPP::OptppArray<RealSymMatrix> &hess_g, int &result_mode)`
constraint evaluator function which provides constraint values, gradients, and Hessians to OPT++ Gauss-Newton methods.

Private Attributes

- **SNLLLeastSq * prevSnllSqInstance**
  pointer to the previously active object instance used for restoration in the case of iterator/model recursion
- **OPTPP::NLP0 * nlfObjective**
  objective NLF base class pointer
- **OPTPP::NLP0 * nlfConstraint**
  constraint NLF base class pointer
- **OPTPP::NLP * nlpConstraint**
  constraint NLP pointer
- **OPTPP::NLF2 * nlf2**
  pointer to objective NLF for full Newton optimizers
- **OPTPP::NLF2 * nlf2Con**
  pointer to constraint NLF for full Newton optimizers
- **OPTPP::NLF1 * nlf1Con**
  pointer to constraint NLF for Quasi Newton optimizers
- **OPTPP::OptimizeClass * theOptimizer**
  optimizer base class pointer
- **OPTPP::OptNewton * optnewton**
  Newton optimizer pointer.
- **OPTPP::OptBCNewton * optbcnewton**
  Bound constrained Newton optimizer ptr.
- **OPTPP::OptDHNIPS * optdhnips**
  Disaggregated Hessian NIPS optimizer ptr.

Static Private Attributes

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

Additional Inherited Members

13.152.1 Detailed Description

Wrapper class for the OPT++ optimization library.

The **SNLLLeastSq** class provides a wrapper for OPT++, a C++ optimization library of nonlinear programming and pattern search techniques from the Computational Sciences and Mathematics Research (CSMR) department at Sandia’s Livermore CA site. It uses a function pointer approach for which passed functions must be either global functions or static member functions. Any attribute used within static member functions must be either local to that function, a static member, or accessed by static pointer.

The user input mappings are as follows: max_iterations, max_function_evaluations, convergence_tolerance, max_step, gradient_tolerance, search_method, and search_scheme_size are
set using OPT++’s setMaxIter(), setMaxFeval(), setFcnTol(), setMaxStep(), setGradTol(), setSearchStrategy(), and setSSS() member functions, respectively; output verbosity is used to toggle OPT++’s debug mode using the setDebug() member function. Internal to OPT++, there are 3 search strategies, while the DAKOTA search method specification supports 4 (value_based_line_search, gradient_based_line_search, trust_region, or tr_pds). The difference stems from the “is_expensive” flag in OPT++. If the search strategy is LineSearch and “is_expensive” is turned on, then the value_based_line_search is used. Otherwise (the “is_expensive” default is off), the algorithm will use the gradient_based_line_search. Refer to [Meza, J.C., 1994] and to the OPT++ source in the Dakota/packages/OPTPP directory for information on OPT++ class member functions.

13.152.2 Member Function Documentation

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

invokes snll_post_run and re-implements post_run (does not call parent) and performs other solution processing

SNLLLeastSq requires fn DB lookup, so overrides LeastSq::post_run and directly invokes Iterator::post_run when complete.

Reimplemented from LeastSq.

References Iterator::activeSet, Iterator::bestResponseArray, Iterator::bestVariablesArray, SNLLBase::copy_con_vals_oppp_to_dak(), Dakota::copy_data_partial(), Minimizer::cvScaleMultipliers, Minimizer::cvScaleOffsets, Minimizer::cvScaleTypes, Dakota::data_pairs, Minimizer::expData, LeastSq::get_confidence_intervals(), Model::interface_id(), Iterator::iteratedModel, Dakota::lookup_by_val(), Minimizer::modify_s2n(), Minimizer::need_resp_trans_byvars(), SNLLLeastSq::nlfObjective, Minimizer::numExperiments, Minimizer::numFunctions, LeastSq::numLeastSqTerms, Minimizer::numNonlinearConstraints, Minimizer::numReplicates, Minimizer::numUserPrimaryFns, Minimizer::obsDataFlag, Minimizer::post_run(), ActiveSet::request_values(), ActiveSet::request_vector(), Minimizer::responseScaleMultipliers, Minimizer::responseScaleOffsets, Minimizer::responseScaleTypes, ExperimentData::scalar_data(), Minimizer::secondaryRespScaleFlag, SNLLBase::snll_post_run(), SNLLLeastSq::theOptimizer, and Minimizer::varsScaleFlag.

void nlf2_evaluator_gn ( int mode, int n, const RealVector & x, double & f, RealVector & grad_f, RealSymMatrix & hess_f, int & result_mode ) [static], [private]

objective function evaluator function which obtains values and gradients for least square terms and computes objective function value, gradient, and Hessian using the Gauss-Newton approximation.

This nlf2 evaluator function is used for the Gauss-Newton method in order to exploit the special structure of the nonlinear least squares problem. Here, \( \text{fx} = \sum (T_i - T_{\text{bar}})^2 \) and Response is made up of residual functions and their gradients along with any nonlinear constraints. The objective function and its gradient vector and Hessian matrix are computed directly from the residual functions and their derivatives (which are returned from the Response object).

References Dakota::abort_handler(), Iterator::activeSet, Model::compute_response(), Model::continuous_variables(), Model::current_response(), Response::function_gradients(), Response::function_values(), Iterator::iteratedModel, SNLLBase::lastEvalMode, SNLLBase::lastEvalVars, SNLLBase::lastFnEvalLocn, Minimizer::numFunctions, LeastSq::numLeastSqTerms, Minimizer::numNonlinearConstraints, Iterator::outputLevel, ActiveSet::request_vector(), SNLLLeastSq::snllLSqInstance, and Dakota::write_precision.

Referenced by SNLLLeastSq::SNLLLeastSq().

void constraint1_evaluator_gn ( int mode, int n, const RealVector & x, RealVector & g, RealMatrix & grad_g, int & result_mode ) [static], [private]

cost constraint evaluator function which provides constraint values and gradients to OPT++ Gauss-Newton methods.
While it does not employ the Gauss-Newton approximation, it is distinct from constraint1::evaluator() due to its need to anticipate the required modes for the least squares terms. This constraint evaluator function is used with diaggregated Hessian NIPS and is currently active.

References Dakota::abort_handler(), Iterator::activeSet, Model::compute_response(), Model::continuous_variables(), SNLLBase::copy_con_grad(), SNLLBase::copy_con_vals_dak_to_optpp(), Model::current_response(), Response::function_gradients(), Response::function_values(), Iterator::iteratedModel, SNLLBase::lastEvalMode, SNLLBase::lastEvalVars, SNLLBase::lastFnEvalLocn, Minimizer::numFunctions, LeastSq::numLeastSqTerms, Iterator::outputLevel, ActiveSet::request_vector(), and SNLLLeastSq::snllLSqInstance.

Referenced by SNLLLeastSq::SNLLLeastSq().

```cpp
void constraint2::evaluator_gn ( int mode, int n, const RealVector &x, RealVector &g, RealMatrix &grad_g, OPTPP::OptppArray<RealSymMatrix> &hess_g, int &result_mode ) [static], [private]
```

constraint evaluator function which provides constraint values, gradients, and Hessians to OPT++ Gauss-Newton methods.

While it does not employ the Gauss-Newton approximation, it is distinct from constraint2::evaluator() due to its need to anticipate the required modes for the least squares terms. This constraint evaluator function is used with full Newton NIPS and is currently inactive.

References Dakota::abort_handler(), Iterator::activeSet, Model::compute_response(), Model::continuous_variables(), SNLLBase::copy_con_grad(), SNLLBase::copy_con_hess(), SNLLBase::copy_con_vals_dak_to_optpp(), Model::current_response(), Response::function_gradients(), Response::function_hessians(), Response::function_values(), Iterator::iteratedModel, SNLLBase::lastEvalMode, SNLLBase::lastEvalVars, SNLLBase::lastFnEvalLocn, SNLLBase::modeOverrideFlag, Minimizer::numFunctions, LeastSq::numLeastSqTerms, Iterator::outputLevel, ActiveSet::request_vector(), and SNLLLeastSq::snllLSqInstance.

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

- SNLLLeastSq.hpp
- SNLLLeastSq.cpp

### 13.153 SNLLOptimizer Class Reference

Wrapper class for the OPT++ optimization library.

Inheritance diagram for SNLLOptimizer:

```
   +----------------------------------------+       +-----------------+
   |        Iterator                       |       |     SNLLBase    |
   |                     |       |                |       |               |
   +-------------------> Minimizer          +-------+-----------------+-------+
         |                                |       |                |       |                |
         |                                |-------+-----------------+-------+                |
         v                                |               |               |       |                |
   +----------------------------------------+       +-----------------+       +-----------------+
   |         Optimizer                  |       |     SNLLOptimizer|
   v                                    +-------+-----------------+-----------------+
SNLLOptimizer                           |       |                |
```

Public Member Functions

- **SNLLOptimizer** (ProblemDescDB &problem_db, Model &model)
standard constructor

- **SNLLOptimizer** (const String &method_string, 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 RealMatrix &lin_eq_coeffs, const RealVector &lin_eq_tgts, const RealVector &nln_ineq_l_bnds, const RealVector &nln_ineq_u_bnds, const RealVector &nln_eq_tgts, void(*user_obj_eval)(int mode, int n, const RealVector &x, double &f, RealVector &grad_f, int &result_mode), void(*user_con_eval)(int mode, int n, const RealVector &x, RealVector &g, RealMatrix &grad_g, int &result_mode))

alternate constructor for instantiations "on the fly"

- ∼SNLLOptimizer ()

destructor

- void **find_optimum** ()

  Performs the iterations to determine the optimal solution.

Protected Member Functions

- void **initialize_run** ()

  invokes Optimizer::initialize_run(), SNLLBase::snll_initialize_run(), and performs other set-up

- void **post_run** (std::ostream &s)

  performs data recovery and calls Optimizer::post_run()

- void **finalize_run** ()

  performs cleanup, restores instances and calls parent finalize

Static Private Member Functions

- static void **nlf0_evaluator** (int n, const RealVector &x, double &f, int &result_mode)

  objective function evaluator function for OPT++ methods which require only function values.

- static void **nlf1_evaluator** (int mode, int n, const RealVector &x, double &f, RealVector &grad_f, int &result_mode)

  objective function evaluator function which provides function values and gradients to OPT++ methods.

- static void **nlf2_evaluator** (int mode, int n, const RealVector &x, double &f, RealVector &grad_f, RealSymMatrix &hess_f, int &result_mode)

  objective function evaluator function which provides function values, gradients, and Hessians to OPT++ methods.

- static void **constraint0_evaluator** (int n, const RealVector &x, RealVector &g, int &result_mode)

  constraint evaluator function for OPT++ methods which require only constraint values.

- static void **constraint1_evaluator** (int mode, int n, const RealVector &x, RealVector &g, RealMatrix &grad_g, int &result_mode)

  constraint evaluator function which provides constraint values and gradients to OPT++ methods.

- static void **constraint2_evaluator** (int mode, int n, const RealVector &x, RealVector &g, RealMatrix &grad_g, OPTPP::OptppArray<RealSymMatrix> &hess_g, int &result_mode)

  constraint evaluator function which provides constraint values, gradients, and Hessians to OPT++ methods.
Private Attributes

- `SNLLOptimizer * prevSnllOptInstance`  
  pointer to the previously active object instance used for restoration in the case of iterator/model recursion
- `OPTPP::NLP0 * nlfObjective`  
  objective NLF base class pointer
- `OPTPP::NLP0 * nlfConstraint`  
  constraint NLF base class pointer
- `OPTPP::NLP * nlpConstraint`  
  constraint NLP pointer
- `OPTPP::NLF0 * nl0`  
  pointer to objective NLF for nongradient optimizers
- `OPTPP::NLF1 * nl1`  
  pointer to objective NLF for (analytic) gradient-based optimizers
- `OPTPP::NLF1 * nl1Con`  
  pointer to constraint NLF for (analytic) gradient-based optimizers
- `OPTPP::FDNLF1 * fdlf1`  
  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 * nl2`  
  pointer to objective NLF for full Newton optimizers
- `OPTPP::NLF2 * nl2Con`  
  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`
816  CHAPTER 13. CLASS DOCUMENTATION

* Bnd constrained FD-Newton opt ptr.  
  * OPTPP::OptNIPS * optnips  
    NIPS optimizer pointer.  
  * OPTPP::OptQNIPS * optqnips  
    Quasi-Newton NIPS optimizer pointer.  
  * OPTPP::OptFDNIPS * optfdnips  
    Finite Difference NIPS opt pointer.  
  * String setUpType  
    flag for iteration mode: "model" (normal usage) or "user functions" (user-supplied functions mode for "on the fly" instantiations). NonDReliability currently uses the user functions mode.  
  * RealVector initialPoint  
    holds initial point passed in for "user functions" mode.  
  * RealVector lowerBounds  
    holds variable lower bounds passed in for "user functions" mode.  
  * RealVector upperBounds  
    holds variable upper bounds passed in for "user functions" mode.  

Static Private Attributes

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

Additional Inherited Members

13.153.1 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(), setMaxEval(), setFctol(), setMaxStep(), setGradTol(), setSearchStrategy(), and setSSS() member functions, respectively; output verbosity is used to toggle OPT++’s debug mode using the setDebug() member function. Internal to OPT++, there are 3 search strategies, while the DAKOTA search_method specification supports 4 (value_based_line_search, gradient_based_line_search, trust_region, or tr_pds). The difference stems from the "is_expensive" flag in OPT++. If the search strategy is LineSearch and "is_expensive" is turned on, then the value_based_line_search is used. Otherwise (the "is_expensive" default is off), the algorithm will use the gradient_based_line_search. Refer to [Meza, J.C., 1994] and to the OPT++ source in the Dakota/packages/OPTPP directory for information on OPT++ class member functions.
13.153.2 Constructor & Destructor Documentation

SNLLOptimizer ( ProblemDescDB & problem_db, Model & model )

standard constructor

This constructor is used for normal instantiations using data from the ProblemDescDB.

References Dakota::abort_handler(), Minimizer::boundConstraintFlag, SNLLBase::centeringParam, SNLLOptimizer::constraint0_evaluator(), SNLLOptimizer::constraint1_evaluator(), SNLLOptimizer::constraint2_evaluator(), Iterator::convergenceTol, Model::fd_gradient_step_size(), SNLLOptimizer::fdnlf1, SNLLOptimizer::fdnlf1Con, ProblemDescDB::get_int(), ProblemDescDB::get_real(), SNLLBase::init_fn(), Model::interval_type(), Iterator::iteratedModel, Dakota::LARGE_SCALE, Iterator::maxEvalConcurrency, Iterator::maxFunctionEvals, Iterator::maxIterations, SNLLBase::maxStep, SNLLBase::meritFn, Iterator::method_enum_to_string(), Iterator::methodName, SNLLOptimizer::nlf0, SNLLOptimizer::nlf1_evaluator(), SNLLOptimizer::nlf1_f1_evaluator(), SNLLOptimizer::nlf1Con, SNLLOptimizer::nlf2, SNLLOptimizer::nlf2_evaluator(), SNLLOptimizer::nlf2Con, SNLLOptimizer::nlfConstraint, SNLLOptimizer::nlfObjective, SNLLOptimizer::nlpConstraint, Minimizer::numConstraints, Minimizer::numContinuousVars, Minimizer::numNonLinearConstraints, SNLLOptimizer::optbcqnewton, SNLLOptimizer::optbcnewton, SNLLOptimizer::optcg, SNLLOptimizer::optfdnewton, SNLLOptimizer::optfdnips, SNLLOptimizer::optlbfgs, SNLLOptimizer::optnips, SNLLOptimizer::optpds, SNLLOptimizer::optqnewton, SNLLOptimizer::optqnips, Iterator::outputLevel, Iterator::probDescDB, SNLLBase::searchStrat, SNLLBase::solverPostInstantiate(), SNLLBase::solverPreInstantiate(), SNLLBase::stepLenToBndry, SNLLOptimizer::theOptimizer, and Minimizer::vendorNumericalGradFlag.

SNLLOptimizer ( const String & method_string, Model & model )

alternate constructor for instantiations "on the fly"

This is an alternate constructor for instantiations on the fly using a Model but no ProblemDescDB.

References Minimizer::boundConstraintFlag, SNLLOptimizer::constraint1_evaluator(), Iterator::convergenceTol, Model::fd_gradient_step_size(), SNLLBase::init_fn(), Model::interval_type(), Iterator::iteratedModel, Dakota::LARGE_SCALE, Iterator::maxFunctionEvals, Iterator::maxIterations, SNLLBase::meritFn, Iterator::method_name, SNLLOptimizer::nlf1, SNLLOptimizer::nlf1_evaluator(), SNLLOptimizer::nlf1Con, SNLLOptimizer::nlfConstraint, SNLLOptimizer::nlfObjective, SNLLOptimizer::nlpConstraint, Minimizer::numConstraints, Minimizer::numContinuousVars, Minimizer::numNonLinearConstraints, SNLLOptimizer::optbcqnewton, SNLLOptimizer::optbcnewton, SNLLOptimizer::optcg, SNLLOptimizer::optfdnewton, SNLLOptimizer::optfdnips, SNLLOptimizer::optlbfgs, SNLLOptimizer::optnips, SNLLOptimizer::optqnewton, SNLLOptimizer::optqnips, Iterator::outputLevel, SNLLBase::searchStrat, SNLLBase::solverPostInstantiate(), SNLLBase::solverPreInstantiate(), SNLLOptimizer::theOptimizer, and Minimizer::vendorNumericalGradFlag.

SNLLOptimizer ( const RealVector & initialPt, 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 RealVector & lin_eq_coeffs, const RealVector & lin_eq_tgtfs, const RealMatrix & nln_ineq_l_bnds, const RealVector & nln_ineq_u_bnds, const RealVector & nln_eq_tgtfs, void(*)(int mode, int n, const RealVector & x, double & f, RealVector & grad_f, int & result_mode) user_obj_eval, void(*)(int mode, int n, const RealVector & x, RealVector & 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.

References Minimizer::bigRealBoundSize, Minimizer::boundConstraintFlag, SNLLBase::init_fn(), SNLLOptimizer::initialPoint, Dakota::LARGE_SCALE, SNLLOptimizer::lowerBounds, SNLLBase::meritFn, SNLLOptimizer::nlf1, SNLLOptimizer::nlf1Con, SNLLOptimizer::nlfConstraint, SNLLOptimizer::nlfObjective, SNLLOptimizer::nlpConstraint, Minimizer::numConstraints, Minimizer::numContinuousVars, Minimizer::numNonLinearConstraints, SNLLOptimizer::optbcqnewton, SNLLOptimizer::optlbfgs, SNLLOptimizer::optqnewton, SNLLOptimizer::optqnips,
13.153.3 Member Function Documentation

void nlf0_evaluator ( int n, const RealVector & x, double & f, int & result_mode ) [static], [private]

objective function evaluator function for OPT++ methods which require only function values.

For use when DAKOTA computes f and gradients are not directly available. This is used by nongradient-based optimizers such as PDS and by gradient-based optimizers in vendor numerical gradient mode (opt++’s internal finite difference routine is used).

References Iterator::outputLevel, Model::compute_response(), Model::continuous_variables(), Model::current_response(), Response::function_value(), Iterator::iteratedModel, SNLLBase::lastEvalVars, SNLLBase::lastFnEvalLocn, Minimizer::numNonlinearConstraints, Iterator::outputLevel, Model::primary_response_fn_sense(), and SNLLOptimizer::snllOptInstance.

Referenced by SNLLOptimizer::SNLLOptimizer().

void nlf1_evaluator ( int mode, int n, const RealVector & x, double & f, RealVector & grad_f, int & result_mode ) [static], [private]

objective function evaluator function which provides function values and gradients to OPT++ methods.

For use when DAKOTA computes f and df/dX (regardless of gradient type). Vendor numerical gradient case is handled by nlf0_evaluator.

References Iterator::activeSet, Model::compute_response(), Model::continuous_variables(), Model::current_response(), Response::function_gradients_copy(), Response::function_value(), Iterator::iteratedModel, SNLLBase::lastEvalMode, SNLLBase::lastEvalVars, SNLLBase::lastFnEvalLocn, Minimizer::numNonlinearConstraints, Iterator::outputLevel, Model::primary_response_fn_sense(), ActiveSet::request_values(), and SNLLOptimizer::snllOptInstance.

Referenced by SNLLOptimizer::SNLLOptimizer().

void nlf2_evaluator ( int mode, int n, const RealVector & x, double & f, RealVector & grad_f, RealSymMatrix & hess_f, int & result_mode ) [static], [private]

objective function evaluator function which provides function values, gradients, and Hessians to OPT++ methods.

For use when DAKOTA receives f, df/dX, & d^2f/dx^2 from the ApplicationInterface (analytic only). Finite differencing does not make sense for a full Newton approach, since lack of analytic gradients & Hessian should dictate the use of quasi-newton or fd-newton. Thus, there is no fdnlf2_evaluator for use with full Newton approaches, since it is preferable to use quasi-newton or fd-newton with nlf1. Gauss-Newton does not fit this model; it uses nlf2_evaluator gn instead of nlf2_evaluator.

References Iterator::activeSet, Model::compute_response(), Model::continuous_variables(), Model::current_response(), Response::function_gradients_copy(), Response::hessian(), Response::function_value(), Iterator::iteratedModel, SNLLBase::lastEvalMode, SNLLBase::lastEvalVars, SNLLBase::lastFnEvalLocn, Minimizer::numNonlinearConstraints, Iterator::outputLevel, Model::primary_response_fn_sense(), ActiveSet::request_values(), and SNLLOptimizer::snllOptInstance.

Referenced by SNLLOptimizer::SNLLOptimizer().

void constraint0_evaluator ( int n, const RealVector & x, RealVector & g, int & result_mode ) [static], [private]

constraint evaluator function for OPT++ methods which require only constraint values.
For use when DAKOTA computes g and gradients are not directly available. This is used by non-gradient-based optimizers and by gradient-based optimizers in vendor numerical gradient mode (opt++’s internal finite difference routine is used).

References Model::compute_response(), Model::continuous_variables(), SNLLBase::copy_con_vals_dak_to_optpp(), Model::current_response(), Response::function_values(), Iterator::iteratedModel, SNLLBase::lastEvalVars, SNLLBase::lastFnEvalLocn, Optimizer::numObjectiveFns, Iterator::outputLevel, and SNLLOptimizer::snllOptInstance.

Referenced by SNLLOptimizer::SNLLOptimizer().

```cpp
void constraint1_evaluator ( int mode, int n, const RealVector & x, RealVector & g, RealMatrix & grad_g, int & result_mode ) [static], [private]
```
constraint evaluator function which provides constraint values and gradients to OPT++ methods.

For use when DAKOTA computes g and dg/dX (regardless of gradient type). Vendor numerical gradient case is handled by constraint0_evaluator.

References Iterator::activeSet, Model::compute_response(), Model::continuous_variables(), SNLLBase::copy_con_grad(), SNLLBase::copy_con_vals_dak_to_optpp(), Model::current_response(), Response::function_gradients(), Response::function_values(), Iterator::iteratedModel, SNLLBase::lastEvalMode, SNLLBase::lastEvalVars, SNLLBase::lastFnEvalLocn, Optimizer::numObjectiveFns, Iterator::outputLevel, ActiveSet::request_values(), and SNLLOptimizer::snllOptInstance.

Referenced by SNLLOptimizer::SNLLOptimizer().

```cpp
void constraint2_evaluator ( int mode, int n, const RealVector & x, RealVector & g, RealMatrix & grad_g, OPTPP::OptppArray<RealSymMatrix> & hess_g, int & result_mode ) [static], [private]
```
constraint evaluator function which provides constraint values, gradients, and Hessians to OPT++ methods.

For use when DAKOTA computes g, dg/dX, & d^2g/dx^2 (analytic only).

References Iterator::activeSet, Model::compute_response(), Model::continuous_variables(), SNLLBase::copy_con_grad(), SNLLBase::copy_con_hess(), SNLLBase::copy_con_vals_dak_to_optpp(), Model::current_response(), Response::function_gradients(), Response::function_hessians(), Response::function_values(), Iterator::iteratedModel, SNLLBase::lastEvalMode, SNLLBase::lastEvalVars, SNLLBase::lastFnEvalLocn, Optimizer::numObjectiveFns, Iterator::outputLevel, ActiveSet::request_values(), and SNLLOptimizer::snllOptInstance.

Referenced by SNLLOptimizer::SNLLOptimizer().

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

- SNLLOptimizer.hpp
- SNLLOptimizer.cpp

## 13.154 SOLBase Class Reference

Base class for Stanford SOL software.

Inheritance diagram for SOLBase:
Public Member Functions

- **SOLBase ()**
  
  default constructor
- **SOLBase (Model &model)**
  
  standard constructor
- **~SOLBase ()**
  
  destructor

Protected Member Functions

- void **allocate_arrays** (int num_cv, 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** (int num_cv, int num_nln_con, int num_lin_con, 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, int verify_lev, Real fn_prec, Real linesrch_tol, int max_iter, Real constr_tol, Real conv_tol, const std::string &grad_type, const RealVector &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 &nln_ineq_l_bnds, const RealVector &nln_ineq_u_bnds, const RealVector &nln_eq_targets)

  augments variable bounds with linear and nonlinear constraint bounds.

Static Protected Member Functions

- static void **constraint_eval** (int &mode, int &ncnln, int &n, int &nrowj, int *needc, double *x, double *c, double *cjac, int &nstate)

  CONFUN in NPSOL manual: computes the values and first derivatives of the nonlinear constraint functions.

Protected Attributes

- int **realWorkspaceSize**
  
  size of realWorkspace
- int **intWorkspaceSize**
  
  size of intWorkspace
- RealArray **realWorkspace**

  real work space for NPSOL/NLSSOL
- IntArray **intWorkspace**

  int work space for NPSOL/NLSSOL
- int **nlnConstraintArraySize**

  used for non-zero array sizing (nonlinear constraints)
- int **linConstraintArraySize**
used for non-zero array sizing (linear constraints)

- **RealArray** cLambda
  
  CLAMBDA from NPSOL manual: 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
  
  length of augmented bounds arrays (variable bounds plus linear and nonlinear constraint bounds)

- **double** linConstraintMatrixF77
  
  [A] matrix from NPSOL manual: linear constraint coefficients

- **double** upperFactorHessianF77
  

- **double** constraintJacMatrixF77
  
  [CJAC] matrix from NPSOL manual: nonlinear constraint Jacobian

- **int** fnEvalCntr
  
  counter for testing against maxFunctionEvals

- **size_t** constrOffset
  
  used in constraint_eval() to bridge NLSSOLLeastSq::numLeastSqTerms and NPSOLOptimizer::numObjectiveFns

### Static Protected Attributes

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

- **static** Minimizer * optLSqInstance
  
  pointer to the active base class object instance used within the static evaluator functions in order to avoid the need for static data

### 13.154.1 Detailed Description

Base class for Stanford SOL software.

The SOLBase class provides a common base class for NPSOLOptimizer and NLSSOLLeastSq, both of which are Fortran 77 sequential quadratic programming algorithms from Stanford University marketed by Stanford Business Associates.

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

- SOLBase.hpp
- SOLBase.cpp
13.155 SpawnApplicInterface Class Reference

Derived application interface class which spawns simulation codes using spawnvp.

Inheritance diagram for SpawnApplicInterface:

```
    Interface
     |        
    ApplicationInterface
     |        
    ProcessApplicInterface
     |        
  ProcessHandleApplicInterface
    |        
  SpawnApplicInterface
```

Public Member Functions

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

- **~SpawnApplicInterface** ()
  
  *destructor*

Protected Member Functions

- **void wait_local_evaluations** (PRPQueue &prp_queue)
  
  For asynchronous function evaluations, this method is used to detect completion of jobs and process their results. It provides the processing code that is specific to derived classes. This version waits for at least one completion.

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

- **pid_t create_analysis_process** (bool block_flag, bool new_group)
  
  *spawn a child process for an analysis component within an evaluation*

- **size_t wait_local_analyses** ()
  
  *wait for asynchronous analyses on the local processor, completing at least one job*

- **size_t test_local_analyses_send** (int analysis_id)
  
  *test for asynchronous analysis completions on the local processor and return results for any completions by sending messages*

Additional Inherited Members

13.155.1 Detailed Description

Derived application interface class which spawns simulation codes using spawnvp.
SpawnApplicInterface is used on Windows systems and is a peer to ForkApplicInterface for Unix systems. The documentation for this class was generated from the following files:

- SpawnApplicInterface.hpp
- SpawnApplicInterface.cpp

### 13.156 SurfpackApproximation Class Reference

Derived approximation class for Surfpack approximation classes. Interface between Surfpack and Dakota.

Inheritance diagram for SurfpackApproximation:

```
Approximation
```

```
SurfpackApproximation
```

#### Public Member Functions

- **SurfpackApproximation ()**
  - default constructor

- **SurfpackApproximation (const ProblemDescDB &problem_db, const SharedApproxData &shared_data)**
  - standard constructor: Surfpack surface of appropriate type will be created

- **SurfpackApproximation (const SharedApproxData &shared_data)**
  - alternate constructor

- **~SurfpackApproximation ()**
  - destructor

#### Protected Member Functions

- **int min_coefficients () const**
  - return the minimum number of samples (unknowns) required to build the derived class approximation type in num-Vars dimensions

- **int recommended_coefficients () const**
  - return the recommended number of samples (unknowns) required to build the derived class approximation type in numVars dimensions

- **void build ()**
  - SurfData object will be created from Dakota’s SurrogateData, and the appropriate Surfpack build method will be invoked.

- **Real value (const Variables &vars)**
  - Return the value of the Surfpack surface for a given parameter vector x.

- **const RealVector & gradient (const Variables &vars)**
  - retrieve the approximate function gradient for a given parameter vector x

- **const RealSymMatrix & hessian (const Variables &vars)**
  - retrieve the approximate function Hessian for a given parameter vector x

- **Real prediction_variance (const Variables &vars)**
retrieve the variance of the predicted value for a given parameter set x (KrigingModel only)

- bool diagnostics_available ()
  check if the diagnostics are available (true for the Surfpack types)
- Real diagnostic (const String &metric_type)
  retrieve a single diagnostic metric for the diagnostic type specified on the primary model and data
- Real diagnostic (const String &metric_type, const SurfpackModel &model, const SurfData &data)
  retrieve a single diagnostic metric for the diagnostic type specified on the given model and data

- void primary_diagnostics (int fn_index)
  compute and print all requested diagnostics and cross-validation
- void challenge_diagnostics (const RealMatrix &challenge_points, int fn_index)
  compute and print all requested diagnostics for user provided challenge pts

Private Member Functions

- SurfData * surrogates_to_surf_data ()
  copy from SurrogateData to SurfPoint/SurfData
- void add_anchor_to_surfdata (SurfData &surf_data)
  set the anchor point (including gradient and hessian if present) into surf_data

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.

Additional Inherited Members

13.156.1 Detailed Description

Derived approximation class for Surfpack approximation classes. Interface between Surfpack and Dakota.

The SurfpackApproximation class is the interface between Dakota and Surfpack. Based on the information in the ProblemDescDB that is passed in through the constructor, SurfpackApproximation builds a Surfpack Surface object that corresponds to one of the following data-fitting techniques: polynomial regression, kriging, artificial neural networks, radial basis function network, or multivariate adaptive regression splines (MARS).

13.156.2 Constructor & Destructor Documentation

SurfpackApproximation (const ProblemDescDB & problem_db, const SharedApproxData & shared_data)

standard constructor: Surfpack surface of appropriate type will be created

Initialize the embedded Surfpack surface object and configure it using the specifications from the input file. Data for the surface is created later.

References Dakota::abort_handler(), SharedSurfpackApproxData::approxOrder, SharedApproxData::approx_Type, SharedApproxData::buildDataOrder, Dakota::copy_data(), SharedSurfpackApproxData::crossValidateFlag,
SurfpackApproximation ( const SharedApproxData & shared_data )
alternate constructor
  On-the-fly constructor which uses mostly Surfpack model defaults.
    References Dakota::abort_handler(), SharedSurfpackApproxData::approxOrder, SharedApproxData::approxType, SharedApproxData::buildDataOrder, SurfpackApproximation::factory, SharedSurfpackApproxData::merge_variable_arrays(), SurfpackApproximation::model, SharedApproxData::outputLevel, Approximation::build(), SharedSurfpackApproxData::exportModelName, SurfpackApproximation::surfData, and SurfpackApproximation::surrogates_to_surf_data().

13.156.3 Member Function Documentation

void build() [protected], [virtual]
 SurfData object will be created from Dakota’s SurrogateData, and the appropriate Surfpack build method will be invoked.
   surfData will be deleted in dtor
   Todo  Right now, we’re completely deleting the old data and then recopying the current data into a SurfData object. This was just the easiest way to arrive at a solution that would build and run. This function is frequently called from addPoint rebuild, however, and it’s not good to go through this whole process every time one more data point is added.
     Reimplemented from Approximation.
     References Dakota::abort_handler(), SharedApproxData::approxCLowerBnds, SharedApproxData::approxCUpperBnds, SharedApproxData::approxDILowerBnds, SharedApproxData::approxDIUpperBnds, SharedApproxData::approxDRLowerBnds, SharedApproxData::approxDRUpperBnds, Approximation::build(), SharedSurfpackApproxData::exportModelName, SurfpackApproximation::factory, SharedSurfpackApproxData::merge_variable_arrays(), SurfpackApproximation::model, SharedApproxData::outputLevel, Approximation::build(), SharedSurfpackApproxData::exportModelName, SurfpackApproximation::surfData, and SurfpackApproximation::surrogates_to_surf_data().

const RealSymMatrix & hessian( const Variables & vars ) [protected], [virtual]
 retrieve the approximate function Hessian for a given parameter vector x
   Todo  Make this acceptably efficient
     Reimplemented from Approximation.
   References Dakota::abort_handler(), Approximation::approxHessian, SharedApproxData::approxType, Variables::cv(), SurfpackApproximation::model, and Approximation::build().
     Referenced by SurfpackApproximation::add_anchor_to_surfdata().

SurfData * surrogates_to_surf_data() [private]
 copy from SurrogateData to SurfPoint/SurfData
   Copy the data stored in Dakota-style SurrogateData into Surfpack-style SurfPoint and SurfData objects.
    References SurfpackApproximation::add_anchor_to_surfdata(), SharedSurfpackApproxData::add_sd_to_surfdata(), Approximation::approxData, SharedApproxData::buildDataOrder, SurfpackApproximation::factory, SharedApproxData::outputLevel, and Approximation::build().
     Referenced by SurfpackApproximation::build().
void add_anchor_to_sufdata ( SurfData & surf_data ) [private]

set the anchor point (including gradient and hessian if present) into surf_data

If there is an anchor point, add an equality constraint for its response value. Also add constraints for gradient
and hessian, if applicable.

References Dakota::abort_handler(), Approximation::approxData, Dakota::copy_data(), SharedSurfpackApprox-
Data::copy_matrix(), SurfpackApproximation::gradient(), SurfpackApproximation::hessian(), SharedApproxData-
::outputLevel, SharedSurfpackApproxData::svd_to_reaarray(), and Approximation::sharedDataRep.
The documentation for this class was generated from the following files:

- SurfpackApproximation.hpp
- SurfpackApproximation.cpp

13.157 SurrBasedGlobalMinimizer Class Reference

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

Inheritance diagram for SurrBasedGlobalMinimizer:

```
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SurrBasedGlobalMinimizer</td>
</tr>
<tr>
<td>------------------</td>
</tr>
<tr>
<td>SurrBasedMinimizer</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

- SurrBasedGlobalMinimizer (ProblemDescDB &problem_db, Model &model)
  constructor
- ~SurrBasedGlobalMinimizer ()
  destructor

Protected Member Functions

- void init_communicators ()
  initialize the communicators associated with iteratedModel as well as any subordinate iterators or subordinate
  models
- void free_communicators ()
  free the communicators associated with iteratedModel as well as any subordinate iterators or subordinate models
- bool returns_multiple_points () const
  Global surrogate-based methods can return multiple points.
Private Member Functions

- void minimize_surrogates()
  
  Performs global surrogate-based optimization by repeatedly optimizing on and improving surrogates of the response functions.

Private Attributes

- bool replacePoints
  
  Flag for replacing the previous iteration’s point additions, rather than continuing to append, during construction of the next surrogate

Additional Inherited Members

13.157.1 Detailed Description

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

This method uses a SurrogateModel to perform minimization (optimization or nonlinear least squares) through a set of iterations. At each iteration, a surrogate is built, the surrogate is minimized, and the optimal points from the surrogate are then evaluated with the ”true” function, to generate new points upon which the surrogate for the next iteration is built.

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

- SurrBasedGlobalMinimizer.hpp
- SurrBasedGlobalMinimizer.cpp

13.158 SurrBasedLocalMinimizer Class Reference

Class for provably-convergent local surrogate-based optimization and nonlinear least squares.

Inheritance diagram for SurrBasedLocalMinimizer:

```
Iterator
    
Minimizer
    
SurrBasedMinimizer
    
SurrBasedLocalMinimizer
```

Public Member Functions

- SurrBasedLocalMinimizer (ProblemDescDB &problem_db, Model &model)
  
  Constructor

- ~SurrBasedLocalMinimizer()
  
  Destructor
CHAPTER 13. CLASS DOCUMENTATION

Protected Member Functions

- void `init_communicators()`
  
  initialize the communicators associated with iteratedModel as well as any subordinate iterators or subordinate models

- void `free_communicators()`
  
  free the communicators associated with iteratedModel as well as any subordinate iterators or subordinate models

- void `reset()`
  
  reset convergence controls in case of multiple SBLM executions

Private Member Functions

- void `minimize_surrogates()`
  
  Performs local surrogate-based minimization by minimizing local, global, or hierarchical surrogates over a series of trust regions.

- bool `tr_bounds` (const RealVector &global_lower_bnds, const RealVector &global_upper_bnds, RealVector &tr_lower_bnds, RealVector &tr_upper_bnds)
  
  compute current trust region bounds

- void `find_center_truth` (const Iterator &dace_iterator, Model &truth_model)
  
  retrieve responseCenterTruth if possible, evaluate it if not

- void `find_center_approx` ()
  
  retrieve responseCenter_approx if possible, evaluate it if not

- void `hard_convergence_check` (const Response &response_truth, const RealVector &c_vars, const RealVector &lower_bnds, const RealVector &upper_bnds)
  
  check for hard convergence (norm of projected gradient of merit function near zero)

- void `tr_ratio_check` (const RealVector &c_vars_star, const RealVector &tr_lower_bounds, const RealVector &tr_upper_bounds)
  
  compute trust region ratio (for SBLM iterate acceptance and trust region resizing) and check for soft convergence (diminishing returns)

- void `update_penalty` (const RealVector &fns_center_truth, const RealVector &fns_star_truth)
  
  initialize and update the penaltyParameter

- void `relax_constraints` (const RealVector &lower_bnds, const RealVector &upper_bnds)
  
  relax constraints by updating bounds when current iterate is infeasible

Static Private Member Functions

- static void `approx_subprob_objective_eval` (const Variables &surrogate_vars, const Variables &recast_vars, const Response &surrogate_response, Response &recast_response)
  
  static function used to define the approximate subproblem objective.

- static void `approx_subprob_constraint_eval` (const Variables &surrogate_vars, const Variables &recast_vars, const 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 &)
  
  static function used by NPSOL as the objective function in the homotopy constraint relaxation formulation.

- static void `hom_constraint_eval` (int &mode, int &ncnln, int &n, int &nrowj, int *needc, double *tau_and_x, double *c, double *cjac, int &nstate)
  
  static function used by NPSOL as the constraint function in the homotopy constraint relaxation formulation.
Private Attributes

- Real origTrustRegionFactor
  original user specification for trustRegionFactor
- Real trustRegionFactor
  the trust region factor is used to compute the total size of the trust region – it is a percentage, e.g. for trustRegionFactor = 0.1, the actual size of the trust region will be 10% of the global bounds (upper bound - lower bound for each design variable).
- Real minTrustRegionFactor
  a soft convergence control: stop SBLM when the trust region factor is reduced below the value of minTrustRegionFactor
- Real trRatioContractValue
  trust region ratio min value: contract tr if ratio below this value
- Real trRatioExpandValue
  trust region ratio sufficient value: expand tr if ratio above this value
- Real gammaContract
  trust region contraction factor
- Real gammaExpand
  trust region expansion factor
- short approxSubProbObj
  type of approximate subproblem objective: ORIGINAL_OBJ, LAGRANGIAN_OBJ, or AUGMENTED_LAGRANGIAN_OBJ
- short approxSubProbCon
  type of approximate subproblem constraints: NO_CON, LINEARIZED_CON, or ORIGINAL_CON
- Model approxSubProbModel
  the approximate sub-problem formulation solved on each approximate minimization cycle: may be a shallow copy of iteratedModel, or may involve a RecastModel recursion applied to iteratedModel
- bool recastSubProb
  flag to indicate when approxSubProbModel involves a RecastModel recursion
- short trConstraintRelax
  type of trust region constraint relaxation for infeasible starting points: NO_RELAX or HOMOTOPY
- short meritFnType
  type of merit function used in trust region ratio logic: PENALTY_MERIT, ADAPTIVE_PENALTY_MERIT, LAGRANGIAN_MERIT, or AUGMENTED_LAGRANGIAN_MERIT
- short acceptLogic
  type of iterate acceptance test logic: FILTER or TR_RATIO
- int penaltyIterOffset
  iteration offset used to update the scaling of the penalty parameter for adaptive_penalty merit functions
- short convergenceFlag
  code indicating satisfaction of hard or soft convergence conditions
- short softConvCount
  number of consecutive candidate point rejections. If the count reaches softConvLimit, stop SBLM.
- short softConvLimit
  the limit on consecutive candidate point rejections. If exceeded by softConvCount, stop SBLM.
- bool truthGradientFlag
• bool approxGradientFlag
  flags the use/availability of truth gradients within the SBLM process
• bool truthHessianFlag
  flags the use/availability of truth Hessians within the SBLM process
• bool approxHessianFlag
  flags the use/availability of surrogate Hessians within the SBLM process
• short correctionType
  flags the use of surrogate correction techniques at the center of each trust region
• bool globalApproxFlag
  flags the use of a global data fit surrogate (rsm, ann, mars, kriging)
• bool multiptApproxFlag
  flags the use of a multipoint data fit surrogate (TANA)
• bool localApproxFlag
  flags the use of a local data fit surrogate (Taylor series)
• bool hierarchApproxFlag
  flags the use of a model hierarchy/multifidelity surrogate
• bool newCenterFlag
  flags the acceptance of a candidate point and the existence of a new trust region center
• bool daceCenterPtFlag
  flags the availability of the center point in the DACE evaluations for global approximations (CCD, Box-Behnken)
• bool multiLayerBypassFlag
  flags the simultaneous presence of two conditions: (1) additional layerings w/i actual model (e.g., surrogateModel = layered/nested/layered -> actual_model = nested/layered), and (2) a user-specification to bypass all layerings within actual_model for the evaluation of truth data (responseCenterTruth and responseStarTruth).
• bool useDerivsFlag
  flag for the "use derivatives" specification for which derivatives are to be evaluated at each DACE point in global surrogate builds.
• RealVector nonlinIneqLowerBndsSlack
  individual violations of nonlinear inequality constraint lower bounds
• RealVector nonlinIneqUpperBndsSlack
  individual violations of nonlinear inequality constraint upper bounds
• RealVector nonlinEqTargetsSlack
  individual violations of nonlinear equality constraint targets
• Real tau
  constraint relaxation parameter
• Real alpha
  constraint relaxation parameter backoff parameter (multiplier)
• Variables varsCenter
  variables at the trust region center
• Response responseCenterApprox
  approx response at trust region center
• Response responseStarApprox
  approx response at SBLM cycle minimum
• IntResponsePair responseCenterTruth
  truth response at trust region center
• IntResponsePair responseStarTruth
  truth response at SBLM cycle minimum

Static Private Attributes
• static SurrBasedLocalMinimizer * sblmInstance
  pointer to SBLM instance used in static member functions

Additional Inherited Members

13.158.1 Detailed Description
Class for provably-convergent local surrogate-based optimization and nonlinear least squares.

This minimizer uses a SurrogateModel to perform minimization based on local, global, or hierarchical surrogates. It achieves provable convergence through the use of a sequence of trust regions and the application of surrogate corrections at the trust region centers.

13.158.2 Member Function Documentation

void minimize_surrogates( ) [private], [virtual]
Performs local surrogate-based minimization by minimizing local, global, or hierarchical surrogates over a series of trust regions.

Trust region-based strategy to perform surrogate-based optimization in subregions (trust regions) of the parameter space. The minimizer operates on approximations in lieu of the more expensive simulation-based response functions. The size of the trust region is varied according to the goodness of the agreement between the approximations and the true response functions.

Implements SurrBasedMinimizer.

References Dakota::abort_handler(), Iterator::active_set(), Response::active_set(), Model::active_variables(), Graphics::add_datapoint(), DiscrepancyCorrection::apply(), SurrBasedLocalMinimizer::approxGradientFlag, SurrBasedLocalMinimizer::approxHessianFlag, SurrBasedMinimizer::approxSubProbMinimizer, SurrBasedLocalMinimizer::approxSubProbModel, Iterator::bestResponseArray, Iterator::bestVariablesArray, Model::build_approximation(), Model::component_parallel_mode(), DiscrepancyCorrection::compute(), Model::compute_response(), Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), Variables::continuous_variables(), Model::continuous_variables(), SurrBasedLocalMinimizer::convergenceFlag, Variables::copy(), Dakota::copy_data(), SurrBasedLocalMinimizer::correctionType, Model::current_response(), Model::current_variables(), SurrBasedLocalMinimizer::daceCenterPtFlag, Dakota::dakota_graphics, Model::discrepancy_correction(), Model::evaluation_id(), SurrBasedLocalMinimizer::find_center_approx(), SurrBasedLocalMinimizer::find_center_truth(), SurrBasedLocalMinimizer::globalApproxFlag, SurrBasedLocalMinimizer::hard_convergence_check(), Iterator::is_null(), Iterator::iteratedModel, SurrBasedLocalMinimizer::LocalApproxFlag, Iterator::maxIterations, SurrBasedLocalMinimizer::minTrustRegionFactor, SurrBasedLocalMinimizer::multiLayerBypassFlag, SurrBasedLocalMinimizer::multiptApproxFlag, SurrBasedLocalMinimizer::newCenterFlag, Model::nonlinear_eq_constraint_targets(), Model::nonlinear_ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Minimizer::numContinuousVars, SurrBasedMinimizer::origNonlinEqTargets, SurrBasedMinimizer::origNonlinIneqLowerBnds, SurrBasedMinimizer::origNonlinIneqUpperBnds, SurrBasedLocalMinimizer::recastSubProb, SurrBasedLocalMinimizer::relax_constraints(), ActiveSet::request_values(), SurrBasedLocalMinimizer::reset(), Iterator::response_results(), SurrBasedLocalMinimizer::responseCenterApprox, SurrBasedLocalMinimizer::responseCenterTruth, SurrBasedLocalMinimizer::responseStarApprox, SurrBasedLocalMinimizer::responseStarTruth, Iterator::run(), Iterator::sampling_scheme(), SurrBased-
Minimizer::sbIterNum, SurrBasedLocalMinimizer::sblmInstance, SurrBasedLocalMinimizer::softConvCount, SurrBasedLocalMinimizer::softConvLimit, Model::subordinate_iterator(), Model::surrogate_model(), Model::surrogate_response_mode(), SurrBasedLocalMinimizer::tr_bounds(), SurrBasedLocalMinimizer::tr_ratio_check(), SurrBasedLocalMinimizer::trConstraintRelax, SurrBasedLocalMinimizer::trustRegionFactor, Model::truth_model(), SurrBasedLocalMinimizer::truthGradientFlag, SurrBasedLocalMinimizer::truthHessianFlag, Response::update(), SurrBasedLocalMinimizer::useDerivsFlag, Iterator::variables_results(), and SurrBasedLocalMinimizer::varsCenter.

```cpp
void hard_convergence_check ( const Response & response_truth, const RealVector & c_vars, const RealVector & lower_bnds, const RealVector & upper_bnds ) [private]
```

check for hard convergence (norm of projected gradient of merit function near zero)

The hard convergence check computes the gradient of the merit function at the trust region center, performs a projection for active bound constraints (removing any gradient component directed into an active bound), and signals convergence if the 2-norm of this projected gradient is less than convergenceTol.

References SurrBasedLocalMinimizer::acceptLogic, SurrBasedLocalMinimizer::approxSubProbObj, SurrBasedMinimizer::constraintViolation(), Minimizer::constraintTol, SurrBasedLocalMinimizer::convergenceFlag, Iterator::convergenceTol, Response::function_gradients(), Response::function_values(), Iterator::iteratedModel, SurrBasedMinimizer::lagrangianGradient(), SurrBasedLocalMinimizer::meritFnType, Minimizer::numNonlinearConstraints, SurrBasedMinimizer::origNonlinEqTargets, SurrBasedMinimizer::origNonlinIneqLowerBnds, SurrBasedMinimizer::origNonlinIneqUpperBnds, SurrBasedMinimizer::penaltyMerit(), Model::primary_response_fn_sense(), Model::primary_response_fn_weights(), SurrBasedMinimizer::sbIterNum, SurrBasedLocalMinimizer::truthGradientFlag, SurrBasedMinimizer::update_augmented_lagrange_multipliers(), SurrBasedMinimizer::update_filter(), and SurrBasedMinimizer::update_penalty().

Referenced by SurrBasedLocalMinimizer::minimize_surrogates().

```cpp
void tr_ratio_check ( const RealVector & c_vars_star, const RealVector & tr_lower_bnds, const RealVector & tr_upper_bnds ) [private]
```

compute trust region ratio (for SBLM iterate acceptance and trust region resizing) and check for soft convergence (diminishing returns)

Assess acceptance of SBLM iterate (trust region ratio or filter) and compute soft convergence metrics (number of consecutive failures, min trust region size, etc.) to assess whether the convergence rate has decreased to a point where the process should be terminated (diminishing returns).

References SurrBasedLocalMinimizer::acceptLogic, SurrBasedLocalMinimizer::approxSubProbObj, SurrBasedMinimizer::augmented_lagrangian_merit(), SurrBasedMinimizer::constraintViolation(), Minimizer::constraintTol, Iterator::convergenceTol, SurrBasedMinimizer::etaSequence, Response::function_values(), SurrBasedLocalMinimizer::gammaContract, SurrBasedLocalMinimizer::gammaExpand, SurrBasedLocalMinimizer::globalApproxFlag, Iterator::iteratedModel, SurrBasedMinimizer::lagrangian_merit(), SurrBasedLocalMinimizer::meritFnType, SurrBasedLocalMinimizer::newCenterFlag, Minimizer::numContinuousVars, SurrBasedMinimizer::origNonlinEqTargets, SurrBasedMinimizer::origNonlinIneqLowerBnds, SurrBasedMinimizer::origNonlinIneqUpperBnds, SurrBasedMinimizer::penalty_merit(), Model::primary_response_fn_sense(), Model::primary_response_fn_weights(), SurrBasedLocalMinimizer::responseCenterApprox, SurrBasedLocalMinimizer::responseCenterTruth, SurrBasedLocalMinimizer::responseStarApprox, SurrBasedLocalMinimizer::responseStarTruth, SurrBasedLocalMinimizer::softConvCount, SurrBasedLocalMinimizer::trRatioContractValue, SurrBasedLocalMinimizer::trRatioExpandValue, SurrBasedLocalMinimizer::trustRegionFactor, SurrBasedMinimizer::update_augmented_lagrange_multipliers(), SurrBasedMinimizer::update_filter(), and SurrBasedLocalMinimizer::update_penalty().

Referenced by SurrBasedLocalMinimizer::minimize_surrogates().
void update_penalty ( const RealVector & fns_center_truth, const RealVector & fns_star_truth )
[private]
initialize and update the penaltyParameter
Scaling of the penalty value is important to avoid rejecting SBLM iterates which must increase the objective to achieve a reduction in constraint violation. In the basic penalty case, the penalty is ramped exponentially based on the iteration counter. In the adaptive case, the ratio of relative change between center and star points for the objective and constraint violation values is used to rescale penalty values.
References SurrBasedMinimizer::alphaEta, SurrBasedLocalMinimizer::approxSubProbObj, SurrBasedMinimizer::constraint_violation(), Minimizer::constraintTol, SurrBasedMinimizer::eta, SurrBasedMinimizer::etaSequence, Iterator::iteratedModel, SurrBasedLocalMinimizer::meritFnType, Minimizer::objective(), SurrBasedLocalMinimizer::penaltyIterOffset, SurrBasedMinimizer::penaltyParameter, Model::primary_response_fn_sense(), Model::primary_response_fn_weights(), and SurrBasedMinimizer::sbIterNum.
Referenced by SurrBasedLocalMinimizer::tr_ratio_check().

void approx_subprob_objective_eval ( const Variables & surrogate_vars, const Variables & recast_vars, const Response & surrogate_response, Response & recast_response ) [static], [private]
static function used to define the approximate subproblem objective.
Objective functions evaluator for solution of approximate subproblem using a RecastModel.
References Response::active_set_request_vector(), SurrBasedLocalMinimizer::approxSubProbCon, SurrBasedLocalMinimizer::approxSubProbModel, SurrBasedLocalMinimizer::approxSubProbObj, SurrBasedMinimizer::augmented_lagrangian_gradient(), SurrBasedMinimizer::augmented_lagrangian_merit(), Response::function_gradient(), Response::function_value(), Response::function_gradients(), Response::function grdient(), Response::function_gradient_view(), Response::function_value(), Response::function_gradients(), Response::function_value(), Iterator::iteratedModel, SurrBasedMinimizer::lagrangian_gradient(), SurrBasedMinimizer::lagrangian_merit(), Model::nonlinear_eq_constraint_targets(), Model::nonlinear_ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Minimizer::numUserPrimaryFns, Minimizer::objective(), Minimizer::objective_gradient(), SurrBasedMinimizer::origNonlinEqTargets, SurrBasedMinimizer::origNonlinEqLowerBnds, SurrBasedMinimizer::origNonlinEqUpperBnds, Model::primary_response_fn_sense(), Model::primary_response_fn_weights(), and SurrBasedLocalMinimizer::sblmInstance.
Referenced by SurrBasedLocalMinimizer::SurrBasedLocalMinimizer().

void approx_subprob_constraint_eval ( const Variables & surrogate_vars, const Variables & recast_vars, const Response & surrogate_response, Response & recast_response ) [static], [private]
static function used to define the approximate subproblem constraints.
Constraint functions evaluator for solution of approximate subproblem using a RecastModel.
References Response::active_set_request_vector(), Response::active_set_request_vector(), SurrBasedLocalMinimizer::approxSubProbCon, SurrBasedLocalMinimizer::approxSubProbObj, Variables::continuous_variables(), Response::function_gradient(), Response::function_gradient_view(), Response::function_gradients(), Response::function_value(), Response::function_values(), Minimizer::numUserPrimaryFns, SurrBasedLocalMinimizer::responseCenterApprox, SurrBasedLocalMinimizer::sblmInstance, and SurrBasedLocalMinimizer::varsCenter.
Referenced by SurrBasedLocalMinimizer::SurrBasedLocalMinimizer().

void hom_objective_eval ( int & mode, int & n, double * tau_and_x, double & f, double * grad_f, int & ) [static], [private]
static function used by NPSOL as the objective function in the homotopy constraint relaxation formulation.
NPSOL objective functions evaluator for solution of homotopy constraint relaxation parameter optimization. This constrained optimization problem performs the update of the tau parameter in the homotopy heuristic approach used to relax the constraints in the original problem.
Referenced by SurrBasedLocalMinimizer::relax_constraints().

```cpp
void hom_constraint_eval ( int & mode, int & ncnln, int & n, int & nrowj, int * needc, double * tau_and_x, double * c, double * cjac, int & nstate ) [static], [private]
```

static function used by NPSOL as the constraint function in the homotopy constraint relaxation formulation. NPSOL constraint functions evaluator for solution of homotopy constraint relaxation parameter optimization. This constrained optimization problem performs the update of the tau parameter in the homotopy heuristic approach used to relax the constraints in the original problem.

References Response::active_set(), SurrBasedLocalMinimizer::approxSubProbModel, Model::compute_response(), Model::continuous_variables(), Model::current_response(), Response::function_gradients(), Response::function_values(), SurrBasedLocalMinimizer::nonlinEqTargetsSlack, SurrBasedLocalMinimizer::nonlinIneqLowerBndsSlack, SurrBasedLocalMinimizer::nonlinIneqUpperBndsSlack, Model::num_functions(), Minimizer::numNonlinearEqConstraints, Minimizer::numNonlinearIneqConstraints, ActiveSet::request_vector(), SurrBasedLocalMinimizer::sblmInstance, and SurrBasedLocalMinimizer::tau.

Referenced by SurrBasedLocalMinimizer::relax_constraints().

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

- SurrBasedLocalMinimizer.hpp
- SurrBasedLocalMinimizer.cpp

### 13.159 SurrBasedMinimizer Class Reference

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

Inheritance diagram for SurrBasedMinimizer:

```
   SurrBasedMinimizer
      |                  
      v                  
  Minimizer
      |                  
      v                  
  SurrBasedMinimizer
      |                  
      v                  
  EffGlobalMinimizer  SurrBasedGlobalMinimizer  SurrBasedLocalMinimizer
```

#### Protected Member Functions

- **SurrBasedMinimizer (ProblemDescDB &problem_db, Model &model)**
  
  *constructor*

- **∼SurrBasedMinimizer ()**
  
  *destructor*

- **void initialize_graphics (int iterator_server_id=1)**
  
  *initialize graphics customized for surrogate-based iteration*

- **void core_run ()**
  
  *core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post*

- **void print_results (std::ostream &s)**
virtual void minimize_surrogates ()=0
    Used for computing the optimal solution using a surrogate-based approach. Redefines the Iterator::core_run() virtual function.

void update_lagrange_multipliers (const RealVector &fn_vals, const RealMatrix &fn_grads)
    initialize and update Lagrange multipliers for basic Lagrangian

void update_augmented_lagrange_multipliers (const RealVector &fn_vals)
    initialize and update the Lagrange multipliers for augmented Lagrangian

bool update_filter (const RealVector &fn_vals)
    update a filter from a set of function values

Real lagrangian_merit (const RealVector &fn_vals, const BoolDeque &sense, const RealVector &primary_wts, const RealVector &nln_ineq_l_bnds, const RealVector &nln_ineq_u_bnds, const RealVector &nln_eq_tgts)
    compute a Lagrangian function from a set of function values

void lagrangian_gradient (const RealVector &fn_vals, const RealMatrix &fn_grads, const BoolDeque &sense, const RealVector &primary_wts, const RealVector &nln_ineq_l_bnds, const RealVector &nln_ineq_u_bnds, const RealVector &nln_eq_tgts, RealVector &lag_grad)
    compute the gradient of the Lagrangian function

Real augmented_lagrangian_merit (const RealVector &fn_vals, const BoolDeque &sense, const RealVector &primary_wts, const RealVector &nln_ineq_l_bnds, const RealVector &nln_ineq_u_bnds, const RealVector &nln_eq_tgts)
    compute an augmented Lagrangian function from a set of function values

void augmented_lagrangian_gradient (const RealVector &fn_vals, const RealMatrix &fn_grads, const BoolDeque &sense, const RealVector &primary_wts, const RealVector &nln_ineq_l_bnds, const RealVector &nln_ineq_u_bnds, const RealVector &nln_eq_tgts, RealVector &alag_grad)
    compute the gradient of the augmented Lagrangian function

Real penalty_merit (const RealVector &fn_vals, const BoolDeque &sense, const RealVector &primary_wts)
    compute a penalty function from a set of function values

void penalty_gradient (const RealVector &fn_vals, const RealMatrix &fn_grads, const BoolDeque &sense, const RealVector &primary_wts, const RealVector &pen_grad)
    compute the gradient of the penalty function

Real constraint_violation (const RealVector &fn_vals, const Real &constraint_tol)
    compute the constraint violation from a set of function values

Protected Attributes

- Iterator approxSubProbMinimizer
  the minimizer used on the surrogate model to solve the approximate subproblem on each surrogate-based iteration

- int sbIterNum
  surrogate-based minimization iteration number

- RealVectorList sbFilter
  Set of response function vectors defining a filter (objective vs. constraint violation) for iterate selection/rejection.

- RealVector lagrangeMult
  Lagrange multipliers for basic Lagrangian calculations.

- RealVector augLagrangeMult
  Lagrange multipliers for augmented Lagrangian calculations.
• Real penaltyParameter
  
  the penalization factor for violated constraints used in quadratic penalty calculations; increased in update_penalty()

• RealVector origNonlinIneqLowerBnds
  
  original nonlinear inequality constraint lower bounds (no relaxation)

• RealVector origNonlinIneqUpperBnds
  
  original nonlinear inequality constraint upper bounds (no relaxation)

• RealVector origNonlinEqTargets
  
  original nonlinear equality constraint targets (no relaxation)

• Real eta
  
  constant used in etaSequence updates

• Real alphaEta
  
  power for etaSequence updates when updating penalty

• Real betaEta
  
  power for etaSequence updates when updating multipliers

• Real etaSequence
  
  decreasing sequence of allowable constraint violation used in augmented Lagrangian updates (refer to Conn, Gould, and Toint, section 14.4)

Additional Inherited Members

13.159.1 Detailed Description

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

These minimizers use a SurrogateModel to perform optimization based either on local trust region methods or global updating methods.

13.159.2 Member Function Documentation

void core_run ( ) [inline], [protected], [virtual]

core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post

Virtual run function for the iterator class hierarchy. All derived classes need to redefine it.

Reimplemented from Iterator.

References SurrBasedMinimizer::minimize_surrogates().

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

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

Reimplemented from Iterator.

References Dakota::abort_handler(), Iterator::activeSet, Minimizer::archive_allocate_best(), Minimizer::archive_best(), Iterator::bestResponseArray, Iterator::bestVariablesArray, Dakota::data_pairs, Model::interface_id(), Iterator::iteratedModel, Dakota::lookup_by_val(), Iterator::methodName, Minimizer::numFunctions, Minimizer::numUserPrimaryFns, Minimizer::optimizationFlag, ActiveSet::request_values(), and Model::truth_model().
void update_lagrange_multipliers ( const RealVector & fn_vals, const RealMatrix & fn_grads ) [protected]
initialize and update Lagrange multipliers for basic Lagrangian
For the Rockafellar augmented Lagrangian, simple Lagrange multiplier updates are available which do not require the active constraint gradients. For the basic Lagrangian, Lagrange multipliers are estimated through solution of a nonnegative linear least squares problem.
References Dakota::abort_handler(), Minimizer::bigRealBoundSize, Minimizer::constraintTol, Iterator::iteratedModel, SurrBasedMinimizer::lagrangeMult, Minimizer::numContinuousVars, Minimizer::numNonlinearEqConstraints, Minimizer::numNonlinearIneqConstraints, Minimizer::numUserPrimaryFns, Minimizer::objective_gradient(), SurrBasedMinimizer::origNonlinIneqLowerBnds, SurrBasedMinimizer::origNonlinIneqUpperBnds, Model::primary_response_fn_sense(), and Model::primary_response_fn_weights().
Referenced by SurrBasedLocalMinimizer::hard_convergence_check().

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.
References SurrBasedMinimizer::augLagrangeMult, SurrBasedMinimizer::betaEta, Minimizer::bigRealBoundSize, SurrBasedMinimizer::etaSequence, Minimizer::numNonlinearEqConstraints, Minimizer::numNonlinearIneqConstraints, Minimizer::numUserPrimaryFns, SurrBasedMinimizer::origNonlinEqTargets, SurrBasedMinimizer::origNonlinIneqLowerBnds, SurrBasedMinimizer::origNonlinIneqUpperBnds, and SurrBasedMinimizer::penalty_Parameter.
Referenced by SurrBasedLocalMinimizer::hard_convergence_check(), EffGlobalMinimizer::minimize_surrogates_on_model(), and SurrBasedLocalMinimizer::tr_ratio_check().

bool update_filter ( const RealVector & fn_vals ) [protected]
update a filter from a set of function values
Update the sbFilter with fn_vals if new iterate is non-dominated.
References SurrBasedMinimizer::constraint_violation(), Iterator::iteratedModel, Minimizer::numNonlinearConstraints, Minimizer::objective(), Model::primary_response_fn_sense(), Model::primary_response_fn_weights(), and SurrBasedMinimizer::sbFilter.
Referenced by SurrBasedLocalMinimizer::hard_convergence_check(), and SurrBasedLocalMinimizer::tr_ratio_check().

Real lagrangian_merit ( const RealVector & fn_vals, const BoolDeque & sense, const RealVector & primary_wts, const RealVector & nln_jineq_l_bnds, const RealVector & nln_jineq_u_bnds, const RealVector & nln_eq_tgts ) [protected]
compute a Lagrangian function from a set of function values
The Lagrangian function computation sums the objective function and the Lagrange multiplier terms for inequality/equality constraints. This implementation follows the convention in Vanderplaats with $g \leq 0$ and $h = 0$. The bounds/targets passed in may reflect the original constraints or the relaxed constraints.
References Minimizer::bigRealBoundSize, Minimizer::constraintTol, SurrBasedMinimizer::lagrangeMult, Minimizer::numNonlinearEqConstraints, Minimizer::numNonlinearIneqConstraints, Minimizer::numUserPrimaryFns, and Minimizer::objective().
Referenced by SurrBasedLocalMinimizer::approx_subprob_objective_eval(), and SurrBasedLocalMinimizer::tr_ratio_check().
Real augmented_lagrangian_merit (const RealVector & fn_vals, const BoolDeque & sense, const RealVector & primary_wts, const RealVector & nln_ineq_l_bnds, const RealVector & nln_ineq_u_bnds, const RealVector & nln_eq_tgts) [protected]

compute an augmented Lagrangian function from a set of function values

The Rockafellar augmented Lagrangian function sums the objective function, Lagrange multiplier terms for inequality/equality constraints, and quadratic penalty terms for inequality/equality constraints. This implementation follows the convention in Vanderplaats with $g < 0$ and $h = 0$. The bounds/targets passed in may reflect the original constraints or the relaxed constraints.

References SurrBasedMinimizer::augLagrangeMult, Minimizer::bigRealBoundSize, Minimizer::numNonlinearEqConstraints, Minimizer::numNonlinearIneqConstraints, Minimizer::numUserPrimaryFns, Minimizer::objective(), and SurrBasedMinimizer::penaltyParameter.

Referenced by SurrBasedLocalMinimizer::approx_subprob_objective_eval(), EffGlobalMinimizer::get_best_-sample(), EffGlobalMinimizer::minimize_surrogates_on_model(), and SurrBasedLocalMinimizer::tr_ratio_check().

Real penalty_merit (const RealVector & fn_vals, const BoolDeque & sense, const RealVector & primary_wts) [protected]

compute a penalty function from a set of function values

The penalty function computation applies a quadratic penalty to any constraint violations and adds this to the objective function(s) $p = f + rp cv$.

References SurrBasedMinimizer::constraintViolation(), Minimizer::constraintTol, Minimizer::objective(), and SurrBasedMinimizer::penaltyParameter.

Referenced by SurrBasedLocalMinimizer::tr_ratio_check().

Real constraintViolation (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 $cv = g^+ h^+ T g^+ h^+$. This implementation supports equality constraints and 2-sided inequalities. The constraint_tol allows for a small constraint infeasibility (used for penalty methods, but not Lagrangian methods).

References Minimizer::bigRealBoundSize, Minimizer::numNonlinearEqConstraints, Minimizer::numNonlinearIneqConstraints, Minimizer::numUserPrimaryFns, SurrBasedMinimizer::origNonlinEqTargets, SurrBasedMinimizer::origNonlinIneqLowerBnds, and SurrBasedMinimizer::origNonlinIneqUpperBnds.

Referenced by SurrBasedLocalMinimizer::hard_convergence_check(), EffGlobalMinimizer::minimize_surrogates_on_model(), SurrBasedMinimizer::penalty_merit(), SurrBasedLocalMinimizer::relax_constraints(), SurrBasedLocalMinimizer::tr_ratio_check(), SurrBasedMinimizer::update_filter(), and SurrBasedLocalMinimizer::update_penalty().

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

- SurrBasedMinimizer.hpp
- SurrBasedMinimizer.cpp

13.160 SurrogateModel Class Reference

Base class for surrogate models (DataFitSurrModel and HierarchSurrModel). Inheritance diagram for SurrogateModel:
Protected Member Functions

- **SurrogateModel** (ProblemDescDB &problem_db)
  - constructor
- **SurrogateModel** (ParallelLibrary &parallel_lib, const SharedVariablesData &svd, const ActiveSet &set, short output_level)
  - alternate constructor
- **~SurrogateModel** ()
  - destructor
- **Model & subordinate_model** ()
  - return truth_model()
- **short surrogate_response_mode** () const
  - return responseMode
- **DiscrepancyCorrection & discrepancy_correction** ()
  - return deltaCorr
- **void check_submodel_compatibility** (const Model &sub_model)
  - verify compatibility between SurrogateModel attributes and attributes of the submodel (DataFitSurrModel::actualModel or HierarchSurrModel::highFidelityModel)
- **bool force_rebuild** ()
  - evaluate whether a rebuild of the approximation should be forced based on changes in the inactive data
- **void asv_mapping** (const ShortArray &orig_asv, ShortArray &actual_asv, ShortArray &approx_asv, bool build_flag)
  - distributes the incoming orig_asv among actual_asv and approx_asv
- **void asv_mapping** (const ShortArray &actual_asv, const ShortArray &approx_asv, ShortArray &combined_asv)
  - reconstitutes a combined_asv from actual_asv and approx_asv
- **void response_mapping** (const Response &actual_response, const Response &approx_response, Response &combined_response)
  - overlays actual_response and approx_response to update combined_response

Protected Attributes

- **IntSet surrogateFnIndices**
  - for mixed response sets, this array specifies the response function subset that is approximated
- **IntResponseMap surrResponseMap**
  - map of surrogate responses used in derived_synchronize() and derived_synchronize_nowait() functions
- **IntVariablesMap rawVarsMap**
map of raw continuous variables used by apply_correction(). Model::varsList cannot be used for this purpose since it does not contain lower level variables sets from finite differencing.

- **IntIntMap truthIdMap**
  map from actualModel/highFidelityModel evaluation ids to DataFitSurrModel/hierarchSurrModel ids

- **IntIntMap surrIdMap**
  map from approxInterface/lowFidelityModel evaluation ids to DataFitSurrModel/hierarchSurrModel ids

- **IntResponseMap cachedApproxRespMap**
  map of approximate responses retrieved in derived_synchronize_nowait() that could not be returned since corresponding truth model response portions were still pending.

- **short responseMode**
  an enumeration that controls the response calculation mode in \{DataFit,Hierarch\}SurrModel approximate response computations

- **size_t approxBuilds**
  number of calls to build_approximation()

- **RealVector referenceCLBnds**
  stores a reference copy of active continuous lower bounds when the approximation is built; used to detect when a rebuild is required.

- **RealVector referenceCUBnds**
  stores a reference copy of active continuous upper bounds when the approximation is built; used to detect when a rebuild is required.

- **IntVector referenceDILBnds**
  stores a reference copy of active discrete int lower bounds when the approximation is built; used to detect when a rebuild is required.

- **IntVector referenceDIUBnds**
  stores a reference copy of active discrete int upper bounds when the approximation is built; used to detect when a rebuild is required.

- **RealVector referenceDRLBnds**
  stores a reference copy of active discrete real lower bounds when the approximation is built; used to detect when a rebuild is required.

- **RealVector referenceDRUBnds**
  stores a reference copy of active discrete real upper bounds when the approximation is built; used to detect when a rebuild is required.

- **RealVector referenceICVars**
  stores a reference copy of the inactive continuous variables when the approximation is built using a Distinct view; used to detect when a rebuild is required.

- **IntVector referenceIDIVars**
  stores a reference copy of the inactive discrete int variables when the approximation is built using a Distinct view; used to detect when a rebuild is required.

- **RealVector referenceIDRVars**
  stores a reference copy of the inactive discrete real variables when the approximation is built using a Distinct view; used to detect when a rebuild is required.

- **DiscrepancyCorrection deltaCorr**
  manages construction and application of correction functions that are applied to a surrogate model (DataFitSurr or HierarchSurr) in order to reproduce high fidelity data.
Private Attributes

- **Variables truthModelVars**
  
  copy of the truth model variables object used to simplify conversion among differing variable views in `force_rebuild`

- **Constraints truthModelCons**
  
  copy of the truth model constraints object used to simplify conversion among differing variable views in `force_rebuild`

Additional Inherited Members

13.160.1 Detailed Description

Base class for surrogate models (DataFitSurrModel and HierarchSurrModel).

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

13.160.2 Member Function Documentation

```cpp
bool force_rebuild() [protected], [virtual]
```
evaluate whether a rebuild of the approximation should be forced based on changes in the inactive data.

This function forces a rebuild of the approximation according to the sub-model variables view, the approximation type, and whether the active approximation bounds or inactive variable values have changed since the last approximation build.

Reimplemented from Model.

References Constraints::all_continuous_lower_bounds(), Constraints::all_continuous_upper_bounds(), Variables::all_continuous_variables(), Constraints::all_discrete_int_lower_bounds(), Constraints::all_discrete_int_upper_bounds(), Variables::all_discrete_int_variables(), Constraints::all_discrete_real_lower_bounds(), Constraints::all_discrete_real_upper_bounds(), Variables::all_discrete_real_variables(), Constraints::continuous_lower_bounds(), Model::continuous_lower_bounds(), Constraints::continuous_upper_bounds(), Model::continuous_upper_bounds(), Variables::continuous_variables(), Constraints::copy(), Variables::copy(), Model::current_variables(), Model::currentVariables, Constraints::discrete_int_lower_bounds(), Model::discrete_int_lower_bounds(), Constraints::discrete_int_upper_bounds(), Model::discrete_int_upper_bounds(), Variables::discrete_int_variables(), Constraints::discrete_real_lower_bounds(), Model::discrete_real_lower_bounds(), Variables::discrete_real_variables(), Variables::inactive_continuous_variables(), Variables::inactive_discrete_int_variables(), Variables::inactive_discrete_real_variables(), Constraints::is_null(), Variables::is_null(), Model::is_null(), Model::model_type(), SurrogateModel::referenceCLBnds, SurrogateModel::referenceCUBnds, SurrogateModel::referenceDILBnds, SurrogateModel::referenceDIUBnds, SurrogateModel::referenceDRLBnds, SurrogateModel::referenceDRUBnds, SurrogateModel::referenceICVars, SurrogateModel::referenceIDIVars, SurrogateModel::referenceIDRVars, Dakota::strbegins(), Model::subordinate_model(), Model::surrogateType, Model::truth_model(), SurrogateModel::truthModelCons, SurrogateModel::truthModelVars, Model::user_defined_constraints(), Model::userDefinedConstraints, and Variables::view().

Referenced by HierarchSurrModel::derived_asynch_compute_response(), DataFitSurrModel::derived_asynch_compute_response(), HierarchSurrModel::derived_comput_response(), and DataFitSurrModel::derived_comput_response().
13.160.3 Member Data Documentation

short responseMode  [protected]

an enumeration that controls the response calculation mode in {DataFit,Hierarch}SurrModel approximate response computations

SurrBasedLocalMinimizer toggles this mode since compute_correction() does not back out old corrections.

Referenced by HierarchSurrModel::derived_asynch_compute_response(), DataFitSurrModel::derived_asynch_compute_response(), DataFitSurrModel::derived_compute_response(), HierarchSurrModel::derived_compute_response(), DataFitSurrModel::derived_set_communicators(), HierarchSurrModel::derived_synchronize(), DataFitSurrModel::derived_synchronize(), HierarchSurrModel::derived_synchronize_nowait(), DataFitSurrModel::derived_synchronize_nowait(), HierarchSurrModel::serve(), SurrogateModel::surrogate_response_mode(), HierarchSurrModel::surrogate_response_mode(), and DataFitSurrModel::surrogate_response_mode().

size_t approxBuilds  [protected]

number of calls to build_approximation()

used as a flag to automatically build the approximation if one of the derived compute_response functions is called prior to build_approximation().

Referenced by DataFitSurrModel::append_approximation(), DataFitSurrModel::approximation_coefficients(), HierarchSurrModel::build_approximation(), DataFitSurrModel::build_approximation(), HierarchSurrModel::derived_asynch_compute_response(), DataFitSurrModel::derived_asynch_compute_response(), HierarchSurrModel::derived_compute_response(), DataFitSurrModel::derived_compute_response(), DataFitSurrModel::pop_approximation(), and DataFitSurrModel::update_approximation().

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

- SurrogateModel.hpp
- SurrogateModel.cpp

13.161 SysCallApplicInterface Class Reference

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

Inheritance diagram for SysCallApplicInterface:
Public Member Functions

- **SysCallApplicInterface** (const ProblemDescDB &problem_db)
  
  *Constructor*

- **~SysCallApplicInterface** ()
  
  *Destructor*

Protected Member Functions

- **void wait_local_evaluations** (PRPQueue &prp_queue)
- **void test_local_evaluations** (PRPQueue &prp_queue)
- **int synchronous_local_analysis** (int analysis_id)
- **void init_communicators_checks** (int max_eval_concurrency)
- **void set_communicators_checks** (int max_eval_concurrency)
- **void map_bookkeeping** (pid_t pid, int fn_eval_id)

  *Bookkeeping of process and evaluation ids for asynchronous maps*

- **pid_t create_evaluation_process** (bool block_flag)

  *Spawn the evaluation by managing the input filter, analysis drivers, and output filter. Called from derived_map() & derived_map_asynch().*

Private Member Functions

- **bool system_call_file_test** (const std::string &root_file)

  *Detect completion of a function evaluation through existence of the necessary results file(s)*

- **void spawn_evaluation_to_shell** (bool block_flag)

  *Spawn a complete function evaluation*

- **void spawn_input_filter_to_shell** (bool block_flag)

  *Spawn the input filter portion of a function evaluation*

- **void spawn_analysis_to_shell** (int analysis_id, bool block_flag)

  *Spawn a single analysis as part of a function evaluation*

- **void spawn_output_filter_to_shell** (bool block_flag)

  *Spawn the output filter portion of a function evaluation*

Private Attributes

- **IntSet sysCallSet**

  *Set of function evaluation id's for active asynchronous system call evaluations*

- **IntShortMap failCountMap**

  *Map linking function evaluation id's to number of response read failures*

Additional Inherited Members

13.161.1 Detailed Description

Derived application interface class which spawns simulation codes using system calls. system() is part of the C API and can be used on both Windows and Unix systems.
13.161.2 Member Function Documentation

**void wait_local_evaluations ( PRPQueue & prp_queue )** [inline], [protected], [virtual]

Check for completion of active asynch jobs (tracked with sysCallSet). Wait for at least one completion and complete all jobs that have returned. This satisfies a "fairness" principle, in the sense that a completed job will *always* be processed (whereas accepting only a single completion could always accept the same completion - the case of very inexpensive fn. evals. - and starve some servers).

- Reimplemented from ApplicationInterface.
- References ApplicationInterface::completionSet, and SysCallApplicInterface::test_local_evaluations().

**void test_local_evaluations ( PRPQueue & prp_queue )** [protected], [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.
- References Dakota::abort_handler(), Response::active_set(), ApplicationInterface::completionSet, SysCallApplicInterface::failCountMap, ProcessApplicInterface::fileNameMap, Interface::final_eval_id_tag(), Dakota::lookup-by_eval_id(), ApplicationInterface::manage_failure(), ProcessApplicInterface::read_results_files(), SysCallApplicInterface::sysCallSet, and SysCallApplicInterface::system_call_file_test().

  Referenced by SysCallApplicInterface::wait_local_evaluations().

**int synchronous_local_analysis ( int analysis_id )** [inline], [protected], [virtual]

This code provides the derived function used by ApplicationInterface::serve_analyses_synch().

- Reimplemented from ApplicationInterface.
- References SysCallApplicInterface::spawn_analysis_to_shell().

**void init_communicators_checks ( int max_eval_concurrency )** [inline], [protected], [virtual]

No derived interface plug-ins, so perform construct-time checks. However, process init issues as warnings since some contexts (e.g., HierarchSurrModel) initialize more configurations than will be used.

- Reimplemented from ApplicationInterface.
- References ApplicationInterface::check_multiprocessor_analysis().

**void set_communicators_checks ( int max_eval_concurrency )** [inline], [protected], [virtual]

Process run-time issues as hard errors.

- Reimplemented from ApplicationInterface.
- References Dakota::abort_handler(), and ApplicationInterface::check_multiprocessor_analysis().

**void spawn_evaluation_to_shell ( bool block_flag )** [private]

- spawn a complete function evaluation

  Put the SysCallApplicInterface to the shell. This function is used when all portions of the function evaluation (i.e., all analysis drivers) are executed on the local processor.

- Reimplemented from ApplicationInterface.
- References CommandShell::asynch_flag(), ProcessApplicInterface::commandLineArgs, ProcessApplicInterface::curWorkdir, Dakota::flush(), ProcessApplicInterface::iFilterName, ProcessApplicInterface::multipleParamsFiles, ProcessApplicInterface::oFilterName, ProcessApplicInterface::paramsFileName, ProcessApplicInterface::programNames, ProcessApplicInterface::resultsFileName, CommandShell::suppress_output_flag(), ApplicationInterface::suppressOutput, and ProcessApplicInterface::useWorkdir.

  Referenced by SysCallApplicInterface::create_evaluation_process().
void spawn_input_filter_to_shell ( bool block_flag ) [private]

spawn the input filter portion of a function evaluation

Put the input filter to the shell. This function is used when multiple analysis drivers are spread between processors. No need to check for a Null input filter, as this is checked externally. Use of nonblocking shells is supported in this fn, although its use is currently prevented externally.

References CommandShell::asynch_flag(), ProcessApplicInterface::commandLineArgs, ProcessApplicInterface::curWorkdir, Dakota::flush(), ProcessApplicInterface::iFilterName, ProcessApplicInterface::paramsFileName, ProcessApplicInterface::resultsFileName, CommandShell::suppress_output_flag(), ApplicationInterface::suppressOutput, and ProcessApplicInterface::useWorkdir.

Referenced by SysCallApplicInterface::create_evaluation_process().

void spawn_analysis_to_shell ( int analysis_id, bool block_flag ) [private]

spawn a single analysis as part of a function evaluation

Put a single analysis to the shell. This function is used when multiple analysis drivers are spread between processors. Use of nonblocking shells is supported in this fn, although its use is currently prevented externally.

References CommandShell::asynch_flag(), ProcessApplicInterface::commandLineArgs, ProcessApplicInterface::curWorkdir, Dakota::flush(), ProcessApplicInterface::multipleParamsFiles, ProcessApplicInterface::paramsFileName, ProcessApplicInterface::programNames, ProcessApplicInterface::resultsFileName, CommandShell::suppress_output_flag(), ApplicationInterface::suppressOutput, and ProcessApplicInterface::useWorkdir.

Referenced by SysCallApplicInterface::create_evaluation_process(), SysCallApplicInterface::synchronous_local_analysis(), and GridApplicInterface::synchronous_local_analysis().

void spawn_output_filter_to_shell ( bool block_flag ) [private]

spawn the output filter portion of a function evaluation

Put the output filter to the shell. This function is used when multiple analysis drivers are spread between processors. No need to check for a Null output filter, as this is checked externally. Use of nonblocking shells is supported in this fn, although its use is currently prevented externally.

References CommandShell::asynch_flag(), ProcessApplicInterface::commandLineArgs, ProcessApplicInterface::curWorkdir, Dakota::flush(), ProcessApplicInterface::oFilterName, ProcessApplicInterface::paramsFileName, ProcessApplicInterface::resultsFileName, CommandShell::suppress_output_flag(), ApplicationInterface::suppressOutput, and ProcessApplicInterface::useWorkdir.

Referenced by SysCallApplicInterface::create_evaluation_process().

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

- SysCallApplicInterface.hpp
- SysCallApplicInterface.cpp

13.162 TANA3Approximation Class Reference

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

Inheritance diagram for TANA3Approximation:

```
Approximation
   `-- TANA3Approximation
```
CHAPTER 13. CLASS DOCUMENTATION

Public Member Functions

- **TANA3Approximation ()**
  default constructor
- **TANA3Approximation (ProblemDescDB &problem_db, const SharedApproxData &shared_data)**
  standard constructor
- **TANA3Approximation (const SharedApproxData &shared_data)**
  alternate constructor
- **~TANA3Approximation ()**
  destructor

Protected Member Functions

- **int min_coefficients () const**
  return the minimum number of samples (unknowns) required to build the derived class approximation type in num-Vars dimensions
- **int num_constraints () const**
  return the number of constraints to be enforced via an anchor point
- **void build ()**
  builds the approximation from scratch
- **Real value (const Variables &vars)**
  retrieve the approximate function value for a given parameter vector
- **const RealVector & gradient (const Variables &vars)**
  retrieve the approximate function gradient for a given parameter vector
- **void clear_current ()**

Private Member Functions

- **void find_scaled_coefficients ()**
  compute TANA coefficients based on scaled inputs
- **void offset (const RealVector &x, RealVector &s)**
  based on minX, apply offset scaling to x to define s

Private Attributes

- **RealVector pExp**
  vector of exponent values
- **RealVector minX**
  vector of minimum parameter values used in scaling
- **RealVector scX1**
  vector of scaled x1 values
- **RealVector scX2**
  vector of scaled x2 values
- **Real H**
  the scalar Hessian value in the TANA-3 approximation
Additional Inherited Members

13.162.1 Detailed Description

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

The TANA3Approximation class provides a multipoint approximation based on matching value and gradient data from two points (typically the current and previous iterates) in parameter space. It forms an exponential approximation in terms of intervening variables.

13.162.2 Member Function Documentation

void build() [protected], [virtual]

builds the approximation from scratch

This is the common base class portion of the virtual fn and is insufficient on its own; derived implementations should explicitly invoke (or reimplement) this base class contribution.

Reimplemented from Approximation.

References Dakota::abort_handler(), Approximation::approxData, Approximation::build(), TANA3Approximation::find_scaled_coefficients(), TANA3Approximation::minX, SharedApproxData::numVars, TANA3Approximation::pExp, and Approximation::sharedDataRep.

void clear_current() [inline], [protected], [virtual]

Redefine default implementation to support history mechanism.

Reimplemented from Approximation.

References Approximation::approxData.

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

- TANA3Approximation.hpp
- TANA3Approximation.cpp

13.163 TaylorApproximation Class Reference

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

Inheritance diagram for TaylorApproximation:

```
Approximation
```

```
TaylorApproximation
```

Public Member Functions

- TaylorApproximation()
  
  *default constructor*

- TaylorApproximation(const ProblemDescDB &problem_db, const SharedApproxData &shared_data)
  
  *standard constructor*

- TaylorApproximation(const SharedApproxData &shared_data)
alternate constructor

- \~\text{TaylorApproximation}()

destructor

Protected Member Functions

- \text{int min\_coefficients}() \text{const}
  
  return the minimum number of samples (unknowns) required to build the derived class approximation type in num-Vars dimensions
- \text{void build}()
  
  builds the approximation from scratch
- \text{Real value}(const Variables &vars)
  
  retrieve the approximate function value for a given parameter vector
- \text{const RealVector & gradient} (const Variables &vars)
  
  retrieve the approximate function gradient for a given parameter vector
- \text{const RealSymMatrix & hessian} (const Variables &vars)
  
  retrieve the approximate function Hessian for a given parameter vector

Additional Inherited Members

13.163.1 Detailed Description

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

The TaylorApproximation class provides a local approximation based on data from a single point in parameter space. It uses a zeroth-, first- or second-order Taylor series expansion: \( f(x) = f(x_c) \) for zeroth-order, plus \( \text{grad}(x-x_c)'(x-x_c) \) for first- and second-order, and plus \( (x-x_c)'\text{Hess}(x_c)(x-x_c)/2 \) for second-order.

13.163.2 Member Function Documentation

\text{void build}() [protected], [virtual]

builds the approximation from scratch

This is the common base class portion of the virtual fn and is insufficient on its own; derived implementations should explicitly invoke (or reimplement) this base class contribution.

Reimplemented from Approximation.

References Dakota::abort\_handler(), Approximation::approxData, Approximation::\text{build}(), SharedApproxData::buildDataOrder, SharedApproxData::\text{numVars}, and Approximation::\text{sharedDataRep}.

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

- TaylorApproximation.hpp
- TaylorApproximation.cpp

13.164 TestDriverInterface Class Reference

Inheritance diagram for TestDriverInterface:
Public Member Functions

- **TestDriverInterface** (const ProblemDescDB &problem_db)
  
  *constructor*
- **~TestDriverInterface()**
  
  *destructor*

Protected Member Functions

- virtual int **derived_map_ac** (const Dakota::String &ac_name)
  
  *execute an analysis code portion of a direct evaluation invocation*

Private Member Functions

- int **cantilever()**
  
  *scaled cantilever test function for optimization*
- int **mod_cantilever()**
  
  *unscaled cantilever test function for UQ*
- int **cyl_head()**
  
  *the cylinder head constrained optimization test fn*
- int **multimodal()**
  
  *multimodal UQ test function*
- int **log_ratio()**
  
  *the log_ratio UQ test function*
- int **short_column()**
  
  *the short_column UQ/OUU test function*
- int **lf_short_column()**
  
  *a low fidelity short_column test function*
- int **mf_short_column()**
  
  *alternate short_column formulations for < multifidelity or model form studies*
- int **alternate_short_column_forms** (int form)
  
  *helper fn for alternate forms*
- int **side_impact_cost()**
  
  *the side_impact_cost UQ/OUU test function*
• int side_impact_perf ()
  the side_impact_perf UQ/OUU test function
• int rosenbrock ()
  the Rosenbrock optimization and least squares test fn
• int generalized_rosenbrock ()
  n-dimensional Rosenbrock (Schittkowski)
• int extended_rosenbrock ()
  n-dimensional Rosenbrock (Nocedal/Wright)
• int lf_rosenbrock ()
  a low fidelity version of the Rosenbrock function
• int mf_rosenbrock ()
  alternate Rosenbrock formulations for multifidelity or model form studies
• int gerstner ()
  the isotropic/anisotropic Gerstner test function family
• int scalable_gerstner ()
  scalable versions of the Gerstner test family
• void get_genz_coefficients (int num_dims, Real factor, int c_type, RealVector &c, RealVector &w)
  define coefficients needs for genz model
• int genz ()
  scalable test functions from the Genz test suite
• int steel_column_cost ()
  the steel_column_cost UQ/OUU test function
• int steel_column_perf ()
  the steel_column_perf UQ/OUU test function
• int sobol_rational ()
  Sobol SA rational test function.
• int sobol_g_function ()
  Sobol SA discontinuous test function.
• int sobol_ishigami ()
  Sobol SA transcendental test function.
• int text_book ()
  the text_book constrained optimization test function
• int text_book1 ()
  portion of text_book() evaluating the objective fn
• int text_book2 ()
  portion of text_book() evaluating constraint 1
• int text_book3 ()
  portion of text_book() evaluating constraint 2
• int text_book_ouu ()
  the text_book_ouu OUU test function
• int scalable_text_book ()
  scalable version of the text_book test function
• int scalable_monomials ()
simple monomials for UQ exactness testing

- void herbie1D (size_t der_mode, Real xc_loc, std::vector<Real> &w_and_ders)
  ID components of herbie function
- void smooth_herbie1D (size_t der_mode, Real xc_loc, std::vector<Real> &w_and_ders)
  ID components of smooth_herbie function
- void shubert1D (size_t der_mode, Real xc_loc, std::vector<Real> &w_and_ders)
  ID components of shubert function

- int herbie ()
  returns the N-D herbie function
- int smooth_herbie ()
  returns the N-D smooth herbie function
- int shubert ()
  returns the N-D shubert function
- void separable_combine (Real mult_scale_factor, std::vector<Real> &w, std::vector<Real> &d1w, std::vector<Real> &d2w)
  utility to combine components of separable fns
- int salinas ()
  direct interface to the SALINAS structural dynamics code
- int mc_api_run ()
  direct interface to ModelCenter via API, HKIM 4/3/03

### Additional Inherited Members

#### 13.164.1 Detailed Description

Specialization of DirectApplicInterface to embed algebraic test function drivers directly in Dakota

#### 13.164.2 Member Function Documentation

`int derived_map_ac (const Dakota::String &ac_name) [protected], [virtual]`

execute an analysis code portion of a direct evaluation invocation

Derived map to evaluate a particular built-in test analysis function
Reimplemented from DirectApplicInterface.

References Dakota::abort_handler(), ApplicationInterface::analysisServerId, TestDriverInterface::cantilever(), TestDriverInterface::cyl_head(), DirectApplicInterface::driverTypeMap, TestDriverInterface::extended_rosenbrock(), TestDriverInterface::generalized_rosenbrock(), TestDriverInterface::genz(), TestDriverInterface::gerstner(), TestDriverInterface::herbie(), TestDriverInterface::lf_rosenbrock(), TestDriverInterface::lf_short_column(), TestDriverInterface::log_ratio(), TestDriverInterface::mc_api_run(), TestDriverInterface::mf_rosenbrock(), TestDriverInterface::mf_short_column(), TestDriverInterface::mod_cantilever(), TestDriverInterface::multimodal(), TestDriverInterface::rosenbrock(), TestDriverInterface::salinas(), TestDriverInterface::scalable_gerstner(), TestDriverInterface::scalable_monomials(), TestDriverInterface::scalable_text_book(), TestDriverInterface::short_column(), TestDriverInterface::shubert(), TestDriverInterface::side_impact_cost(), TestDriverInterface::side_impact_perf(), TestDriverInterface::smooth_herbie(), TestDriverInterface::sobol_g_function(), TestDriverInterface::sobol_ishigami(), TestDriverInterface::sobol_rational(), TestDriverInterface::steel_column_cost(), TestDriverInterface::steel_column_perf(), TestDriverInterface::text_book(), TestDriverInterface::text_book1(), TestDriverInterface::text_book2(), TestDriverInterface::text_book3(), and TestDriverInterface::text_book_ouu().
void herbie1D ( size_t der_mode, Real xc_loc, std::vector< Real >& w and ders ) [private]

1D components of herbie function
   1D Herbie function and its derivatives (apart from a multiplicative factor)
   Referenced by TestDriverInterface::herbie().

void smooth_herbie1D ( size_t der_mode, Real xc_loc, std::vector< Real >& w and ders ) [private]

1D components of smooth_herbie function
   1D Smoothed Herbie= 1DHerbie minus the high frequency sine term, and its derivatives (apart from a multiplicative factor)
   Referenced by TestDriverInterface::smooth_herbie().

void shubert1D ( size_t der_mode, Real xc_loc, std::vector< Real >& w and ders ) [private]

1D components of shubert function
   1D Shubert function and its derivatives (apart from a multiplicative factor)
   Referenced by TestDriverInterface::shubert().

int herbie ( ) [private]

returns the N-D herbie function
   N-D Herbie function and its derivatives.
   References DirectApplicInterface::directFnASV, DirectApplicInterface::directFnDVV, TestDriverInterface::herbie1D(), DirectApplicInterface::numDerivVars, DirectApplicInterface::numVars, TestDriverInterface::separable_combine(), and DirectApplicInterface::xC.
   Referenced by TestDriverInterface::derived_map_ac().

int smooth_herbie ( ) [private]

returns the N-D smooth herbie function
   N-D Smoothed Herbie function and its derivatives.
   References DirectApplicInterface::directFnASV, DirectApplicInterface::directFnDVV, DirectApplicInterface::numDerivVars, DirectApplicInterface::numVars, TestDriverInterface::separable_combine(), TestDriverInterface::smooth_herbie1D(), and DirectApplicInterface::xC.
   Referenced by TestDriverInterface::derived_map_ac().

void separable_combine ( Real mult_scale_factor, std::vector< Real > & w, std::vector< Real > & d1w, std::vector< Real > & d2w ) [private]

utility to combine components of separable fns
   this function combines N 1D functions and their derivatives to compute a N-D separable function and its derivatives, logic is general enough to support different 1D functions in different dimensions (can mix and match)
   References DirectApplicInterface::directFnASV, DirectApplicInterface::directFnDVV, DirectApplicInterface::fnGrads, DirectApplicInterface::fnHessians, DirectApplicInterface::fnVals, DirectApplicInterface::numDerivVars, and DirectApplicInterface::numVars.
   Referenced by TestDriverInterface::herbie(), TestDriverInterface::shubert(), and TestDriverInterface::smooth_herbie().
13.165 TrackerHTTP Class Reference

TrackerHTTP: a usage tracking module that uses HTTP/HTTPS via the curl library.

Public Member Functions

- TrackerHTTP (int world_rank=0)
  standard constructor with ProblemDescDB, rank
- ~TrackerHTTP ()
  destructor to free handles
- void post_start (ProblemDescDB &problem_db)
  post the start of an analysis and archive start time
- void post_finish (unsigned runtime=0)
  post the completion of an analysis including elapsed time

Private Member Functions

- void initialize (int world_rank=0)
  shared initialization functions across constructors
- void url_add_field (std::string &url, const char *keyword, const std::string &value, bool delimit=true) const
  append keyword/value pair to url in GET style (with &keyword=value); set delimit = false to omit the &
- void build_default_data (std::string &url, std::time_t &rawtime, const std::string &mode) const
  construct URL with shared information for start/finish
- void send_data_using_get (const std::string &urltopost) const
  transmit data to the web server using GET
- void send_data_using_post (const std::string &datatopost) const
  POST separate location and query; datatopost="name=daniel&project=curl."
- void populate_method_list (ProblemDescDB &problem_db)
  extract list of methods from problem database
• std::string get_uid() const
  get the real user ID
• std::string get_username() const
  get the username as reported by the environment
• std::string get_hostname() const
  get the system hostname
• std::string get_os() const
  get the operating system
• std::string get_datetime(const std::time_t &rawtime) const
  get the date and time as a string YYYYMMDDHHMMSS

Private Attributes

• CURL * curlPtr
  pointer to the curl handler instance
• FILE * devNull
  pointer to /dev/null
• std::string trackerLocation
  base URL for the tracker
• std::string proxyLocation
  if empty, proxy may still be specified via environment variables (unlike default CURL behavior)
• long timeoutSeconds
  seconds until the request will timeout (may have issues with signals)
• std::string methodList
  list of active methods
• std::string dakotaVersion
  DAKOTA version.
• std::time_t startTime
  cached starting time in raw seconds
• short outputLevel
  verbosity control

13.165.1 Detailed Description

TrackerHTTP: a usage tracking module that uses HTTP/HTTPS via the curl library.

13.165.2 Member Function Documentation

void send_data_using_get( const std::string &urltopost ) const [private]
transmit data to the web server using GET
whole url including location&fields
References TrackerHTTP::curlPtr, and TrackerHTTP::outputLevel.
void send_data_using_post ( const std::string & datatopost ) const  \[private\]

POST separate location and query; datatopost="name=daniel\&project=curl".
References TrackerHTTP::curlPtr, TrackerHTTP::outputLevel, and TrackerHTTP::trackerLocation.
Referenced by TrackerHTTP::post_finish(), and TrackerHTTP::post_start().
The documentation for this class was generated from the following files:

- TrackerHTTP.hpp
- TrackerHTTP.cpp

13.166 Variables Class Reference

Base class for the variables class hierarchy.

Inheritance diagram for Variables:

```
Variables
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MixedVariables RelaxedVariables</td>
</tr>
</tbody>
</table>
```

Public Member Functions

- Variables ()
  *default constructor*
- Variables (const ProblemDescDB &problem_db)
  *standard constructor*
- Variables (const SharedVariablesData &svd)
  *alternate constructor for instantiations on the fly*
- Variables (const Variables &vars)
  *copy constructor*
- virtual ~Variables ()
  *destructor*
- Variables operator= (const Variables &vars)
  *assignment operator*
- virtual void reshape (const SizetArray &vc_totals)
  *reshapes an existing Variables object based on the incoming variablesComponents*
- virtual void read (std::istream &s)
  *read a variables object from an std::istream*
- virtual void write (std::ostream &s) const
  *write a variables object to an std::ostream*
- virtual void write_aprepro (std::ostream &s) const
  *write a variables object to an std::ostream in aprepro format*
- virtual void read_annotated (std::istream &s)
read a variables object in annotated format from an istream

- virtual void write_annotated (std::ostream &s) const
  write a variables object in annotated format to an std::ostream

- virtual void read_tabular (std::istream &s)
  read a variables object in tabular format from an istream

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

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

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

- size_t tv () const
  total number of vars

- size_t cv () const
  number of active continuous vars

- size_t cv_start () const
  start index of active continuous vars

- size_t div () const
  number of active discrete int vars

- size_t div_start () const
  start index of active discrete int vars

- size_t drv () const
  number of active discrete real vars

- size_t drv_start () const
  start index of active discrete real vars

- size_t icv () const
  number of inactive continuous vars

- size_t icv_start () const
  start index of inactive continuous vars

- size_t idiv () const
  number of inactive discrete int vars

- size_t idiv_start () const
  start index of inactive discrete int vars

- size_t idrv () const
  number of inactive discrete real vars

- size_t idrv_start () const
  start index of inactive discrete real vars

- size_t acv () const
  total number of continuous vars

- size_t adiv () const
  total number of discrete integer vars

- size_t adrv () const
  total number of discrete real vars
• const SharedVariablesData & shared_data () const
  return sharedVarsData

• SharedVariablesData & shared_data ()
  return sharedVarsData

• Real continuous_variable (size_t index) const
  return an active continuous variable

• const RealVector & continuous_variables () const
  return the active continuous variables (Note: returns a view by const reference, but initializing a RealVector from
  this reference invokes the Teuchos matrix copy constructor to create a Teuchos::Copy instance; to obtain a mutable
  view, use continuous_variables_view())

• void continuous_variable (Real c_var, size_t index)
  set an active continuous variable

• void continuous_variables (const RealVector &c_vars)
  set the active continuous variables

• int discrete_int_variable (size_t index) const
  return an active discrete integer variable

• const IntVector & discrete_int_variables () const
  return the active discrete integer variables (Note: returns a view by const reference, but initializing an IntVector
  from this reference invokes the Teuchos matrix copy constructor to create a Teuchos::Copy instance; to obtain a
  mutable view, use discrete_int_variables_view())

• void discrete_int_variable (int di_var, size_t index)
  set an active discrete integer variable

• void discrete_int_variables (const IntVector &di_vars)
  set the active discrete integer variables

• Real discrete_real_variable (size_t index) const
  return an active discrete real variable

• const RealVector & discrete_real_variables () const
  return the active discrete real variables (Note: returns a view by const reference, but initializing a RealVector from
  this reference invokes the Teuchos matrix copy constructor to create a Teuchos::Copy instance; to obtain a mutable
  view, use discrete_real_variables_view())

• void discrete_real_variable (Real dr_var, size_t index)
  set an active discrete real variable

• void discrete_real_variables (const RealVector &dr_vars)
  set the active discrete real variables

• void active_variables (const Variables &vars)
  copy the active cv/div/drv variables from vars

• RealVector continuous_variables_view () const
  return a mutable view of the active continuous variables

• IntVector discrete_int_variables_view () const
  return a mutable view of the active discrete integer variables

• RealVector discrete_real_variables_view () const
  return a mutable view of 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, size_t index)
  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, size_t index)
  set an active discrete integer variable label
• StringMultiArrayConstView discrete_real_variable_labels () const
  return the active discrete real variable labels
• void discrete_real_variable_labels (StringMultiArrayConstView drv_labels)
  set the active discrete real variable labels
• void discrete_real_variable_label (const String &drv_label, size_t index)
  set an active discrete real variable label
• UShortMultiArrayConstView continuous_variable_types () const
  return the active continuous variable types
• void continuous_variable_types (UShortMultiArrayConstView cv_types)
  set the active continuous variable types
• void continuous_variable_type (unsigned short cv_type, size_t index)
  set an active continuous variable type
• UShortMultiArrayConstView discrete_int_variable_types () const
  return the active discrete integer variable types
• void discrete_int_variable_types (UShortMultiArrayConstView div_types)
  set the active discrete integer variable types
• void discrete_int_variable_type (unsigned short div_type, size_t index)
  set an active discrete integer variable type
• UShortMultiArrayConstView discrete_real_variable_types () const
  return the active discrete real variable types
• void discrete_real_variable_types (UShortMultiArrayConstView drv_types)
  set the active discrete real variable types
• void discrete_real_variable_type (unsigned short drv_type, size_t index)
  set an active discrete real variable type
• SizetMultiArrayConstView continuous_variable_ids () const
  return the active continuous variable position identifiers
• void continuous_variable_ids (SizetMultiArrayConstView cv_ids)
  set the active continuous variable position identifiers
• void continuous_variable_id (size_t cv_id, size_t index)
  set an active continuous variable position identifier
• const SizetArray & relaxed_discrete_ids () const
  returns the set of discrete variable ids relaxed into a continuous array
• const RealVector & inactive_continuous_variables () const
return the inactive continuous variables

- void inactive_continuous_variables (const RealVector &ic_vars)
  set the inactive continuous variables
- const IntVector & inactive_discrete_int_variables () const
  return the inactive discrete variables
- void inactive_discrete_int_variables (const IntVector &idi_vars)
  set the inactive discrete variables
- const RealVector & inactive_discrete_real_variables () const
  return the inactive discrete variables
- void inactive_discrete_real_variables (const RealVector &idr_vars)
  set the inactive discrete variables
- StringMultiArrayConstView inactive_continuous_variable_labels () const
  return the inactive continuous variable labels
- void inactive_continuous_variable_labels (StringMultiArrayConstView ic_vars)
  set the inactive continuous variable labels
- StringMultiArrayConstView inactive_discrete_int_variable_labels () const
  return the inactive discrete 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 real variable labels
- void inactive_discrete_real_variable_labels (StringMultiArrayConstView idr_vars)
  set the inactive discrete real variable labels
- UShortMultiArrayConstView inactive_continuous_variable_types () const
  return the inactive continuous variable types
- UShortMultiArrayConstView inactive_discrete_int_variable_types () const
  return the inactive discrete integer variable types
- UShortMultiArrayConstView inactive_discrete_real_variable_types () const
  return the inactive discrete real variable types
- SizetMultiArrayConstView inactive_continuous_variable_ids () const
  return the inactive continuous variable position identifiers
- const RealVector & all_continuous_variables () const
  returns a single array with all continuous variables
- void all_continuous_variables (const RealVector &ac_vars)
  sets all continuous variables using a single array
- void all_continuous_variable (Real ac_var, size_t index)
  set a variable within the all continuous array
- const IntVector & all_discrete_int_variables () const
  returns a single array with all discrete variables
- void all_discrete_int_variables (const IntVector &adi_vars)
  sets all discrete variables using a single array
- void all_discrete_int_variable (int adi_var, size_t index)
  set a variable within the all discrete array
- const RealVector & all_discrete_real_variables () const
  returns a single array with all discrete variables
- void all_discrete_real_variables (const RealVector &adr_vars)
  sets all discrete variables using a single array
- void all_discrete_real_variable (Real adr_var, size_t index)
  set a variable within the all discrete array
- StringMultiArrayView all_continuous_variable_labels () const
  returns a single array with all continuous variable labels
- void all_continuous_variable_labels (StringMultiArrayConstView acv_labels)
  sets all continuous variable labels using a single array
- void all_continuous_variable_label (const String &acv_label, size_t index)
  set a label within the all continuous label array
- StringMultiArrayView all_discrete_int_variable_labels () const
  returns a single array with all discrete variable labels
- void all_discrete_int_variable_labels (StringMultiArrayConstView adiv_labels)
  sets all discrete variable labels using a single array
- void all_discrete_int_variable_label (const String &adiv_label, size_t index)
  set a label within the all discrete label array
- StringMultiArrayView all_discrete_real_variable_labels () const
  returns a single array with all discrete variable labels
- void all_discrete_real_variable_labels (StringMultiArrayConstView adrv_labels)
  sets all discrete variable labels using a single array
- void all_discrete_real_variable_label (const String &adrv_label, size_t index)
  set a label within the all discrete label array
- UShortMultiArrayConstView all_continuous_variable_types () const
  return all continuous variable types
- UShortMultiArrayConstView all_discrete_int_variable_types () const
  return all discrete variable types
- UShortMultiArrayConstView all_discrete_real_variable_types () const
  return all discrete variable types
- SizetMultiArrayConstView all_continuous_variable_ids () const
  return all continuous variable position identifiers
- Variables copy (bool deep_svd=false) const
  for use when a deep copy is needed (the representation is not shared)
- const std::pair< short, short > & view () const
  returns variablesView
- std::pair< short, short > get_view (const ProblemDescDB &problem_db) const
  defines variablesView from problem_db attributes
- void inactive_view (short view2)
  sets the inactive view based on higher level (nested) context
- const String & variables_id () const
  returns the variables identifier string
- const SizetArray & variables_components_totals () const
returns the number of variables for each of the constitutive components

- bool is_null () const
  function to check variablesRep (does this envelope contain a letter)

Protected Member Functions

- Variables (BaseConstructor, const ProblemDescDB &problem_db, const std::pair<short, short> &view)
  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

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

- virtual void build_active_views ()
  construct active views of all variables arrays

- virtual void build_inactive_views ()
  construct inactive views of all variables arrays

- void build_views ()
  construct active/inactive views of all variables arrays

Protected Attributes

- SharedVariablesData sharedVarsData
  reference-counted instance of shared variables data: id’s, labels, counts

- RealVector allContinuousVars
  array combining all of the continuous variables (design, uncertain, state)

- IntVector allDiscreteIntVars
  array combining all of the discrete integer variables (design, state)

- RealVector allDiscreteRealVars
  array combining all of the discrete real variables (design, state)

- RealVector continuousVars
  the active continuous variables array view

- IntVector discreteIntVars
  the active discrete integer variables array view

- RealVector discreteRealVars
  the active discrete real variables array view

- RealVector inactiveContinuousVars
  the inactive continuous variables array view

- IntVector inactiveDiscreteIntVars
  the inactive discrete integer variables array view

- RealVector inactiveDiscreteRealVars
  the inactive discrete real variables array view
Private Member Functions

- Variables * get_variables (const ProblemDescDB &problem_db)
  Used by the standard envelope constructor to instantiate the correct letter class.
- Variables * get_variables (const SharedVariablesData &svd) const
  Used by the alternate envelope constructors, by read functions, and by copy() to instantiate a new letter class.
- short method_map (short view_spec, bool relaxed) const
  infer domain from method selection
- short method_domain (const ProblemDescDB &problem_db) const
  infer domain from method selection
- short method_view (const ProblemDescDB &problem_db) const
  infer view from method selection
- short response_view (const ProblemDescDB &problem_db) const
  infer view from type of response data set
- void check_view_compatibility ()
  perform sanity checks on view.first and view.second after update
- template<class Archive>
  void load (Archive &ar, const unsigned int version)
  read a Variables object from an archive
- template<class Archive>
  void save (Archive &ar, const unsigned int version) const
  write a Variables object to an archive
- BOOST_SERIALIZATION_SPLIT_MEMBER () Variables *variablesRep
  pointer to the letter (initialized only for the envelope)

Private Attributes

- int referenceCount
  number of objects sharing variablesRep

Friends

- class boost::serialization::access
  for serializing private data members
- bool operator==(const Variables &vars1, const Variables &vars2)
  strict equality operator (for boost hash-based lookups)
- bool operator!=(const Variables &vars1, const Variables &vars2)
  strict inequality operator
- bool nearby(const Variables &vars1, const Variables &vars2, Real rel_tol)
  tolerance-based equality operator
- std::size_t hash_value (const Variables &vars)
  hash_value
13.167. VERIFICATION CLASS REFERENCE

13.166.1 Detailed Description
Base class for the variables class hierarchy.

The Variables class is the base class for the class hierarchy providing design, uncertain, and state variables for continuous and discrete domains within a Model. Using the fundamental arrays from the input specification, different derived classes define different views of the data. For memory efficiency and enhanced polymorphism, the variables hierarchy employs the "letter/envelope idiom" (see Coplien "Advanced C++", p. 133), for which the base class (Variables) serves as the envelope and one of the derived classes (selected in Variables::get_variables()) serves as the letter.

13.166.2 Member Function Documentation

void build_views ( ) [inline], [protected]
construct active/inactive views of all variables arrays

References Variables::build_active_views(), Variables::build_inactive_views(), Variables::sharedVarsData, Variables::view(), and SharedVariablesData::view().

Referenced by MixedVariables::MixedVariables(), RelaxedVariables::RelaxedVariables(), MixedVariables::reshape(), and RelaxedVariables::reshape().

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

- DakotaVariables.hpp

13.167 Verification Class Reference
Base class for managing common aspects of verification studies.

Inheritance diagram for Verification:

```
Verification
    \-- Analyzer
        \-- Iterator
    \-- RichExtrapVerification
```

Protected Member Functions

- Verification (ProblemDescDB &problem_db, Model &model)
  constructor
- Verification (unsigned short method_name, Model &model)
  alternate constructor for instantiations "on the fly"
- ~Verification ()
  destructor
• `void core_run ()`
  
  `core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post`

• `void print_results (std::ostream &s)`
  
  `print the final iterator results`

• `virtual void perform_verification ()=0`
  
  `Redefines the core_run() virtual function for the Verification branch.`

**Additional Inherited Members**

**13.167.1 Detailed Description**

Base class for managing common aspects of verification studies.

The `Verification` base class manages common data and functions, such as those involving ...

**13.167.2 Member Function Documentation**

`void core_run ( ) [inline], [protected], [virtual]`

`core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post`

  Virtual run function for the iterator class hierarchy. All derived classes need to redefine it.

  Reimplemented from `Iterator`.  
  References `Verification::perform_verification()`.

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

`print the final iterator results`

  This virtual function provides additional iterator-specific final results outputs beyond the function evaluation summary printed in `finalize_run()`.  
  Reimplemented from `Analyzer`.  
  Reimplemented in `RichExtrapVerification`.  
  References `Analyzer::print_results()`.  
  Referenced by `RichExtrapVerification::print_results()`.  
  The documentation for this class was generated from the following files:

• DakotaVerification.hpp
• DakotaVerification.cpp
Chapter 14

File Documentation

14.1 dakota_dll_api.cpp File Reference

This file contains a DakotaRunner class, which launches DAKOTA.

Namespaces

- Dakota
  
  The primary namespace for DAKOTA.

Constant Groups

- Dakota
  
  The primary namespace for DAKOTA.

Functions

- void DAKOTA_DLL_FN dakota_create (int *dakota_ptr_int, const char *logname)
  
  create and configure a new DakotaRunner, adding it to list of instances

- int DAKOTA_DLL_FN dakota_readInput (int id, const 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 ()

865
get the DAKOTA pointer to ModelCenter
• void set_mc_ptr_int (int ptr_int)
  set the DAKOTA pointer to ModelCenter
• int get_dc_ptr_int ()
  get the DAKOTA pointer to ModelCenter current design point
• void set_dc_ptr_int (int ptr_int)
  set the DAKOTA pointer to ModelCenter current design point

14.1.1 Detailed Description
This file contains a DakotaRunner class, which launches DAKOTA.

14.1.2 Function Documentation
void DAKOTA_DLL_FN dakota_stop ( int * id )
command DakotaRunner instance id to stop execution
  TODO: trick application to quit through the syscall interface or throw exception.

14.2 dakota_dll_api.h File Reference
API for DLL interactions.

Functions
• void DAKOTA_DLL_FN dakota_create (int *dakota_ptr_int, const char *logname)
  create and configure a new DakotaRunner, adding it to list of instances
• int DAKOTA_DLL_FN dakota_readInput (int id, const 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
14.3. **DAKOTA\_TABULAR\_IO.HPP FILE REFERENCE**

- void DAKOTA\_DLL\_FN dakota\_get\_variable\_info (int id, char **pVarNames, int *pNumVarNames, char **pRespNames, int *pNumRespNames)
  
  *return the variable and response names*

### 14.2.1 Detailed Description

API for DLL interactions.

### 14.2.2 Function Documentation

**void DAKOTA\_DLL\_FN dakota\_stop ( int *id )**

command DakotaRunner instance id to stop execution

  *TODO: trick application to quit through the syscall interface or throw exception.*

### 14.3 **dakota\_tabular\_io.hpp File Reference**

Utility functions for reading and writing tabular data files. Emerging utilities for tabular file I/O. For now, just extraction of capability from separate contexts to facilitate rework. These augment (and leverage) those in data\_util.h.

#### Namespaces

- **Dakota**
  
  *The primary namespace for DAKOTA.*

#### Constant Groups

- **Dakota**
  
  *The primary namespace for DAKOTA.*

#### Functions

- **void open\_file (std::ifstream &data\_file, const std::string &input\_filename, const std::string &context\_message)**
  
  *open the file specified by name for reading, using passed input stream, presenting context-specific error on failure*

- **void open\_file (std::ofstream &data\_file, const std::string &output\_filename, const std::string &context\_message)**
  
  *open the file specified by name for writing, using passed output stream, presenting context-specific error on failure*

- **void write\_header\_tabular (std::ostream &tabular\_ostream, const Variables &vars, const Response &response, const std::string &counter\_label, bool active\_only=false, bool response\_labels=true)**
  
  *output the header row (labels) for a tabular data file used by Analyzer and Graphics*

- **void write\_data\_tabular (std::ostream &tabular\_ostream, const Variables &vars, const Response &response, size\_t counter=NPOS, bool active\_only=false, bool write\_responses=true)**
  
  *output a row of tabular data from variables and response object used by graphics to append to tabular file during iteration*

- **void write\_data\_tabular (const std::string &output\_filename, const std::string &context\_message, const RealVectorArray &output\_coeffs, const UShort2DArray &output\_indices)**
PCE export: write freeform format file with whitespace-separated data where each row has num_fns reals from coeffs, followed by num_vars unsigned shorts from indices.

- **bool exists_extra_data (std::istream &tabular_file)**
  Check if an input stream contains unexpected additional data.

- **void read_header_tabular (std::istream &input_stream)**
  read and discard header line from the stream

- **void read_data_tabular (const std::string &input_filename, const std::string &context_message, RealVector &input_data, size_t num_entries, bool annotated)**
  read possibly header-annotated whitespace-separated data into a vector of length num_entries; if annotated then it’s a column vector for now

- **void read_data_tabular (const std::string &input_filename, const std::string &context_message, RealArray &input_vector, bool annotated, size_t num_vars)**
  read possibly header-annotated whitespace-separated data into a dynamic vector with minimal error checking

- **void read_data_tabular (const std::string &input_filename, const std::string &context_message, RealVectorArray &input_coeffs, UShort2DArray &input_indices, bool annotated, size_t num_vars, size_t num_fns)**
  PCE import: read possibly header-annotated whitespace-separated data of unknown length where each row has num_fns reals followed by num_vars unsigned shorts; append data to arrays passed by reference.

- **void read_data_tabular (const std::string &input_filename, const std::string &context_message, VariablesList &input_vars, ResponseList &input_resp, const SharedVariablesData &svd, size_t num_c_vars, const ActiveSet &temp_set, bool annotated, bool verbose=false)**
  read whitespace-separated data with optional row and column headers into lists of Variables (using provided SVD) and Responses until out of data; continuous variables only

- **void read_data_tabular (const std::string &input_filename, const std::string &context_message, RealMatrix &input_matrix, size_t num_rows, size_t num_cols, bool annotated, bool verbose=false)**
  read whitespace-separated data with optional row and column headers into a single matrix

### 14.3.1 Detailed Description

Utility functions for reading and writing tabular data files Emerging utilities for tabular file I/O. For now, just extraction of capability from separate contexts to facilitate rework. These augment (and leverage) those in data-util.h. Design/capability goals: Ability to read / write data with row/col headers or in free-form Detect premature end of file, report if extra data More consistent and reliable checks for file open errors Require right number of cols in header mode; only total data checking in free-form (likely) Allow comment character for header rows or even in data? variables vs. variables/responses for both read and write Should we support CSV? delimiter = ‘,’; other? Verify treatment of trailing newline without reading a zero Allow reading into the transpose of the data structure

### 14.4 dll_tester.cpp File Reference

Test the DLL with a DAKOTA input file.

**Functions**

- **int main (int argc, char *argv[])**
  The main program for exercising the DLL API with a simple command-line.
14.4.1 Detailed Description
Test the DLL with a DAKOTA input file.

14.5 JEGAOptimizer.cpp File Reference
Contains the implementation of the JEGAOptimizer class.

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.

Namespaces
- Dakota
  The primary namespace for DAKOTA.

Constant Groups
- Dakota
  The primary namespace for DAKOTA.

Functions
- template<typename T>
  string asstring (const T &val)
  Creates a string from the argument val using an ostringstream.

14.5.1 Detailed Description
Contains the implementation of the JEGAOptimizer class.

14.6 JEGAOptimizer.hpp File Reference
Contains the definition of the JEGAOptimizer class.

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

- Dakota

  The primary namespace for DAKOTA.

Constant Groups

- Dakota

  The primary namespace for DAKOTA.

14.6.1 Detailed Description

Contains the definition of the JEGAOptimizer class.

14.7 library_mode.cpp File Reference

file containing a mock simulator main for testing Dakota in library mode

Classes

- struct callback_data

Functions

- void fpinit_AS ()
- void run_dakota_parse (const char *dakota_input_file)

  Run a Dakota LibraryEnvironment, mode 1: parsing an input file.
- void run_dakota_data ()

  Run a Dakota LibraryEnvironment, mode 2: from C++ API inserted data.
- void run_dakota_mixed (const char *dakota_input_file, bool mpirun_flag)

  Run a Dakota LibraryEnvironment, from string or input file input, supplemented with additional C++ API adjustments.
- void serial_interface_plugin (Dakota::LibraryEnvironment &env)

  Convenience function with simplest example of interface plugin: plugin a serial DirectApplicInterface that can be constructed independent of Dakota's configuration details.
- void parallel_interface_plugin (Dakota::LibraryEnvironment &env)

  Convenience function to plug a library client's interface into the appropriate model, demonstrating use of Dakota parallel configuration in constructing the plugin Interface on the right MPI_Comm.
- static void callback_function (Dakota::ProblemDescDB *db, void *ptr)

  Example: user-provided post-parse callback (Dakota::DbCallbackFunction)
- int main (int argc, char *argv[ ])

  A mock simulator main for testing Dakota in library mode.
14.7. LIBRARY_MODE.CPP FILE REFERENCE

Variables

- static const char serial_input []
  Default Dakota input string for serial case (rosenbrock):
- static const char parallel_input []
  Default Dakota input string for parallel case (text_book)

14.7.1 Detailed Description

file containing a mock simulator main for testing Dakota in library mode

14.7.2 Function Documentation

void fpinit ASL ( )
Floating-point initialization from AMPL: switch to 53-bit rounding if appropriate, to eliminate some cross-platform differences.
  Referenced by main().

void run_dakota_parse ( const char * dakota_input_file )
Run a Dakota LibraryEnvironment, mode 1: parsing an input file.
  Simplest library case: this function parses from an input file to define the ProblemDescDB data.
  References Environment::execute(), ProgramOptions::input_file(), Environment::mpi_manager(), MPIManager::mpirun_flag(), parallel_interface_plugin(), serial_interface_plugin(), and MPIManager::world_rank().
  Referenced by main().

void run_dakota_data ( )
Run a Dakota LibraryEnvironment, mode 2: from C++ API inserted data.
  Rather than parsing from an input file, this function populates Data class objects directly using a minimal specification and relies on constructor defaults and post-processing in post_process() to fill in the rest.
  References DataInterface::data_rep(), DataResponses::data_rep(), DataVariables::data_rep(), DataMethod::data_rep(), LibraryEnvironment::done_modifying_db(), Environment::execute(), DataResponsesRep::gradientType, DataResponsesRep::hessianType, LibraryEnvironment::insert_nodes(), DataMethodRep::methodName, Environment::mpi_manager(), MPIManager::mpirun_flag(), ParallelLibrary::mpirun_flag(), DataVariablesRep::numContinuousDesVars, DataResponsesRep::numNonlinearIneqConstraints, DataResponsesRep::numObjectiveFunctions, parallel_interface_plugin(), Environment::parallel_library(), serial_interface_plugin(), and ParallelLibrary::world_rank().
  Referenced by main().

void run_dakota_mixed ( const char * dakota_input_file, bool mpirun_flag )
Run a Dakota LibraryEnvironment, from string or input file input, supplemented with additional C++ API adjustments.
  Function to encapsulate the Dakota object instantiations for mode 3: mixed parsing and direct updating.
  This function showcases multiple features. For parsing, either an input file (dakota_input_file != NULL) or a default input string (dakota_input_file == NULL) are shown. This parsed input is then mixed with input from three sources: (1) input from a user-supplied callback function, (2) updates to the DB prior to Environment instantiation, (3) updates directly to Iterators/Models following Environment instantiation.
  References callback_function(), LibraryEnvironment::done_modifying_db(), ProgramOptions::echo_input(), Environment::execute(), LibraryEnvironment::filtered_model_list(), ProblemDescDB::get_sa(), ProgramOptions::input_file(),
void serial_interface_plugin ( Dakota::LibraryEnvironment & env )

Convenience function with simplest example of interface plugin: plugin a serial DirectApplicInterface that can be constructed independent of Dakota’s configuration details.

Demonstration of simple plugin where client code doesn’t require access to detailed Dakota data (such as Model-based parallel configuration information) to construct the DirectApplicInterface. This example plugs-in a derived serial direct application interface instance (“plugin_rosenbrock”).

References Dakota::abort_handler(), LibraryEnvironment::plugin_interface(), and Environment::problem_description_db().

Referenced by run_dakota.data(), run_dakota.mixed(), and run_dakota.parse().

void parallel_interface_plugin ( Dakota::LibraryEnvironment & env )

Convenience function to plug a library client’s interface into the appropriate model, demonstrating use of Dakota parallel configuration in constructing the plugin Interface on the right MPI Comm.

From a filtered list of Model candidates, plug-in a derived direct application interface instance (“plugin_textbook” for parallel). This approach provides more complete access to the Model, e.g., for access to analysis communicators.

References Dakota::abort_handler(), Interface::assign_rep(), LibraryEnvironment::filtered_model_list(), Environment::problem_description_db(), and ProblemDescDB::set_db_model_nodes().

Referenced by run_dakota.data(), run_dakota.mixed(), and run_dakota.parse().

static void callback_function ( Dakota::ProblemDescDB * db, void * ptr ) [static]

Example: user-provided post-parse callback (Dakota::DbCallbackFunction)

Example of user-provided callback function (an instance of Dakota::DbCallbackFunction) to override input provided by parsed Dakota input file or input string data.

References Dakota::contains(), ProblemDescDB::get_sa(), ProblemDescDB::get_string(), ProblemDescDB::resolve_top_method(), callback_data::rosen_cdv_upper_bd, and ProblemDescDB::set().

Referenced by run_dakota.mixed().

int main ( int argc, char * argv[ ] )

A mock simulator main for testing Dakota in library mode.

Overall Usage: dakota_library_mode [-mixed] [dakota.in]

Uses alternative instantiation syntax as described in the library mode documentation within the Developers Manual. Tests several problem specification modes:

(1) run_dakota_parse: reads all problem specification data from a Dakota input file. Usage: dakota_library_mode dakota.in

(2) run_dakota_data: creates all problem specification from direct Data instance instantiations in the C++ code. Usage: dakota_library_mode

(3) run_dakota_mixed: a mixture of input parsing and direct data updates, where the data updates occur: (a) via the DB during Environment instantiation, and (b) via Iterators/Models following Environment instantiation.
Usage: dakota_library_mode -mixed (input from default string) dakota_library_mode -mixed dakota.in (input from specified file)

Serial cases use a plugin rosenbrock model, while parallel cases use textbook.
References MPIManager::detect_parallel.launch(), fpinit_ASL(), Dakota::mpi_debug_hold(), run_dakota_data(), run_dakota_mixed(), and run_dakota_parse().

### 14.7.3 Variable Documentation

const char serial_input[] [static]

Initial value:

```
"method",
  "optpp_q_newton",
  "max_iterations = 50",
  "convergence_tolerance = 1e-4",
"variables",
  "continuous_design = 2",
  "descriptors 'x1' 'x2'",
"interface",
  "direct",
  "analysis_driver = 'plugin_rosenbrock'",
"responses",
  "num_objective_functions = 1",
  "analytic_gradients",
  "no_hessians"
```

Default Dakota input string for serial case (rosenbrock):
Referenced by run_dakota_mixed().

const char parallel_input[] [static]

Initial value:

```
"method",
  "optpp_q_newton",
  "max_iterations = 50",
  "convergence_tolerance = 1e-4",
"variables",
  "continuous_design = 2",
  "descriptors 'x1' 'x2'",
"interface",
  "direct",
  "analysis_driver = 'plugin_text_book'",
"responses",
  "num_objective_functions = 1",
  "num_nonlinear_inequality_constraints = 2",
  "analytic_gradients",
  "no_hessians"
```

Default Dakota input string for parallel case (text_book)
Referenced by run_dakota_mixed().

### 14.8 library_split.cpp File Reference

file containing a mock simulator main for testing DAKOTA in library mode on a split communicator
Functions

- void manage_mpi (MPI_Comm &my_comm, int &color)
  
  Split MPI_COMM_WORLD, returning the comm and color.
- void gen_dakota_input (const int &color, std::string &input)
  
  Return the appropriate DAKOTA input based on color (1 or 2)
- void run_dakota (const MPI_Comm &comm, const std::string &input, const int &color)
  
  Launch DAKOTA on passed communicator, tagging output/error with color.
- void collect_results ()
  
  Wait for and collect results from DAKOTA runs.
- int main (int argc, char ∗argv[ ])
  
  Driver routine for testing library mode with partitioned MPI_Comm. This test fixture requires MPI and can be run on 3–8 processors.

14.8.1 Detailed Description

file containing a mock simulator main for testing DAKOTA in library mode on a split communicator

14.9 main.cpp File Reference

file containing the main program for DAKOTA

Functions

- void fpinit_ASLS ()
- int main (int argc, char ∗argv[ ])
  
  The main DAKOTA program.

14.9.1 Detailed Description

file containing the main program for DAKOTA

14.9.2 Function Documentation

void fpinit_ASLS ()

Floating-point initialization from AMPL: switch to 53-bit rounding if appropriate, to eliminate some cross-platform differences.

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 ExecutableEnvironment and invoke its execute() virtual function.

References Environment::check(), ExecutableEnvironment::execute(), fpinit_ASLS(), Dakota::mpi_debug_hold(), and Dakota::register_signal handlers().
14.10 restart_util.cpp File Reference

file containing the DAKOTA restart utility main program

Namespaces

• Dakota
  
  The primary namespace for DAKOTA.

Constant Groups

• Dakota
  
  The primary namespace for DAKOTA.

Functions

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

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

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

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

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

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

• int nidr_save_exedir (const char *, int)
  
  Use NIDR to add exedir and . to $PATH.

• int main (int argc, char *argv[])
  
  The main program for the DAKOTA restart utility.

14.10.1 Detailed Description

file containing the DAKOTA restart utility main program

14.10.2 Function Documentation

int main ( int argc, char * argv[] )

The main program for the DAKOTA restart utility.

Parse command line inputs and invoke the appropriate utility function (print_restart(), print_restart_tabular(),
read_neutral(), repair_restart(), or concatenate_restart()).

References Dakota::concatenate_restart(), nidr_save_exedir(), Dakota::print_restart(), Dakota::print_restart_pdb(),
Dakota::print_restart_tabular(), Dakota::read_neutral(), and Dakota::repair_restart().
Index

~Approximation
  Dakota::Approximation, 310
~Constraints
  Dakota::Constraints, 350
~DataFitSurrModel
  Dakota::DataFitSurrModel, 360
~EffGlobalMinimizer
  Dakota::EffGlobalMinimizer, 421
~Environment
  Dakota::Environment, 429
~Interface
  Dakota::Interface, 468
~Iterator
  Dakota::Iterator, 477
~Model
  Dakota::Model, 539
~NonDAdaptiveSampling
  Dakota::NonDAdaptiveSampling, 598
~ProblemDescDB
  Dakota::ProblemDescDB, 731
~SharedApproxData
  Dakota::SharedApproxData, 792
  initPts
    Dakota::JEGAOptimizer, 494
_model
  Dakota::JEGAOptimizer::Evaluator, 437

A
  Dakota::CONMINOptimizer, 342
APPSEvalMgr, 317
  Dakota::APPSEvalMgr, 319
APPSoptimizer, 320
abort_handler_t
  Dakota, 135
accepts_multiple_points
  Dakota::JEGAOptimizer, 493
ActiveSet, 283
actualModel
  Dakota::DataFitSurrModel, 365
add_anchor_to_surfdata
  Dakota::SurfpackApproximation, 825
add_datapoint
  Dakota::Graphics, 455
algorithm_space_model
  Dakota::Analyzer, 290
  Dakota::EffGlobalMinimizer, 421
  Dakota::Minimizer, 511
  Dakota::NonDBayesCalibration, 600
  Dakota::NonDExpansion, 613
  Dakota::NonDGlobalInterval, 620
  Dakota::NonDReliability, 670
allContinuousIds
  Dakota::SharedVariablesDataRep, 805
Analyzer, 285
anisotropic_order_to_dimension_preference
  Dakota::NonDIntegration, 634
append_approximation
  Dakota::ApproximationInterface, 316
  Dakota::DataFitSurrModel, 362, 363
ApplicationInterface, 291
approx_subprob_constraint_eval
  Dakota::SurrBasedLocalMinimizer, 833
approx_subprob_objective_eval
  Dakota::SurrBasedLocalMinimizer, 833
approxBuilds
  Dakota::SurrogateModel, 842
Approximation, 306
  Dakota::Approximation, 309, 310
ApproximationInterface, 312
array_insert
  Dakota::ResultsDBAny, 773
assess_reconstruction
  Dakota::EfficientSubspaceMethod, 424
assign_rep
  Dakota::Interface, 469
  Dakota::Iterator, 480
  Dakota::Model, 544
asstring
  Dakota, 137
asynchronous_local_analyses
  Dakota::ProcessHandleApplicInterface, 739
INDEX

asynchronous_local_evaluations
   Dakota::ApplicationInterface, 303
asynchronous_local_evaluations_nowait
   Dakota::ApplicationInterface, 304
augmented_lagrangianMerit
   Dakota::SurrBasedMinimizer, 837
B
   Dakota::CONMINOptimizer, 342
   BaseConstructor, 322
   brent_minimize
      Dakota::NonlinearCGOptimizer, 684
build
   Dakota::Approximation, 310
   Dakota::PecosApproximation, 723
   Dakota::SurfpackApproximation, 825
   Dakota::TANA3Approximation, 847
   Dakota::TaylorApproximation, 848
build_approximation
   Dakota::ApproximationInterface, 316
   Dakota::DataFitSurrModel, 361
build_global
   Dakota::DataFitSurrModel, 364
build_local_multipoint
   Dakota::DataFitSurrModel, 364
build_views
   Dakota::Constraints, 351
   Dakota::Variables, 863
buildDataOrder
   Dakota::SharedApproxData, 793
C
   Dakota::CONMINOptimizer, 342
   CAUVELbl
      Dakota, 277
   CEUVELbl
      Dakota, 278
   COLINApplication, 323
   COLINOptimizer, 326
      Dakota::COLINOptimizer, 328
   CONMINOptimizer, 336
   CT
      Dakota::CONMINOptimizer, 341
   callback_data, 323
   callback_function
      library_mode.cpp, 872
   ccv_index_map
      Dakota::NestedModel, 562
   cdv_index_map
      Dakota::NestedModel, 562
   cdrv_index_map
      Dakota::NestedModel, 562
   check_and_broadcast
      Dakota::ProblemDescDB, 731
   check_input
      Dakota::ProblemDescDB, 732
   check_variables
      Dakota::NonDIntegration, 634
   check_wait
      Dakota::ProcessHandleApplicInterface, 738
   clear_all
      Dakota::Approximation, 312
   clear_current
      Dakota::Approximation, 311
      Dakota::TANA3Approximation, 847
   Clone
      Dakota::JEGAOptimizer::Evaluator, 436
   colin_cache_lookup
      Dakota::COLINOptimizer, 330
   colin_request_to_dakota_request
      Dakota::COLINApplication, 325
   CollabHybridMetaIterator, 330
   collect_evaluation_impl
      Dakota::COLINApplication, 325
   CommandLineHandler, 332
   CommandShell, 333
   compute
      Dakota::DiscrepancyCorrection, 413
      compute_covariance_metric
         Dakota::NonDExpansion, 614
         Dakota::NonDStochCollocation, 681
   compute_final_statistics_metric
      Dakota::NonDExpansion, 614
      Dakota::NonDStochCollocation, 682
   compute_statistics
      Dakota::NonDExpansion, 614
   concatenate_restart
      Dakota, 139
   ConcurrentMetaIterator, 334
   conminInfo
      Dakota::CONMINOptimizer, 340
   constraint0_evaluator
      Dakota::SNLLOptimizer, 818
   constraint1_evaluator
      Dakota::SNLLOptimizer, 819
   constraint1_evaluator_gn
      Dakota::SNLLLeastSq, 812
   constraint2_evaluator
      Dakota::SNLLOptimizer, 819
constraint2_evaluator_gn
Dakota::SNLLLeastSq, 813

cost
Dakota::COLINOptimizer, 330
Dakota::SurrBasedMinimizer, 838

constraintMappingIndices
Dakota::CONMINOptimizer, 340
Dakota::DOTOptimizer, 416

constraintMappingMultipliers
Dakota::CONMINOptimizer, 340
Dakota::DOTOptimizer, 416

constraintMappingOffsets
Dakota::CONMINOptimizer, 340
Dakota::DOTOptimizer, 417

constraintValues
Dakota::CONMINOptimizer, 340
Dakota::DOTOptimizer, 416

Constraints, 343
Dakota::Constraints, 350
converge_order
Dakota::RichExtrapVerification, 781

converge_qoi
Dakota::RichExtrapVerification, 781

copy
Dakota::Constraints, 351
Dakota::SharedVariablesData, 802

copy_data
Dakota::DDACEDesignCompExp, 405

core_run
Dakota::Iterator, 478
Dakota::LeastSq, 496
Dakota::NonD, 587
Dakota::Optimizer, 693
Dakota::PStudyDACE, 745
Dakota::SurrBasedMinimizer, 836
Dakota::Verification, 864

create_evaluation_process
Dakota::ProcessHandleApplicInterface, 738

create_plots_2d
Dakota::Graphics, 455

create_tabular_datastream
Dakota::Graphics, 455

CreateEvaluator
Dakota::JEGAOptimizer::EvaluatorCreator, 438

cv_index_map
Dakota::NestedModel, 561

DAUIVLbl
Dakota, 278

DAURVLbl
Dakota, 278

DDACEDesignCompExp, 402
Dakota::DDACEDesignCompExp, 404

DEUIVLbl
Dakota, 278

DEURVLbl
Dakota, 278

DF
Dakota::CONMINOptimizer, 342

Dakota::DOTOptimizer, 413

DakFuncs0
Dakota, 140

Dakota, 67

abort_handler_t, 135
asstring, 137

CAUVLbl, 277
CEUVLbl, 278

concatenate_restart, 139
DAUIVLbl, 278

DAURVLbl, 278

DEUIVLbl, 278

DEURVLbl, 278

DakFuncs0, 140

DiscSetLbl, 279

FIELD_NAMES, 140

flush, 135

get_npath, 135

getRmax, 136

getdist, 136

id_vars_exact_compare, 137

kw_1, 140, 214

kw_10, 142, 215

kw_100, 163, 232

kw_101, 163, 232

kw_102, 163, 232

kw_103, 163, 232

kw_104, 163, 232

kw_105, 164, 233

kw_106, 164, 233

kw_107, 164, 233

kw_108, 164, 233

kw_109, 164, 233

kw_11, 142, 215

kw_110, 165, 233

kw_111, 165, 233

kw_112, 165, 234

kw_113, 165, 234

kw_114, 165, 234

kw_115, 165, 234
<p>| kw_204, 183, 250 | kw_249, 195, 259 |
| kw_205, 184, 250 | kw_25, 145, 218 |
| kw_206, 184, 250 | kw_250, 195, 259 |
| kw_207, 184, 251 | kw_251, 195, 259 |
| kw_208, 184, 251 | kw_252, 195, 259 |
| kw_209, 184, 251 | kw_253, 195, 259 |
| kw_21, 145, 217 | kw_254, 195, 260 |
| kw_210, 184, 251 | kw_255, 196, 260 |
| kw_211, 185, 251 | kw_256, 196, 260 |
| kw_212, 185, 251 | kw_257, 196 |
| kw_213, 186, 252 | kw_258, 196, 260 |
| kw_214, 186, 252 | kw_259, 196, 260 |
| kw_215, 186, 252 | kw_26, 146, 218 |
| kw_216, 187, 252 | kw_260, 197, 261 |
| kw_217, 187, 253 | kw_261, 197, 261 |
| kw_218, 187, 253 | kw_262, 197, 261 |
| kw_219, 187, 253 | kw_263, 197, 261 |
| kw_22, 145, 217 | kw_264, 197, 261 |
| kw_220, 188, 253 | kw_265, 198, 261 |
| kw_221, 188, 253 | kw_266, 198, 262 |
| kw_222, 188, 253 | kw_267, 198, 262 |
| kw_223, 188, 254 | kw_268, 199, 262 |
| kw_224, 189, 254 | kw_269, 199, 262 |
| kw_225, 189, 254 | kw_27, 146, 218 |
| kw_226, 189, 254 | kw_270, 199, 262 |
| kw_227, 189, 255 | kw_271, 199, 262 |
| kw_228, 190, 255 | kw_272, 200, 263 |
| kw_229, 190, 255 | kw_273, 200, 263 |
| kw_23, 145, 218 | kw_274, 200, 263 |
| kw_230, 190, 255 | kw_275, 201, 263 |
| kw_231, 190, 255 | kw_276, 201, 263 |
| kw_232, 191, 255 | kw_277, 201, 263 |
| kw_233, 191, 256 | kw_278, 201, 264 |
| kw_234, 192, 256 | kw_279, 202, 264 |
| kw_235, 192, 256 | kw_28, 146, 218 |
| kw_236, 192, 256 | kw_280, 202, 264 |
| kw_237, 192, 256 | kw_281, 202, 264 |
| kw_238, 192, 257 | kw_282, 202, 265 |
| kw_239, 193, 257 | kw_283, 202, 265 |
| kw_24, 145, 218 | kw_284, 203, 265 |
| kw_240, 193, 257 | kw_285, 203, 265 |
| kw_241, 257 | kw_286, 203, 265 |
| kw_242, 193, 257 | kw_287, 204, 265 |
| kw_243, 193, 257 | kw_288, 204, 266 |
| kw_244, 194, 258 | kw_289, 204, 266 |
| kw_245, 194, 258 | kw_29, 146, 219 |
| kw_246, 194, 258 | kw_290, 204, 266 |
| kw_247, 194, 258 | kw_291, 205, 266 |
| kw_248, 194, 258 | kw_292, 205, 267 |</p>
<table>
<thead>
<tr>
<th>kw</th>
<th>START INDEX</th>
<th>END INDEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>kw_75</td>
<td>157</td>
<td>227</td>
</tr>
<tr>
<td>kw_76</td>
<td>157</td>
<td>227</td>
</tr>
<tr>
<td>kw_77</td>
<td>157</td>
<td>227</td>
</tr>
<tr>
<td>kw_78</td>
<td>158</td>
<td>227</td>
</tr>
<tr>
<td>kw_79</td>
<td>158</td>
<td>228</td>
</tr>
<tr>
<td>kw_8</td>
<td>142</td>
<td>215</td>
</tr>
<tr>
<td>kw_80</td>
<td>158</td>
<td>228</td>
</tr>
<tr>
<td>kw_81</td>
<td>159</td>
<td>228</td>
</tr>
<tr>
<td>kw_82</td>
<td>159</td>
<td>228</td>
</tr>
<tr>
<td>kw_83</td>
<td>159</td>
<td>228</td>
</tr>
<tr>
<td>kw_84</td>
<td>159</td>
<td>229</td>
</tr>
<tr>
<td>kw_85</td>
<td>160</td>
<td>229</td>
</tr>
<tr>
<td>kw_86</td>
<td>160</td>
<td>229</td>
</tr>
<tr>
<td>kw_87</td>
<td>160</td>
<td>229</td>
</tr>
<tr>
<td>kw_88</td>
<td>160</td>
<td>229</td>
</tr>
<tr>
<td>kw_89</td>
<td>161</td>
<td>230</td>
</tr>
<tr>
<td>kw_9</td>
<td>142</td>
<td>215</td>
</tr>
<tr>
<td>kw_90</td>
<td>161</td>
<td>230</td>
</tr>
<tr>
<td>kw_91</td>
<td>161</td>
<td>230</td>
</tr>
<tr>
<td>kw_92</td>
<td>161</td>
<td>230</td>
</tr>
<tr>
<td>kw_93</td>
<td>161</td>
<td>230</td>
</tr>
<tr>
<td>kw_94</td>
<td>161</td>
<td>230</td>
</tr>
<tr>
<td>kw_95</td>
<td>162</td>
<td>231</td>
</tr>
<tr>
<td>kw_96</td>
<td>162</td>
<td>231</td>
</tr>
<tr>
<td>kw_97</td>
<td>162</td>
<td>231</td>
</tr>
<tr>
<td>kw_98</td>
<td>162</td>
<td>231</td>
</tr>
<tr>
<td>kw_99</td>
<td>162</td>
<td>231</td>
</tr>
<tr>
<td>lookup_by_val</td>
<td>138</td>
<td></td>
</tr>
<tr>
<td>mindist</td>
<td>136</td>
<td></td>
</tr>
<tr>
<td>mindistindx</td>
<td>136</td>
<td></td>
</tr>
<tr>
<td>mpi_debug_hold</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>my_cp</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>NUMBER_OF_FIELDS</td>
<td>140</td>
<td></td>
</tr>
<tr>
<td>operator==</td>
<td>135–137</td>
<td></td>
</tr>
<tr>
<td>PRPMultiIndexCache</td>
<td>134</td>
<td></td>
</tr>
<tr>
<td>PRPMultiIndexQueue</td>
<td>134</td>
<td></td>
</tr>
<tr>
<td>perform_analysis</td>
<td>137</td>
<td></td>
</tr>
<tr>
<td>print_restart</td>
<td>138</td>
<td></td>
</tr>
<tr>
<td>print_restart_pdb</td>
<td>138</td>
<td></td>
</tr>
<tr>
<td>print_restart_tabular</td>
<td>138</td>
<td></td>
</tr>
<tr>
<td>read_neutral</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>register_signal_handlers</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>repair_restart</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>SUBMETHOD_COLLABORATIVE</td>
<td>134</td>
<td></td>
</tr>
<tr>
<td>SCI_FIELD_NAMES</td>
<td>281</td>
<td></td>
</tr>
<tr>
<td>set_compare</td>
<td>137</td>
<td></td>
</tr>
<tr>
<td>slmap</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>start_dakota_heartbeat</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>start_grid_computing</td>
<td>136</td>
<td></td>
</tr>
<tr>
<td>stop_grid_computing</td>
<td>137</td>
<td></td>
</tr>
<tr>
<td>VLI</td>
<td>279</td>
<td></td>
</tr>
<tr>
<td>VLR</td>
<td>279</td>
<td></td>
</tr>
<tr>
<td>var_mp_ebound</td>
<td>281</td>
<td></td>
</tr>
<tr>
<td>var_mp_check_cau</td>
<td>280</td>
<td></td>
</tr>
<tr>
<td>var_mp_check_ceu</td>
<td>281</td>
<td></td>
</tr>
<tr>
<td>var_mp_check_cv</td>
<td>280</td>
<td></td>
</tr>
<tr>
<td>var_mp_check_daui</td>
<td>280</td>
<td></td>
</tr>
<tr>
<td>var_mp_check_daur</td>
<td>280</td>
<td></td>
</tr>
<tr>
<td>var_mp_check_deui</td>
<td>281</td>
<td></td>
</tr>
<tr>
<td>var_mp_check_deur</td>
<td>281</td>
<td></td>
</tr>
<tr>
<td>var_mp_check_dset</td>
<td>280</td>
<td></td>
</tr>
<tr>
<td>var_mp_drange</td>
<td>281</td>
<td></td>
</tr>
<tr>
<td>Vlch</td>
<td>279</td>
<td></td>
</tr>
</tbody>
</table>

**Dakota::GetLongOpt**
- MandatoryValue, 452
- OptionalValue, 452
- Valueless, 452

**Dakota::APPSEvalMgr**
- APPSEvalMgr, 319
- isReadyForWork, 319
- recv, 319
- submit, 319

**Dakota::APPSOptimizer**
- find_optimum, 321
- initialize_variables_and_constraints, 322
- set_apps_parameters, 322

**Dakota::Analyzer**
- algorithm_space_model, 290
- evaluate_parameter_sets, 290
- initialize_run, 289
- num_samples, 289
- post_run, 289
- pre_output, 289
- print_results, 290
- print_sobol_indices, 291
- read_variables_responses, 291
- variance_based_decomp, 290

**Dakota::ApplicationInterface**
- asynchronous_local_evaluations, 303
- asynchronous_local_evaluations_nowait, 304
- duplication_detect, 301
- init_communicators_checks, 300
- init_serial, 299
- map, 299
- master_dynamic_schedule_analyses, 301
master_dynamic_schedule_evaluations, 302
master_dynamic_schedule_evaluations_nowait, 303
peer_dynamic_schedule_evaluations, 302
peer_dynamic_schedule_evaluations_nowait, 304
peer_static_schedule_evaluations, 302
serve_analyses_synch, 301
serve_evaluations, 300
serve_evaluations_asynch, 305
serve_evaluations_synch, 304
serve_evaluations_synch_peer, 305
set_communicators_checks, 300
stop_evaluation_servers, 300
synch, 299
synch_nowait, 299
synchronous_local_evaluations, 303

Dakota::Approximation
~Approximation, 310
Approximation, 309, 310
build, 310
clear_all, 312
clear_current, 311
finalize, 311
get_approx, 312
operator=, 310
pop, 311
rebuild, 311
restore, 311

Dakota::ApproximationInterface
append_approximation, 316
build_approximation, 316
functionSurfaces, 317
pop_approximation, 317
rebuild_approximation, 316
restore_approximation, 317
update_approximation, 315

Dakota::COLINApplication
colin_request_to_dakota_request, 325
collect_evaluation_impl, 325
dakota_response_to_colin_response, 326
evaluation_available, 325
map_domain, 326
perform_evaluation_impl, 325
set_problem, 324
spawn_evaluation_impl, 325

Dakota::CONMINOptimizer
find_optimum, 328
post_run, 329
returns_multiple_points, 329
set_rng, 329
set_solver_parameters, 329
solver_setup, 329

Dakota::CONMINOptimizer
A, 342
B, 342
C, 342
CT, 341
conminInfo, 340
constraintMappingIndices, 340
constraintMappingMultipliers, 340
constraintMappingOffsets, 340
constraintValues, 340
DF, 342
G1, 341
G2, 342
IC, 343
ISC, 343
MS1, 342
N1, 341
N2, 341
N3, 341
N4, 341
N5, 341
printControl, 340
S, 341
SCAL, 342

Dakota::CommandLineHandler
output_helper, 333

Dakota::CommandShell
flush, 334
operator<<, 334

Dakota::ConcurrentMetaIterator
print_results, 336

Dakota::Constraints
~Constraints, 350
build_views, 351
Constraints, 350
copy, 351
get_constraints, 352
manage_linear_constraints, 352
operator=, 351
reshape, 351

Dakota::DDACEDesignCompExp
copy_data, 405
DDACEDesignCompExp, 404
num_samples, 405
post_run, 405
pre_run, 405
resolve_samples_symbols, 405
Dakota::DOTOptimizer
  constraintMappingIndices, 416
  constraintMappingMultipliers, 416
  constraintMappingOffsets, 417
  constraintValues, 416
dotFDSinfo, 415
dotInfo, 415
dotMethod, 416
intCntlParmArray, 416
printControl, 416
realCntlParmArray, 416
Dakota::DataFitSurrModel
  ~DataFitSurrModel, 360
  actualModel, 365
  append_approximation, 362, 363
  build_approximation, 361
  build_global, 364
  build_local_multipoint, 364
  derived_async_compute_response, 360
  derived_compute_response, 360
  derived_init_communicators, 363
  derived_synchronize, 360
  derived_synchronize_nowait, 361
evaluation_id, 363
export_point, 364
import_points, 363
initialize_export, 364
manage_data_recastings, 364
update_actual_model, 365
update_approximation, 362
update_from_actual_model, 365
Dakota::DirectApplicInterface
  derived_map_ac, 409
  init_communicators_checks, 409
  set_communicators_checks, 409
  synchronous_local_analysis, 409
Dakota::DiscrepancyCorrection
  compute, 413
Dakota::EffGlobalMinimizer
  ~EffGlobalMinimizer, 421
  algorithm_space_model, 421
  get_best_sample, 421
Dakota::EfficientSubspaceMethod
  assess_reconstruction, 424
  init_communicators, 424
  map_xi_to_x, 424
  reduced_space_uq, 424
  uncertain_vars_to_subspace, 424
Dakota::Environment
  ~Environment, 429
  Environment, 428–430
  get_environment, 430
  operator=, 430
  parse, 430
Dakota::ExperimentData
  load_scalar, 440
Dakota::FSUDesignCompExp
  enforce_input_rules, 445
  FSUDesignCompExp, 444
  num_samples, 445
  post_run, 444
  pre_run, 444
Dakota::GaussProcApproximation
  GPmodel_apply, 450
  GaussProcApproximation, 449
trendOrder, 450
Dakota::GetLongOpt
  enroll, 452
  GetLongOpt, 452
  OptType, 452
  parse, 452
  retrieve, 452
  usage, 453
Dakota::Graphics
  add_datapoint, 455
  create_plots_2d, 455
  create_tabular_datastream, 455
  new_dataset, 455
Dakota::GridApplicInterface
  synchronous_local_analysis, 457
Dakota::HierarchSurrModel
  derived_async_compute_response, 460
  derived_compute_response, 460
  derived_synchronize, 460
  derived_synchronize_nowait, 461
evaluation_id, 461
Dakota::Interface
  ~Interface, 468
  assign_rep, 469
  eval_tag_prefix, 469
  get_interface, 470
  Interface, 468
  operator=, 469
  rawResponseMap, 470
response_mapping, 469

Dakota::Iterator
~Iterator, 477
assign_rep, 480
core_run, 478
eval_tag_prefix, 481
finalize_run, 479
get_iterator, 481
initialize_graphics, 479
initialize_run, 478
Iterator, 476, 477
maxEvalConcurrency, 482
operator=, 478
post_run, 478
pre_run, 478
print_results, 479
probDescDB, 482
run, 480
uses_method, 479

Dakota::IteratorScheduler
free_iterator, 486
free_iterator_parallelism, 485
init_evaluation_concurrency, 486, 487
init_iterator, 485
init_iterator_parallelism, 486
init_serial_iterators, 485
IteratorScheduler, 485
master_dynamic_schedule_iterators, 487
run_iterator, 486
schedule_iterators, 487
serve_iterators, 487

Dakota::JEGAOptimizer
_initPts, 494
accepts_multiple_points, 493
find_optimum, 493
GetBestMOSolutions, 492
GetBestSOsolutions, 492
GetBestSolutions, 492
initial_points, 493, 494
JEGAOptimizer, 490
LoadAlgorithmConfig, 490
LoadDakotaResponses, 490
LoadProblemConfig, 490
LoadTheConstraints, 492
LoadTheDesignVariables, 492
LoadTheObjectiveFunctions, 492
LoadTheParameterDatabase, 490
returns_multiple_points, 493
ToDoubleMatrix, 493

Dakota::JEGAOptimizer::Driver
DestroyAlgorithm, 418
Driver, 417
ExtractAllData, 418
PerformIterations, 418

Dakota::JEGAOptimizer::Evaluator
_model, 437
Clone, 436
Description, 434
Evaluate, 435, 436
Evaluator, 433, 434
GetDescription, 436
GetName, 436
GetNumberLinearConstraints, 435
GetNumberNonLinearConstraints, 435
Name, 434
RecordResponses, 435
SeparateVariables, 435

Dakota::JEGAOptimizer::EvaluatorCreator
CreateEvaluator, 438
EvaluatorCreator, 437

Dakota::LeastSq
core_run, 496
finalize_run, 497
get_confidence_intervals, 497
initialize_run, 496
LeastSq, 496
post_run, 496
primary_resp_weighter, 497
print_results, 497
weight_model, 497

Dakota::LibraryEnvironment
filtered_interface_list, 499
filtered_model_list, 500
LibraryEnvironment, 499

Dakota::MatlabInterface
derived_map_ac, 501
matlab_engine_run, 501

Dakota::MetaIterator
post_run, 503

Dakota::Minimizer
algorithm_space_model, 511
data_transform_model, 512
finalize_run, 511
gnewton_set_recast, 512
initialize_run, 510
initialize_scaling, 515
lin_coeffs_modify_n2s, 516
Minimizer, 510
modify_n2s, 516
modify_s2n, 513
need_resp_trans_byvars, 513
objective, 513
objective_gradient, 514
objective_hessian, 514
pack_parameters_buffer, 511
pack_results_buffer, 511
post_run, 510
primary_resp_differencer, 515
resize_best_resp_array, 514
resize_best_vars_array, 514
response_modify_n2s, 516
response_modify_s2n, 513
scale_model, 512
secondary_resp_copier, 512
secondary_resp_scaler, 515
unpack_parameters_buffer, 511
unpack_results_buffer, 511
variables_scaler, 515
Dakota::MixedVarConstraints
MixedVarConstraints, 518
reshape, 518
Dakota::MixedVariables
MixedVariables, 519
read_tabular, 519
Dakota::Model
~Model, 539
assign_rep, 544
derivative_concurrency, 545
derived_interface, 542
estimate_derivatives, 545
estimate_message_lengths, 544
eval_tag_prefix, 543
evaluation_cache, 543
FDstep1, 545
FDstep2, 545
fdGradStepSize, 547
fdHessByFnStepSize, 547
fdHessByGradStepSize, 547
get_model, 545
init_communicators, 543
init_serial, 544
initialize_h, 545
interface_id, 542
local_eval_concurrency, 542
local_eval_synchronization, 542
manage_asv, 547
Model, 539, 540
operator=, 540
probDescDB, 548
subordinate_iterator, 540
subordinate_model, 540
subordinate_models, 543
surrogate_model, 541
synchronize_derivatives, 546
truth_model, 541
update_from_subordinate_model, 541
update_quasi_hessians, 547
update_response, 546
Dakota::NCSUOptimizer
NCSUOptimizer, 555
objective_eval, 556
Dakota::NIDRProblemDescDB
derived_parse_inputs, 567
Dakota::NL2SOLLeastSq
minimize_residuals, 570
Dakota::NLSSOLLeastSq
NLSSOLLeastSq, 577
Dakota::NPSOLOptimizer
NPSOLOptimizer, 687
Dakota::NestedModel
ccv_index_map, 562
cdiv_index_map, 562
cdrv_index_map, 562
cv_index_map, 561
derived_asynch_compute_response, 561
derived_compute_response, 560
derived_init_communicators, 561
derived_master_overload, 561
div_index_map, 562
drv_index_map, 562
evaluation_id, 561
response_mapping, 562
subModel, 563
Dakota::NomadOptimizer
load_parameters, 580
NomadOptimizer, 579
Dakota::NomadOptimizer::Evaluator
eval_x, 432
Evaluator, 431
Dakota::NonD
core_run, 587
finalize_run, 588
initialize_final_statistics, 588
initialize_random_variable_parameters, 588
initialize_random_variable_types, 588
initialize_random_variables, 587
initialize_run, 587
print_distribution_map, 589
print_distribution_mappings, 589
set_u_to_x_mapping, 589
vars_u_to_x_mapping, 589
vars_x_to_u_mapping, 589
Dakota::NonDAdaptImpSampling
initialize, 593
NonDAdaptImpSampling, 592
Dakota::NonDAdaptiveSampling
~NonDAdaptiveSampling, 598
NonDAdaptiveSampling, 598
Dakota::NonDBayesCalibration
algorithm_space_model, 600
NonDBayesCalibration, 600
Dakota::NonDCalibration
NonDCalibration, 602
Dakota::NonDCubature
increment_grid_preference, 604
increment_reference, 605
NonDCubature, 604
num_samples, 605
sampling_reset, 604
Dakota::NonDDREAMBayesCalibration
likelihoodScale, 609
NonDDREAMBayesCalibration, 607
prior_density, 608
prior_sample, 608
problem_size, 607
problem_value, 608
quantify_uncertainty, 608
sample_likelihood, 608
Dakota::NonDExpansion
algorithm_space_model, 613
compute_covariance_metric, 614
compute_final_statistics_metric, 614
compute_statistics, 614
increment_specification_sequence, 613
useDerivs, 615
Dakota::NonDGPImpSampling
NonDGPImpSampling, 626
quantify_uncertainty, 626
Dakota::NonDGPSABayesCalibration
NonDGPSABayesCalibration, 629
quantify_uncertainty, 629
randomSeed, 629
Dakota::NonDGlobalInterval
algorithm_space_model, 620
Dakota::NonDNLSampling
quantify_uncertainty, 631
Dakota::NonDIntegration
anisotropic_order_to_dimension_preference, 634
check_variables, 634
dimension_preference_to_anisotropic_order, 634
NonDIntegration, 633
Dakota::NonDLHSSampling
NonDLHSSampling, 640
quantify_uncertainty, 641
Dakota::NonDLocalReliability
dg_ds_eval, 654
dp2_dbeta_factor, 655
initial_taylor_series, 652
initialize_class_data, 652
initialize_level_data, 653
initialize_mpp_search_data, 653
PMA2_constraint_eval, 652
PMA_constraint_eval, 651
PMA_objective_eval, 651
probability, 655
RIA_constraint_eval, 651
RIA_objective_eval, 651
update_level_data, 654
update_mpp_search_data, 653
Dakota::NonDPolynomialChaos
increment_order, 662
increment_specification_sequence, 662
NonDPolynomialChaos, 661, 662
Dakota::NonDQUESOBayesCalibration
NonDQUESOBayesCalibration, 668
quantify_uncertainty, 668
Dakota::NonDQuadrature
initialize_grid, 666
NonDQuadrature, 665
num_samples, 666
sampling_reset, 666
Dakota::NonDReliability
algorithm_space_model, 670
Dakota::NonDSampling
get_parameter_sets, 675
NonDSampling, 674
num_samples, 675
sampling_reset, 675
view_aleatory_uncertain_counts, 676
view_design_counts, 676
view_epistemic_uncertain_counts, 676
view_uncertain_counts, 676
Dakota::NonDSparseGrid
INDEX

read_tabular, 769
reset, 770
reset_inactive, 771
reshape, 770
ResponseRep, 769
save, 771
update, 770
update_partial, 770
write.annotated, 769
write_table, 770
Dakota::ResultsDBAny
array_insert, 773
extract_data, 773
insert, 773
Dakota::ResultsEntry
ResultsEntry, 775
Dakota::RichExtrapVerification
converge_order, 781
converge_qoi, 781
estimate_order, 780
print_results, 780
Dakota::SNLLLeastSq
constraint1_evaluator_gn, 812
constraint2_evaluator_gn, 813
nlf2_evaluator_gn, 812
post_run, 812
Dakota::SNLLOptimizer
constraint0_evaluator, 818
constraint1_evaluator, 819
constraint2_evaluator, 819
nlf0_evaluator, 818
nlf1_evaluator, 818
nlf2_evaluator, 818
SNLLOptimizer, 817
Dakota::SeqHybridMetaIterator
extract_parameter_sets, 787
print_results, 786
run_sequential, 786
run_sequential_adaptive, 786
Dakota::SharedApproxData
~SharedApproxData, 792
buildDataOrder, 793
get_shared_data, 793
operator=, 792
SharedApproxData, 791, 792
Dakota::SharedSurfpackApproxData
SharedSurfpackApproxData, 798
Dakota::SharedVariablesData
copy, 802
Dakota::SharedVariablesDataRep
allContinuousIds, 805
SharedVariablesDataRep, 805
Dakota::SingleModel
eval_tag_prefix, 807
Dakota::SurfpackApproximation
add_anchor_to_surfdata, 825
build, 825
hessian, 825
SurfpackApproximation, 824, 825
surrogates_to_surf_data, 825
Dakota::SurrBasedLocalMinimizer
approx_subprob_constraint_eval, 833
approx_subprob_objective_eval, 833
hard_convergence_check, 832
hom_constraint_eval, 834
hom_objective_eval, 833
minimize_surrogates, 831
tr_ratio_check, 832
update_penalty, 832
Dakota::SurrBasedMinimizer
augmented_lagrangian_merit, 837
constraintViolation, 838
core_run, 836
lagrangian_merit, 837
penalty_merit, 838
print_results, 836
update_augmented_lagrange_multipliers, 837
update_filter, 837
update_lagrange_multipliers, 836
Dakota::SurrogateModel
approxBuilds, 842
force_rebuild, 841
responseMode, 842
Dakota::SysCallApplicInterface
init_communicators_checks, 844
set_communicators_checks, 844
spawn_analysis_to_shell, 845
spawn_evaluation_to_shell, 844
spawn_input_filter_to_shell, 844
spawn_output_filter_to_shell, 845
synchronous_local_analysis, 844
test_local_evaluations, 844
wait_local_evaluations, 844
Dakota::TANA3Approximation
build, 847
clear_current, 847
Dakota::TaylorApproximation
build, 848
Dakota::TestDriverInterface
  derived_map_ac, 851
  herbie, 852
  herbie1D, 851
  mc_api_run, 852
  separable_combine, 852
  shubert1D, 852
  smooth_herbie, 852
  smooth_herbie1D, 852
Dakota::TrackerHTTP
  send_data_using_get, 854
  send_data_using_post, 854
Dakota::Variables
  build_views, 863
Dakota::Verification
  core_run, 864
  print_results, 864
  dakota_dll_api.cpp, 865
  dakota_stop, 866
  dakota_dll_api.h, 866
  dakota_stop, 867
  dakota_response_to_colin_response
    Dakota::COLINApplication, 326
  dakota_stop
    dakota_dll_api.cpp, 866
    dakota_dll_api.h, 867
  dakota_tabular_io.hpp, 867
data_transform_model
  Dakota::Minimizer, 512
DataEnvironment, 352
DataEnvironmentRep, 353
DataFitSurrModel, 355
DataInterface, 366
DataMethod, 367
DataMethodRep, 368
DataModel, 380
DataModelRep, 381
DataResponses, 385
DataResponsesRep, 387
DataVariables, 390
DataVariablesRep, 391
derivVarsVector
  Dakota::ActiveSet, 285
derivative_concurrency
  Dakota::Model, 545
derived_asynch_compute_response
  Dakota::DataFitSurrModel, 360
  Dakota::HierarchSurrModel, 460
  Dakota::NestedModel, 561
derived_compute_response
  Dakota::DataFitSurrModel, 360
  Dakota::HierarchSurrModel, 460
  Dakota::NestedModel, 561
  Dakota::RecastModel, 757
derived_init_communicators
  Dakota::DataFitSurrModel, 363
  Dakota::NestedModel, 561
derived_interface
  Dakota::Model, 542
derived_map_ac
  Dakota::DirectApplicInterface, 409
  Dakota::MatlabInterface, 501
  Dakota::PythonInterface, 750
  Dakota::TestDriverInterface, 851
derived_master_overload
  Dakota::NestedModel, 561
derived_parse_inputs
  Dakota::NIDRProblemDescDB, 567
derived_synchronize
  Dakota::DataFitSurrModel, 360
  Dakota::HierarchSurrModel, 460
derived_synchronize_nowait
  Dakota::DataFitSurrModel, 361
  Dakota::HierarchSurrModel, 461
Description
  Dakota::JEGAOptimizer::Evaluator, 434
DestroyAlgorithm
  Dakota::JEGAOptimizer::Driver, 418
dg_ds_eval
  Dakota::NonDLocalReliability, 654
dimension_preference_to_anisotropic_order
  Dakota::NonDIntegration, 634
DirectApplicInterface, 406
DiscSetLbl
  Dakota, 279
DiscrepancyCorrection, 410
div_index_map
  Dakota::NestedModel, 562
dll_tester.cpp, 868
dotFDsinfo
  Dakota::DOTOptimizer, 415
dotInfo
  Dakota::DOTOptimizer, 415
dotMethod
  Dakota::DOTOptimizer, 416
dp2_dbeta_factor
  Dakota::NonDLocalReliability, 655
Driver
Dakota::JEGAOptimizer::Driver, 417

drv_index_map

Dakota::NestedModel, 562
duplication_detect

Dakota::ApplicationInterface, 301

EffGlobalMinimizer, 419
EfficientSubspaceMethod, 421
EmbedHybridMetaIterator, 425
enforce_input_rules

Dakota::FSUDesignCompExp, 445
Dakota::PSUADEDesignCompExp, 748
enroll

Dakota::GetLongOpt, 452
Environment, 426

Dakota::Environment, 428–430
estimate_derivatives

Dakota::Model, 545
estimate_message_lengths

Dakota::Model, 544
estimate_order

Dakota::RichExtrapVerification, 780
eval_tag_prefix

Dakota::Interface, 469
Dakota::Iterator, 481
Dakota::Model, 543
Dakota::RecastModel, 758
Dakota::SingleModel, 807
eval_x

Dakota::NomadOptimizer::Evaluator, 432
evalInterfaceIds

Dakota::ParamResponsePair, 714
Evaluate

Dakota::JEGAOptimizer::Evaluator, 435, 436
evaluate_parameter_sets

Dakota::Analyzer, 290
evaluation_available

Dakota::COLINApplication, 325
evaluation_cache

Dakota::Model, 543
evaluation_id

Dakota::DataFitSurfModel, 363
Dakota::HierarchSurfModel, 461
Dakota::NestedModel, 561
Evaluator

Dakota::JEGAOptimizer::Evaluator, 433, 434
Dakota::NomadOptimizer::Evaluator, 431
EvaluatorCreator

Dakota::JEGAOptimizer::EvaluatorCreator, 437
ExecutableEnvironment, 438

ExperimentData, 439
export_point

Dakota::DataFitSurfModel, 364
extract_data

Dakota::ResultsDBAny, 773
extract_parameter_sets

Dakota::SeqHybridMetaIterator, 787
ExtractAllData

Dakota::JEGAOptimizer::Driver, 418

FDstep1

Dakota::Model, 545
FDstep2

Dakota::Model, 545
FIELD_NAMES

Dakota, 140
FSUDesignCompExp, 442

Dakota::FSUDesignCompExp, 444
fdGradStepSize

Dakota::Model, 547
fdHessByFnStepSize

Dakota::Model, 547
fdHessByGradStepSize

Dakota::Model, 547
filtered_interface_list

Dakota::LibraryEnvironment, 499
filtered_model_list

Dakota::LibraryEnvironment, 500
finalize

Dakota::Approximation, 311
Dakota::PecosApproximation, 723
finalize_run

Dakota::Iterator, 479
Dakota::LeastSq, 497
Dakota::Minimizer, 511
Dakota::NonD, 588
Dakota::Optimizer, 693
find_optimum

Dakota::APPSOptimizer, 321
Dakota::COLINOptimizer, 328
Dakota::JEGAOptimizer, 493
flush

Dakota, 135
Dakota::CommandShell, 334
force_rebuild

Dakota::SurrogateModel, 841
ForkApplicInterface, 440
fpinit_ASIL

library_mode.cpp, 871
main.cpp, 874
free_iterator
   Dakota::IteratorScheduler, 486
free_iterator_parallelism
   Dakota::IteratorScheduler, 485
functionGradients
   Dakota::ResponseRep, 771
functionSurfaces
   Dakota::ApproximationInterface, 317
G1
   Dakota::CONMINOptimizer, 341
G2
   Dakota::CONMINOptimizer, 342
GPmodel_apply
   Dakota::GaussProcApproximation, 450
GaussProcApproximation
   Dakota::GaussProcApproximation, 445, 449
get_approx
   Dakota::Approximation, 312
get_best_sample
   Dakota::EffGlobalMinimizer, 421
get.confidence_intervals
   Dakota::LeastSq, 497
get_constraints
   Dakota::Constraints, 352
get_db
   Dakota::ProblemDescDB, 732
get_environment
   Dakota::Environment, 430
get_interface
   Dakota::Interface, 470
get_iterator
   Dakota::Iterator, 481
get_model
   Dakota::Model, 545
get_npath
   Dakota, 135
get_parameter_sets
   Dakota::NonDSampling, 675
get_shared_data
   Dakota::SharedApproxData, 793
GetBestMOSolutions
   Dakota::JEGAOptimizer, 492
GetBestSOsolutions
   Dakota::JEGAOptimizer, 492
GetBestSolutions
   Dakota::JEGAOptimizer, 492
GetDescription
   Dakota::JEGAOptimizer::Evaluator, 436
GetLongOpt, 450
   Dakota::GetLongOpt, 452
GetName
   Dakota::JEGAOptimizer::Evaluator, 436
GetNumberLinearConstraints
   Dakota::JEGAOptimizer::Evaluator, 435
GetNumberNonLinearConstraints
   Dakota::JEGAOptimizer::Evaluator, 435
getRmax
   Dakota, 136
gedist
   Dakota, 136
gnewton_set_recast
   Dakota::Minimizer, 512
Graphics, 453
GridApplicInterface, 456
hard_convergence_check
   Dakota::SurrBasedLocalMinimizer, 832
herbie
   Dakota::TestDriverInterface, 852
herbie1D
   Dakota::TestDriverInterface, 851
hessian
   Dakota::SurfpackApproximation, 825
HierarchSurrModel, 457
hom.constraint.eval
   Dakota::SurrBasedLocalMinimizer, 834
hom.objective_eval
   Dakota::SurrBasedLocalMinimizer, 833
IC
   Dakota::CONMINOptimizer, 343
ISC
   Dakota::CONMINOptimizer, 343
id_vars_exact_compare
   Dakota, 137
import_points
   Dakota::DataFitSurrModel, 363
increment_grid_preference
   Dakota::NonDCubature, 604
increment_order
   Dakota::NonDPolynomialChaos, 662
increment_parallel_configuration
   Dakota::ParallelLibrary, 710
increment_reference
   Dakota::NonDCubature, 605
incrementSpecification_sequence
   Dakota::NonDExpansion, 613
init_communicators
Dakota::EfficientSubspaceMethod, 424
Dakota::Model, 543
Dakota::ParallelLibrary, 710
init_communicators_checks
Dakota::ApplicationInterface, 300
Dakota::DirectApplicInterface, 409
Dakota::ProcessHandleApplicInterface, 738
Dakota::SysCallApplicInterface, 844
init_evaluation_concurrency
Dakota::IteratorScheduler, 486, 487
init_iterator
Dakota::IteratorScheduler, 485
init_iterator_parallelism
Dakota::IteratorScheduler, 486
init_mpi_comm
Dakota::ParallelLibrary, 710
init_serial
Dakota::ApplicationInterface, 299
Dakota::Model, 544
init_serial_iterators
Dakota::IteratorScheduler, 485
initial_points
Dakota::JEGAOptimizer, 493, 494
initial_taylor_series
Dakota::NonDLocalReliability, 652
initialize
Dakota::NonDAdaptImpSampling, 593
Dakota::RecastModel, 757
initialize_class_data
Dakota::NonDLocalReliability, 652
initialize_export
Dakota::DataFitSurrModel, 364
initialize_final_statistics
Dakota::NonD, 588
initialize_graphics
Dakota::Iterator, 479
initialize_grid
Dakota::NonDQuadrature, 666
initialize_h
Dakota::Model, 545
initialize_level_data
Dakota::NonDLocalReliability, 653
initialize_mpp_search_data
Dakota::NonDLocalReliability, 653
initialize_random_variable_parameters
Dakota::NonD, 588
initialize_random_variable_types
Dakota::NonD, 588
initialize_random_variables
Dakota::NonD, 587
initialize_run
Dakota::Analyzer, 289
Dakota::Iterator, 478
Dakota::LeastSq, 496
Dakota::Minimizer, 510
Dakota::NonD, 587
Dakota::Optimizer, 693
initialize_scaling
Dakota::Minimizer, 515
initialize_variables_and_constraints
Dakota::APPSOptimizer, 322
insert
Dakota::ResultsDBAny, 773
intCntlParmArray
Dakota::DOTOptimizer, 416
Interface, 461
Dakota::Interface, 468
interface_id
Dakota::Model, 542
isReadyForWork
Dakota::APPSEvalMgr, 319
Iterator, 470
Dakota::Iterator, 476, 477
IteratorScheduler, 483
Dakota::IteratorScheduler, 485
JEGAOptimizer, 488
Dakota::JEGAOptimizer, 490
JEGAOptimizer.cpp, 869
JEGAOptimizer.hpp, 869
JEGAOptimizer::Driver, 417
JEGAOptimizer::Evaluator, 432
JEGAOptimizer::EvaluatorCreator, 437

kw_1
Dakota, 140, 214
kw_10
Dakota, 142, 215
kw_100
Dakota, 163, 232
kw_101
Dakota, 163, 232
kw_102
Dakota, 163, 232
kw_103
Dakota, 163, 232
kw_104
Dakota, 163, 232
kw_105
Dakota, 192, 257
kw_239
Dakota, 193, 257
kw_24
Dakota, 145, 218
kw_240
Dakota, 193, 257
kw_241
Dakota, 257
kw_242
Dakota, 193, 257
kw_243
Dakota, 193, 257
kw_244
Dakota, 194, 258
kw_245
Dakota, 194, 258
kw_246
Dakota, 194, 258
kw_247
Dakota, 194, 258
kw_248
Dakota, 194, 258
kw_249
Dakota, 195, 259
kw_25
Dakota, 145, 218
kw_250
Dakota, 195, 259
kw_251
Dakota, 195, 259
kw_252
Dakota, 195, 259
kw_253
Dakota, 195, 259
kw_254
Dakota, 195, 260
kw_255
Dakota, 196, 260
kw_256
Dakota, 196, 260
kw_257
Dakota, 196
kw_258
Dakota, 196, 260
kw_259
Dakota, 196, 260
kw_26
Dakota, 146, 218
kw_260
Dakota, 197, 261
kw_261
Dakota, 197, 261
kw_262
Dakota, 197, 261
kw_263
Dakota, 197, 261
kw_264
Dakota, 197, 261
kw_265
Dakota, 198, 261
kw_266
Dakota, 198, 262
kw_267
Dakota, 198, 262
kw_268
Dakota, 199, 262
kw_269
Dakota, 199, 262
kw_27
Dakota, 146, 218
kw_270
Dakota, 199, 262
kw_271
Dakota, 199, 262
kw_272
Dakota, 200, 263
kw_273
Dakota, 200, 263
kw_274
Dakota, 200, 263
kw_275
Dakota, 201, 263
kw_276
Dakota, 201, 263
kw_277
Dakota, 201, 263
kw_278
Dakota, 201, 264
kw_279
Dakota, 202, 264
kw_28
Dakota, 146, 218
kw_280
Dakota, 202, 264
kw_281
Dakota, 202, 264
kw_282
Dakota, 274
kw_327
Dakota, 274
kw_328
Dakota, 275
kw_329
Dakota, 275
kw_33
Dakota, 147, 219
kw_330
Dakota, 275
kw_331
Dakota, 275
kw_332
Dakota, 275
kw_333
Dakota, 276
kw_334
Dakota, 276
kw_335
Dakota, 276
kw_336
Dakota, 276
kw_337
Dakota, 277
kw_338
Dakota, 277
kw_339
Dakota, 277
kw_34
Dakota, 148, 219
kw_341
Dakota, 277
kw_35
Dakota, 148, 220
kw_36
Dakota, 148, 220
kw_37
Dakota, 148, 220
kw_38
Dakota, 148, 220
kw_39
Dakota, 149, 221
kw_4
Dakota, 141, 214
kw_40
Dakota, 149, 221
kw_41
Dakota, 149, 221
kw_42
Dakota, 149, 221
kw_43
Dakota, 149, 221
kw_44
Dakota, 150, 221
kw_45
Dakota, 150, 222
kw_46
Dakota, 150, 222
kw_47
Dakota, 150, 222
kw_48
Dakota, 150, 222
kw_49
Dakota, 150, 222
kw_5
Dakota, 141, 214
kw_50
Dakota, 151, 222
kw_51
Dakota, 151, 223
kw_52
Dakota, 151, 223
kw_53
Dakota, 151, 223
kw_54
Dakota, 151, 223
kw_55
Dakota, 151, 223
kw_56
Dakota, 152, 223
kw_57
Dakota, 152, 224
kw_58
Dakota, 152, 224
kw_59
Dakota, 152, 224
kw_6
Dakota, 141, 215
kw_60
Dakota, 153, 224
kw_61
Dakota, 153, 224
kw_62
Dakota, 153, 224
kw_63
Dakota, 153, 225
kw_64
Dakota, 154, 225
kw.65
Dakota, 154, 225
kw.66
Dakota, 154, 225
kw.67
Dakota, 154, 225
kw.68
Dakota, 154, 225
kw.69
Dakota, 155, 226
kw.7
Dakota, 142, 215
kw.70
Dakota, 155, 226
kw.71
Dakota, 155, 226
kw.72
Dakota, 155, 226
kw.73
Dakota, 156, 226
kw.74
Dakota, 157, 227
kw.75
Dakota, 157, 227
kw.76
Dakota, 157, 227
kw.77
Dakota, 157, 227
kw.78
Dakota, 158, 227
kw.79
Dakota, 158, 228
kw.8
Dakota, 142, 215
kw.80
Dakota, 158, 228
kw.81
Dakota, 159, 228
kw.82
Dakota, 159, 228
kw.83
Dakota, 159, 228
kw.84
Dakota, 159, 229
kw.85
Dakota, 160, 229
kw.86
Dakota, 160, 229
kw.87
Dakota, 160, 229
kw.88
Dakota, 160, 229
kw.89
Dakota, 161, 230
kw.9
Dakota, 142, 215
kw.90
Dakota, 161, 230
kw.91
Dakota, 161, 230
kw.92
Dakota, 161, 230
kw.93
Dakota, 161, 230
kw.94
Dakota, 161, 230
kw.95
Dakota, 162, 231
kw.96
Dakota, 162, 231
kw.97
Dakota, 162, 231
kw.98
Dakota, 162, 231
kw.99
Dakota, 162, 231

lagrangian_merit
Dakota::SurrBasedMinimizer, 837
LeastSq, 494
Dakota::LeastSq, 496
library_mode.cpp, 870
callback_function, 872
fpinit ASL, 871
main, 872
parallel input, 873
parallel interface plugin, 872
run_dakota_data, 871
run_dakota mixed, 871
run_dakota parse, 871
serial input, 873
serial interface plugin, 872
library_split.cpp, 873
LibraryEnvironment, 498
Dakota::LibraryEnvironment, 499
likelihoodScale
Dakota::NonDDREAMBayesCalibration, 609
lin_coeffs modify n2s
my_cp
  Dakota, 135
N1
  Dakota::CONMINOptimizer, 341
N2
  Dakota::CONMINOptimizer, 341
N3
  Dakota::CONMINOptimizer, 341
N4
  Dakota::CONMINOptimizer, 341
N5
  Dakota::CONMINOptimizer, 341
NCSUOptimizor, 553
  Dakota::NCSUOptimizor, 555
NIDRProblemDescDB, 564
NL2Res, 568
NL2SOLLeasitSq, 568
NLPLPLOptimizor, 571
NLSSOLLesitSq, 575
  Dakota::NLSSOLLesitSq, 577
NPSOLOptimizor, 685
  Dakota::NPSOLOptimizor, 687
NUMBER_OF_FIELDS
  Dakota, 140
Name
  Dakota::JEGAOptimizor::Evaluator, 434
needRespTrans_byvars
  Dakota::Minimizor, 513
NestedModel, 556
new_dataset
  Dakota::Graphics, 455
nl0 evaluato
  Dakota::SNLLOptimizor, 818
nl1 evaulator
  Dakota::SNLLOptimizor, 818
nl2 evaluato
  Dakota::SNLLOptimizor, 818
nl2 evaluato gn
  Dakota::SNLLeasitSq, 812
NoDBBaseConstrucator, 577
NomadOptimizor, 578
  Dakota::NomadOptimizor, 579
NomadOptimizor::Evaluator, 430
NonD, 580
NonDAdaptImpSampling, 590
  Dakota::NonDAdaptImpSampling, 592
NonDAdaptiveSampling, 594
  Dakota::NonDAdaptiveSampling, 598
NonDBayesCalibratio
  Dakota::NonDBayesCalibratio, 600
NonDCalibratio
  Dakota::NonDCalibratio, 602
NonDCubature, 603
  Dakota::NonDCubature, 604
NonDDREAMBayesCalibratio
  Dakota::NonDDREAMBayesCalibratio, 607
NonDExpansion, 609
NonDGPImpSampling, 624
  Dakota::NonDGPImpSampling, 626
NonDGPMSABayesCalibratio
  Dakota::NonDGPMSABayesCalibratio, 627
  Dakota::NonDGPMSABayesCalibratio, 629
NonDGlobalEvidence, 615
NonDGlobalInterval, 617
NonDGlobalReliability, 620
NonDGlobalSingleInterval, 622
NonDInclemLHSSampling, 630
  Dakota::NonDInclemLHSSampling, 631
NonDIntegration, 631
  Dakota::NonDIntegration, 633
NonDInterval, 634
NonDLHSEvidence, 636
NonDLHSInterval, 637
NonDLHSSampling, 639
  Dakota::NonDLHSSampling, 640
NonDLHSSingleInterval, 641
NonDLLocalEvidence, 642
NonDLLocalInterval, 643
NonDLLocalReliability, 646
NonDLLocalSingleInterval, 655
NonDPOFDarts, 657
NonDPolynomialChaos, 659
  Dakota::NonDPolynomialChaos, 661, 662
NonDQUESOBayesCalibratio
  Dakota::NonDQUESOBayesCalibratio, 666
  Dakota::NonDQUESOBayesCalibratio, 668
NonDQuadratu
  Dakota::NonDQuadrature, 665
NonDReliability, 669
NonDSampling, 671
  Dakota::NonDSampling, 674
NonDSparseGrid, 677
  Dakota::NonDSparseGrid, 679
NonDStochCollocation, 680
  Dakota::NonDStochCollocation, 681
NonlinearCGOptimizor, 682
num_samples
  Dakota::Analyzer, 289
  Dakota::DDACEDesignCompExp, 405
  Dakota::FSUDesignCompExp, 445
Dakota::Analyzer, 289
Dakota::COLINOptimizer, 329
Dakota::DDACEDesignCompExp, 405
Dakota::FSUDesignCompExp, 444
Dakota::Iterator, 478
Dakota::LeastSq, 496
Dakota::MetaIterator, 503
Dakota::Minimizer, 510
Dakota::Optimizer, 693
Dakota::ParamStudy, 718
Dakota::PSUADesignCompExp, 748
Dakota::SNLLLeastSq, 812

pre_output
Dakota::Analyzer, 289

pre_run
Dakota::DDACEDesignCompExp, 405
Dakota::FSUDesignCompExp, 444
Dakota::Iterator, 478
Dakota::ParamStudy, 718
Dakota::PSUADesignCompExp, 747

primary_resp_differencer
Dakota::Minimizer, 515

primary_resp_reducer
Dakota::Optimizer, 694

primary_resp_weighter
Dakota::LeastSq, 497

print_distribution_map
Dakota::NonD, 589

print_distribution_mappings
Dakota::NonD, 589

print_restart
Dakota, 138

print_restart_pdb
Dakota, 138

print_restart_tabular
Dakota, 138

print_results
Dakota::Analyzer, 290
Dakota::ConcurrentMetaIterator, 336
Dakota::Iterator, 479
Dakota::LeastSq, 497
Dakota::Optimizer, 694
Dakota::PStudyDACE, 745
Dakota::RichExtrapVerification, 780
Dakota::SeqHybridMetaIterator, 786
Dakota::SurrBasedMinimizer, 836
Dakota::Verification, 864

print_sobol_indices
Dakota::Analyzer, 291

printControl
Dakota::CONMINOptimizer, 340
Dakota::DOTOptimizer, 416

prior_density
Dakota::NonDDREAMBayesCalibration, 608

prior_sample
Dakota::NonDDREAMBayesCalibration, 608

probDescDB
Dakota::Iterator, 482
Dakota::Model, 548

probability
Dakota::NonDLocalReliability, 655

problem_size
Dakota::NonDDREAMBayesCalibration, 607

problem_value
Dakota::NonDDREAMBayesCalibration, 608

ProblemDescDB, 724
Dakota::ProblemDescDB, 730, 731
ProcessApplicInterface, 732
ProcessHandleApplicInterface, 736
ProgramOptions, 739
python_convert_int
Dakota::PythonInterface, 750

PythonInterface, 748

quantify_uncertainty
Dakota::NonDDREAMBayesCalibration, 608
Dakota::NonDGPImpSampling, 626
Dakota::NonDGPSABayesCalibration, 629
Dakota::NonDIncremLHSSampling, 631
Dakota::NonDLHSSampling, 641
Dakota::NonDQUESOBayesCalibration, 668

RIA_constraint_eval
Dakota::NonDLocalReliability, 651

RIA_objective_eval
Dakota::NonDLocalReliability, 651

randomSeed
Dakota::NonDGPSABayesCalibration, 629

rawResponseMap
Dakota::Interface, 470

read
Dakota::ParamResponsePair, 714
Dakota::ResponseRep, 769, 770

read_annotated
Dakota::ResponseRep, 769

read_neutral
Dakota, 139

read_tabular
Dakota::MixedVariables, 519
<table>
<thead>
<tr>
<th>Function/Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dakota::RelaxedVariables</td>
<td>762</td>
</tr>
<tr>
<td>Dakota::ResponseRep</td>
<td>769</td>
</tr>
<tr>
<td>read_variables_responses</td>
<td></td>
</tr>
<tr>
<td>Dakota::Analyzer</td>
<td>291</td>
</tr>
<tr>
<td>realCntrlParmArray</td>
<td></td>
</tr>
<tr>
<td>Dakota::DOTOptimizer</td>
<td>416</td>
</tr>
<tr>
<td>rebuild</td>
<td></td>
</tr>
<tr>
<td>Dakota::Approximation</td>
<td>311</td>
</tr>
<tr>
<td>Dakota::PecosApproximation</td>
<td>723</td>
</tr>
<tr>
<td>rebuild_approximation</td>
<td></td>
</tr>
<tr>
<td>Dakota::ApproximationInterface</td>
<td>316</td>
</tr>
<tr>
<td>RecastBaseConstructor</td>
<td>750</td>
</tr>
<tr>
<td>RecastModel</td>
<td>751</td>
</tr>
<tr>
<td>Dakota::RecastModel</td>
<td>756</td>
</tr>
<tr>
<td></td>
<td>757</td>
</tr>
<tr>
<td>RecordResponses</td>
<td></td>
</tr>
<tr>
<td>Dakota::JEGAOptimizer::Evaluator</td>
<td>435</td>
</tr>
<tr>
<td>recv</td>
<td></td>
</tr>
<tr>
<td>Dakota::APPSEvalMgr</td>
<td>319</td>
</tr>
<tr>
<td>reduce_model</td>
<td></td>
</tr>
<tr>
<td>Dakota::Optimizer</td>
<td>694</td>
</tr>
<tr>
<td>reduced_space_uq</td>
<td></td>
</tr>
<tr>
<td>Dakota::EfficientSubspaceMethod</td>
<td>424</td>
</tr>
<tr>
<td>register_signal_handlers</td>
<td></td>
</tr>
<tr>
<td>Dakota, 135</td>
<td></td>
</tr>
<tr>
<td>RelaxedVarConstraints</td>
<td>759</td>
</tr>
<tr>
<td>Dakota::RelaxedVarConstraints</td>
<td>760</td>
</tr>
<tr>
<td>RelaxedVariables</td>
<td>760</td>
</tr>
<tr>
<td>Dakota::RelaxedVariables</td>
<td>761</td>
</tr>
<tr>
<td>repair_restart</td>
<td></td>
</tr>
<tr>
<td>Dakota, 139</td>
<td></td>
</tr>
<tr>
<td>requestVector</td>
<td></td>
</tr>
<tr>
<td>Dakota::ActiveSet</td>
<td>285</td>
</tr>
<tr>
<td>reset</td>
<td></td>
</tr>
<tr>
<td>Dakota::ResponseSet</td>
<td>770</td>
</tr>
<tr>
<td>reset_inactive</td>
<td></td>
</tr>
<tr>
<td>Dakota::ResponseRep</td>
<td>771</td>
</tr>
<tr>
<td>reshape</td>
<td></td>
</tr>
<tr>
<td>Dakota::Constraints</td>
<td>351</td>
</tr>
<tr>
<td>Dakota::MixedVarConstraints</td>
<td>518</td>
</tr>
<tr>
<td>Dakota::RelaxedVarConstraints</td>
<td>760</td>
</tr>
<tr>
<td>Dakota::ResponseRep</td>
<td>770</td>
</tr>
<tr>
<td>resize_best_resp_array</td>
<td></td>
</tr>
<tr>
<td>Dakota::Minimizer</td>
<td>514</td>
</tr>
<tr>
<td>resize_best_vars_array</td>
<td></td>
</tr>
<tr>
<td>Dakota::Minimizer</td>
<td>514</td>
</tr>
<tr>
<td>resolve_inputs</td>
<td></td>
</tr>
<tr>
<td>Dakota::ParallelLibrary</td>
<td>711</td>
</tr>
<tr>
<td>resolve_samples_symbols</td>
<td></td>
</tr>
<tr>
<td>Dakota::DDACEDesignCompExp</td>
<td>405</td>
</tr>
<tr>
<td>Response, 762</td>
<td></td>
</tr>
<tr>
<td>response_mapping</td>
<td></td>
</tr>
<tr>
<td>Dakota::Interface</td>
<td>469</td>
</tr>
<tr>
<td>Dakota::NestedModel</td>
<td>562</td>
</tr>
<tr>
<td>response_modify_n2s</td>
<td></td>
</tr>
<tr>
<td>Dakota::Minimizer</td>
<td>516</td>
</tr>
<tr>
<td>response_modify_s2n</td>
<td></td>
</tr>
<tr>
<td>Dakota::Minimizer</td>
<td>513</td>
</tr>
<tr>
<td>responseMode</td>
<td></td>
</tr>
<tr>
<td>Dakota::SurrogateModel</td>
<td>842</td>
</tr>
<tr>
<td>ResponseRep, 766</td>
<td></td>
</tr>
<tr>
<td>Dakota::ResponseRep</td>
<td>769</td>
</tr>
<tr>
<td>restart_util.cpp</td>
<td>875</td>
</tr>
<tr>
<td>main, 875</td>
<td></td>
</tr>
<tr>
<td>restore</td>
<td></td>
</tr>
<tr>
<td>Dakota::Approximation</td>
<td>311</td>
</tr>
<tr>
<td>Dakota::PecosApproximation</td>
<td>723</td>
</tr>
<tr>
<td>restore_approximation</td>
<td></td>
</tr>
<tr>
<td>Dakota::ApproximationInterface</td>
<td>317</td>
</tr>
<tr>
<td>ResultsDBAny, 772</td>
<td></td>
</tr>
<tr>
<td>ResultsEntry</td>
<td></td>
</tr>
<tr>
<td>Dakota::ResultsEntry</td>
<td>775</td>
</tr>
<tr>
<td>ResultsEntry &lt; StoredType &gt;, 774</td>
<td></td>
</tr>
<tr>
<td>ResultsID, 775</td>
<td></td>
</tr>
<tr>
<td>ResultsManager, 776</td>
<td></td>
</tr>
<tr>
<td>ResultsNames, 778</td>
<td></td>
</tr>
<tr>
<td>retrieve</td>
<td></td>
</tr>
<tr>
<td>Dakota::GetLongOpt</td>
<td>452</td>
</tr>
<tr>
<td>returns_multiple_points</td>
<td></td>
</tr>
<tr>
<td>Dakota::COLINOptimizer, 329</td>
<td></td>
</tr>
<tr>
<td>Dakota::JEGAOptimizer, 493</td>
<td></td>
</tr>
<tr>
<td>RichExtrapVerification, 779</td>
<td></td>
</tr>
<tr>
<td>run</td>
<td></td>
</tr>
<tr>
<td>Dakota::Iterator, 480</td>
<td></td>
</tr>
<tr>
<td>run_dakota_data</td>
<td></td>
</tr>
<tr>
<td>library_mode.cpp, 871</td>
<td></td>
</tr>
<tr>
<td>run_dakota_mixed</td>
<td></td>
</tr>
<tr>
<td>library_mode.cpp, 871</td>
<td></td>
</tr>
<tr>
<td>run_dakota_parse</td>
<td></td>
</tr>
<tr>
<td>library_mode.cpp, 871</td>
<td></td>
</tr>
<tr>
<td>run_iterator</td>
<td></td>
</tr>
<tr>
<td>Dakota::IteratorScheduler, 486</td>
<td></td>
</tr>
<tr>
<td>run_sequential</td>
<td></td>
</tr>
<tr>
<td>Dakota::SeqHybridMetaIterator, 786</td>
<td></td>
</tr>
<tr>
<td>run_sequential_adaptive</td>
<td></td>
</tr>
<tr>
<td>Dakota::SeqHybridMetaIterator, 786</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td></td>
</tr>
<tr>
<td>Dakota::CONMINOptimizer, 341</td>
<td></td>
</tr>
</tbody>
</table>
serve_analysis, 739
serve_analysis_async
Dakota::ProcessHandleApplicInterface, 739
serve_analysis_sync
Dakota::ApplicationInterface, 301
serve_evaluations
Dakota::ApplicationInterface, 300
serve_evaluations_async
Dakota::ApplicationInterface, 305
serve_evaluations_sync
Dakota::ApplicationInterface, 304
serve_evaluations_sync_peer
Dakota::ApplicationInterface, 305
serve_iterators
Dakota::IteratorScheduler, 487
set_apps_parameters
Dakota::APPSOptimizer, 322
set_communicators_check
Dakota::ApplicationInterface, 300
Dakota::DirectApplicInterface, 409
Dakota::ProcessHandleApplicInterface, 738
Dakota::SysCallApplicInterface, 844
set_compare
Dakota, 137
set_problem
Dakota::COLIAplication, 324
set_rng
Dakota::COLIOptimizer, 329
set_solver_parameters
Dakota::COLIOptimizer, 329
set_u_to_x_mapping
Dakota::NonD, 589
SharedApproxData, 789
Dakota::SharedApproxData, 791, 792
SharedPecosApproxData, 793
SharedSurfpackApproxData, 796
Dakota::SharedSurfpackApproxData, 798
SharedVariablesData, 798
SharedVariablesDataRep, 802
Dakota::SharedVariablesDataRep, 805
shubert1D
Dakota::TestDriverInterface, 852
SingleModel, 805
slmap
Dakota, 139
smooth_herbie
Dakota::TestDriverInterface, 852
smooth_herbie1D

Dakota, 134
SCAL
Dakota::CONMINOptimizer, 342
SCI FIELD NAMES
Dakota, 281
SIM, 282
SIM::ParallelDirectApplicInterface
test_local_evaluations, 700
SIM::SerialDirectApplicInterface
test_local_evaluations, 788
SNLLBase, 807
SNLLLeastSq, 810
SNLLOptimizer, 813
Dakota::SNLLOptimizer, 817
SOLBase, 819
sample_likelihood
Dakota::NonDDREAMBayesCalibration, 608
sampling_reset
Dakota::NonDCubature, 604
Dakota::NonDQuadrature, 666
Dakota::NonDSampling, 675
Dakota::NonDSparseGrid, 679
save
Dakota::Response, 766
Dakota::ResponseRep, 771
scale_model
Dakota::Minimizer, 512
schedule_iterators
Dakota::IteratorScheduler, 487
ScilabInterface, 781
secondary_resp_copier
Dakota::Minimizer, 512
secondary_resp_scaler
Dakota::Minimizer, 515
send_data_using_get
Dakota::TrackerHTTP, 854
send_data_using_post
Dakota::TrackerHTTP, 854
SensAnalysisGlobal, 782
separable_combine
Dakota::TestDriverInterface, 852
SeparateVariables
Dakota::JEGAOptimizer::Evaluator, 435
SeqHybridMetaIterator, 784
serial_input
library_mode.cpp, 873
serial_interface_plugin
library_mode.cpp, 872
SerialDirectApplicInterface, 787
Dakota::TestDriverInterface, 852
 solver_setup
 Dakta::COLINOptimizer, 329
 spawn_analysis_to_shell
 Dakta::SysCallApplicInterface, 845
 spawn_evaluation_impl
 Dakta::COLINApplication, 325
 spawn_evaluation_to_shell
 Dakta::SysCallApplicInterface, 844
 spawn_input_filter_to_shell
 Dakta::SysCallApplicInterface, 844
 spawn_output_filter_to_shell
 Dakta::SysCallApplicInterface, 845
 SpawnApplicInterface, 822
 split_filenames
 Dakta::ProgramOptions, 743
 start_dakota_heartbeat
 Dakta, 135
 start_grid_computing
 Dakta, 136
 stop_evaluation_servers
 Dakta::ApplicationInterface, 300
 stop_grid_computing
 Dakta, 137
 subModel
 Dakta::NestedModel, 563
 submit
 Dakta::APPSEvalMgr, 319
 subordinate_iterator
 Dakta::Model, 540
 subordinate_model
 Dakta::Model, 540
 subordinate_models
 Dakta::Model, 543
 SurfpackApproximation, 823
 Dakta::SurfpackApproximation, 824, 825
 SurfBasedGlobalMinimizer, 826
 SurfBasedLocalMinimizer, 827
 SurfBasedMinimizer, 834
 surrogate_model
 Dakta::Model, 541
 SurrogateModel, 838
 surrogates_to_surf_data
 Dakta::SurfpackApproximation, 825
 synch
 Dakta::ApplicationInterface, 299
 synch_nowait
 Dakta::ApplicationInterface, 299
 synchronize_derivatives
 Dakta::Model, 546
 synchronous_local_analyses
 Dakta::ProcessApplicInterface, 735
 synchronous_local_analysis
 Dakta::DirectApplicInterface, 409
 Dakta::GridApplicInterface, 457
 Dakta::ProcessHandleApplicInterface, 738
 Dakta::SysCallApplicInterface, 844
 synchronous_local_evaluations
 Dakta::ApplicationInterface, 303
 SysCallApplicInterface, 842
 TANA3Approximation, 845
 TaylorApproximation, 847
 terminate_modelcenter
 Dakta::ParallelLibrary, 710
 test_local_evaluations
 Dakta::SysCallApplicInterface, 844
 SIM::ParallelDirectApplicInterface, 700
 SIM::SerialDirectApplicInterface, 788
 TestDriverInterface, 848
 ToDoubleMatrix
 Dakta::JEGAOptimizer, 493
 tr_ratio_check
 Dakta::SurrBasedLocalMinimizer, 832
 TrackerHTTP, 853
 trendOrder
 Dakta::GaussProcApproximation, 450
 truth_model
 Dakta::Model, 541
 uncertain_vars_to_subspace
 Dakta::EfficientSubspaceMethod, 424
 unpack_parameters_buffer
 Dakta::Minimizer, 511
 unpack_results_buffer
 Dakta::Minimizer, 511
 update
 Dakta::ResponseRep, 770
 update_actual_model
 Dakta::DataFitSurrModel, 365
 update_approximation
 Dakta::ApproximationInterface, 315
 Dakta::DataFitSurrModel, 362
 update_augmented_lagrange_multipliers
 Dakta::SurrBasedMinimizer, 837
 update_filter
 Dakta::SurrBasedMinimizer, 837
 update_from_actual_model
 Dakta::DataFitSurrModel, 365
update_from_sub_model
   Dakota::RecastModel, 758
update_from_subordinate_model
   Dakota::Model, 541
update_lagrange_multipliers
   Dakota::SurrBasedMinimizer, 836
update_level_data
   Dakota::NonDLocalReliability, 654
update_mpp_search_data
   Dakota::NonDLocalReliability, 653
update_partial
   Dakota::ResponseRep, 770
update_penalty
   Dakota::SurrBasedLocalMinimizer, 832
update_quasi_hessians
   Dakota::Model, 547
update_response
   Dakota::Model, 546
usage
   Dakota::GetLongOpt, 453
useDerivs
   Dakota::NonDExpansion, 615
uses_method
   Dakota::Iterator, 479

VLI
   Dakota, 279
VLR
   Dakota, 279
Valueless
   Dakota::GetLongOpt, 452
var_mp_cbound
   Dakota, 281
var_mp_check_cau
   Dakota, 280
var_mp_check_ceu
   Dakota, 281
var_mp_check_cv
   Dakota, 280
var_mp_check_daui
   Dakota, 280
var_mp_check_daur
   Dakota, 280
var_mp_check_deui
   Dakota, 281
var_mp_check_deur
   Dakota, 281
var_mp_check_dset
   Dakota, 280
var_mp_drange
   Dakota, 281
variables, 855
variables_scaler
   Dakota::Minimizer, 515
variance_based_decomp
   Dakota::Analyzer, 290
vars_u_to_x_mapping
   Dakota::NonD, 589
vars_x_to_u_mapping
   Dakota::NonD, 589
Verification, 863
view_aleatory_uncertain_counts
   Dakota::NonDSampling, 676
view_design_counts
   Dakota::NonDSampling, 676
view_epistemic_uncertain_counts
   Dakota::NonDSampling, 676
view_uncertain_counts
   Dakota::NonDSampling, 676
Vlch
   Dakota, 279
volumetric_quality
   Dakota::PStudyDACE, 745

wait_local_evaluations
   Dakota::SysCallApplicInterface, 844
weight_model
   Dakota::LeastSq, 497
write
   Dakota::ParamResponsePair, 714
write.annotated
   Dakota::ResponseRep, 769
write.tabular
   Dakota::ResponseRep, 770