KeyValue

version:0.2

Generated by Doxygen 1.7.1

Sat Oct 16 2010 18:32:57

Contents

1 KeyValue **1**

1.1 Introduction 1

2 Todo List **3**

3 Directory Hierarchy **5**

3.1 Directories 5

4 Namespace Index **7**

4.1 Namespace List 7

5 Class Index **9**

5.1 Class Hierarchy 9

6 Class Index **13**

6.1 Class List 13

7 File Index **17**

7.1 File List 17

8 Directory Documentation **21**

8.1 keyvalue/key/generic/bound/ Directory Reference 21

8.2 keyvalue/bridge/ Directory Reference 22

8.3 keyvalue/key/converter/ Directory Reference 23

8.4 keyvalue/sys/exception/ Directory Reference 24

8.5 keyvalue/extern/ Directory Reference 25

8.6 keyvalue/frontend/ Directory Reference 26

8.7 keyvalue/key/generic/ Directory Reference 27

8.8 keyvalue/key/ Directory Reference 28

8.9 keyvalue/bridge/key/ Directory Reference 29

8.10 keyvalue/ Directory Reference 31

8.11 keyvalue/sys/logger/ Directory Reference 32
8.12 keyvalue/key/map/ Directory Reference 34
8.13 keyvalue/sys/message/ Directory Reference 35
8.14 keyvalue/mngt/ Directory Reference 36
8.15 keyvalue/key/generic/monotone/ Directory Reference 37
8.16 keyvalue/pattern/ Directory Reference 38
8.17 keyvalue/sys/logger/policy/ Directory Reference 39
8.18 keyvalue/key/specific/ Directory Reference 40
8.19 keyvalue/sys/ Directory Reference 42
8.20 keyvalue/util/ Directory Reference 43
8.21 keyvalue/value/ Directory Reference 44

9 Namespace Documentation **47**

9.1 keyvalue Namespace Reference 47
9.1.1 Detailed Description 49
9.1.2 Function Documentation 49
9.1.2.1 process 49
9.1.2.2 process 49
9.1.2.3 operator<< 50
9.2 keyvalue::exception Namespace Reference 50
9.2.1 Detailed Description 50
9.2.2 Function Documentation 50
9.2.2.1 operator<< 50
9.2.2.2 operator& 51
9.3 keyvalue::frontend Namespace Reference 51
9.3.1 Detailed Description 52
9.4 keyvalue::key Namespace Reference 52
9.4.1 Detailed Description 54
9.5 keyvalue::logger Namespace Reference 55
9.5.1 Detailed Description 56
9.5.2 Enumeration Type Documentation 56
9.5.2.1 Color 56
9.5.3 Function Documentation 56
9.5.3.1 resetGlobal 56
9.6 keyvalue::pattern Namespace Reference 56
9.6.1 Detailed Description 57
9.6.2 Function Documentation 57

9.6.2.1 isKey . . . 57
9.6.2.2 getKey . . . 57
9.7 keyvalue::tag Namespace Reference . . . 58
9.7.1 Detailed Description . . . 58
9.8 keyvalue::util Namespace Reference . . . 58
9.8.1 Detailed Description . . . 59
9.8.2 Function Documentation . . . 59
9.8.2.1 Lexical . . . 59
9.8.2.2 tenor2ptime . . . 59
9.9 keyvalue::value Namespace Reference . . . 60
9.9.1 Detailed Description . . . 61
9.9.2 Function Documentation . . . 61
9.9.2.1 intrusive_ptr_add_ref . . . 61
9.9.2.2 intrusive_ptr_release . . . 61
9.9.2.3 operator<< . . . 62
9.9.2.4 operator<< . . . 62
9.9.2.5 operator<< . . . 62
9.9.2.6 operator<< . . . 62
9.9.2.7 operator<< . . . 62
9.9.2.8 operator<< . . . 62
9.9.2.9 operator<< . . . 62

10 Class Documentation **63**

10.1 AddPrefix Class Reference . . . 63
10.1.1 Detailed Description . . . 63
10.2 Bounded< ElementType, Bound1, Bound2 > Class Template Reference . . . 64
10.2.1 Detailed Description . . . 65
10.2.2 Constructor & Destructor Documentation . . . 66
10.2.2.1 Bounded . . . 66
10.2.2.2 Bounded . . . 66
10.2.3 Member Function Documentation . . . 66
10.2.3.1 checkOutput . . . 66
10.2.3.2 getName . . . 67
10.2.3.3 getName . . . 67
10.2.3.4 setName . . . 67
10.3 Bridge Class Reference . . . 67
10.3.1 Detailed Description . . . 68

10.3.2 Member Function Documentation . . . 68
10.3.2.1 getCoreLibraryName . . . 68
10.3.2.2 getSimpleInfo . . . 68
10.3.2.3 getCompleteInfo . . . 68
10.4 Builder< ObjectType > Class Template Reference . . . 68
10.4.1 Detailed Description . . . 69
10.4.2 Member Function Documentation . . . 71
10.4.2.1 getName . . . 71
10.4.2.2 getResult . . . 71
10.4.2.3 getObject . . . 71
10.5 BuilderFrom< ObjectType, InputType > Class Template Reference . . . 72
10.5.1 Detailed Description . . . 72
10.5.2 Member Function Documentation . . . 72
10.5.2.1 getObject . . . 72
10.6 BuilderFromVariant< ObjectType > Class Template Reference . . . 73
10.6.1 Detailed Description . . . 73
10.6.2 Member Function Documentation . . . 73
10.6.2.1 getObject . . . 73
10.6.2.2 build . . . 74
10.7 Calculator< Tag > Class Template Reference . . . 74
10.7.1 Detailed Description . . . 75
10.7.2 Member Function Documentation . . . 76
10.7.2.1 getName . . . 76
10.7.2.2 getResult . . . 77
10.7.2.3 getValue . . . 77
10.8 Checker< ConverterType, InputType > Class Template Reference . . . 77
10.8.1 Detailed Description . . . 77
10.9 Checker< ConverterType, value::Matrix > Class Template Reference . . . 78
10.9.1 Detailed Description . . . 79
10.9.2 Member Function Documentation . . . 79
10.9.2.1 checkInput . . . 79
10.9.2.2 checkSoFar . . . 79
10.9.2.3 checkOutput . . . 79
10.9.2.4 getName . . . 79
10.9.2.5 checkSize . . . 79
10.10Checker< ConverterType, value::Single > Class Template Reference . . . 80

10.10.1 Detailed Description . . . 80
10.10.2 Member Function Documentation . . . 80
10.10.2.1 checkInput . . . 80
10.10.2.2 checkSoFar . . . 81
10.10.2.3 checkOutput . . . 81
10.10.2.4 getName . . . 81
10.11 Checker< ConverterType, value::Vector > Class Template Reference . . . 81
10.11.1 Detailed Description . . . 82
10.11.2 Member Function Documentation . . . 82
10.11.2.1 checkInput . . . 82
10.11.2.2 checkSoFar . . . 82
10.11.2.3 checkOutput . . . 82
10.11.2.4 getName . . . 82
10.11.2.5 checkSize . . . 83
10.12 Command Class Reference . . . 83
10.12.1 Detailed Description . . . 84
10.12.2 Member Function Documentation . . . 84
10.12.2.1 getCommandName . . . 84
10.12.2.2 getName . . . 84
10.12.2.3 getResult . . . 84
10.13 ConsoleLogger Class Reference . . . 85
10.13.1 Detailed Description . . . 85
10.14 DataSet Class Reference . . . 85
10.14.1 Detailed Description . . . 87
10.14.2 Constructor & Destructor Documentation . . . 87
10.14.2.1 DataSet . . . 87
10.14.3 Member Function Documentation . . . 87
10.14.3.1 getName . . . 87
10.14.3.2 add . . . 87
10.14.3.3 find . . . 87
10.14.3.4 getValue . . . 88
10.14.3.5 process . . . 88
10.14.3.6 process . . . 89
10.14.3.7 mustUpdate . . . 89
10.14.3.8 getGraph . . . 89
10.14.4 Friends And Related Function Documentation . . . 90

10.14.4.1 operator<< . . . 90
10.15 Debug Class Reference . . . 90
10.15.1 Detailed Description . . . 90
10.16 Decreasing Class Reference . . . 90
10.16.1 Detailed Description . . . 90
10.16.2 Member Function Documentation . . . 90
10.16.2.1 check . . . 90
10.16.2.2 getName . . . 91
10.17 Default< OutputType > Struct Template Reference . . . 91
10.17.1 Detailed Description . . . 91
10.18 DefaultMap< OutputType, isBasic > Struct Template Reference . . . 91
10.18.1 Detailed Description . . . 91
10.19 Device Class Reference . . . 92
10.19.1 Detailed Description . . . 93
10.19.2 Member Function Documentation . . . 93
10.19.2.1 get . . . 93
10.19.2.2 getName . . . 93
10.19.2.3 getName . . . 94
10.19.2.4 setName . . . 94
10.20 Error Class Reference . . . 94
10.20.1 Detailed Description . . . 94
10.21 Exception Class Reference . . . 94
10.21.1 Detailed Description . . . 95
10.21.2 Constructor & Destructor Documentation . . . 96
10.21.2.1 Exception . . . 96
10.21.3 Member Function Documentation . . . 96
10.21.3.1 what . . . 96
10.21.3.2 getMessage . . . 96
10.22 ExceptionImpl< StdExcept, MessageType > Class Template Reference . . . 96
10.22.1 Detailed Description . . . 98
10.22.2 Member Function Documentation . . . 98
10.22.2.1 getMessage . . . 98
10.22.2.2 what . . . 98
10.22.2.3 operator& . . . 98
10.23 Export Class Reference . . . 99
10.23.1 Detailed Description . . . 100

10.23.2 Constructor & Destructor Documentation . . . 100
10.23.2.1 Export . . . 100
10.23.3 Member Function Documentation . . . 100
10.23.3.1 getName . . . 100
10.23.3.2 getName . . . 100
10.23.3.3 setName . . . 100
10.24 LexicalToolKit::Failure Class Reference . . . 100
10.24.1 Detailed Description . . . 100
10.25 FileLogger Class Reference . . . 101
10.25.1 Detailed Description . . . 102
10.25.2 Member Function Documentation . . . 102
10.25.2.1 send . . . 102
10.25.2.2 getLevel . . . 103
10.25.2.3 setLevel . . . 103
10.25.2.4 log . . . 103
10.25.2.5 sendHeader . . . 103
10.26 XtermConsole::FileRaii Class Reference . . . 103
10.26.1 Detailed Description . . . 104
10.27 FlagMap< OutputType > Class Template Reference . . . 104
10.27.1 Detailed Description . . . 104
10.27.2 Member Function Documentation . . . 105
10.27.2.1 map . . . 105
10.27.2.2 get . . . 105
10.27.2.3 getName . . . 105
10.28 ForwardToGlobalLogger Class Reference . . . 106
10.28.1 Detailed Description . . . 106
10.28.2 Member Function Documentation . . . 106
10.28.2.1 apply . . . 106
10.28.2.2 forward . . . 107
10.29 FrontEnd Class Reference . . . 107
10.29.1 Detailed Description . . . 107
10.29.2 Member Function Documentation . . . 107
10.29.2.1 getPath . . . 107
10.29.2.2 lexicalToolKit . . . 108
10.30 Geq< ElementType > Class Template Reference . . . 108
10.30.1 Detailed Description . . . 108

10.30.2 Constructor & Destructor Documentation . . . 108
10.30.2.1 Geq . . . 108
10.30.3 Member Function Documentation . . . 109
10.30.3.1 check . . . 109
10.30.3.2 getBound . . . 109
10.30.3.3 getName . . . 109
10.31 Global< ObjectType > Class Template Reference . . . 109
10.31.1 Detailed Description . . . 110
10.31.2 Constructor & Destructor Documentation . . . 110
10.31.2.1 Global . . . 110
10.31.2.2 Global . . . 111
10.31.3 Member Function Documentation . . . 111
10.31.3.1 get . . . 111
10.31.3.2 set . . . 111
10.31.3.3 getInstance . . . 111
10.32 Global Class Reference . . . 112
10.32.1 Detailed Description . . . 113
10.32.2 Constructor & Destructor Documentation . . . 113
10.32.2.1 Global . . . 113
10.32.3 Member Function Documentation . . . 113
10.32.3.1 getName . . . 113
10.32.3.2 getName . . . 113
10.32.3.3 setName . . . 113
10.33 DataSet::Graph Struct Reference . . . 113
10.33.1 Detailed Description . . . 114
10.34 Greater< ElementType > Class Template Reference . . . 114
10.34.1 Detailed Description . . . 114
10.34.2 Constructor & Destructor Documentation . . . 115
10.34.2.1 Greater . . . 115
10.34.3 Member Function Documentation . . . 115
10.34.3.1 check . . . 115
10.34.3.2 getBound . . . 115
10.34.3.3 getName . . . 115
10.35 LexicalToolKit::Helper< From, To > Struct Template Reference . . . 115
10.35.1 Detailed Description . . . 116
10.36 IgnoreColor Class Reference . . . 116

10.36.1 Detailed Description . . . 117
10.37 IgnoreFailure Class Reference . . . 117
10.37.1 Detailed Description . . . 118
10.37.2 Member Function Documentation . . . 118
10.37.2.1 apply . . . 118
10.38 IgnorePrefix Class Reference . . . 118
10.38.1 Detailed Description . . . 118
10.39 Imports Class Reference . . . 118
10.39.1 Detailed Description . . . 119
10.39.2 Constructor & Destructor Documentation . . . 120
10.39.2.1 Imports . . . 120
10.39.3 Member Function Documentation . . . 120
10.39.3.1 getName . . . 120
10.39.3.2 getName . . . 120
10.39.3.3 setName . . . 120
10.40 Increasing Class Reference . . . 120
10.40.1 Detailed Description . . . 121
10.40.2 Member Function Documentation . . . 121
10.40.2.1 check . . . 121
10.40.2.2 getName . . . 121
10.41 Info Class Reference . . . 121
10.41.1 Detailed Description . . . 121
10.42 IsBasic< ElementType > Struct Template Reference . . . 121
10.42.1 Detailed Description . . . 122
10.43 IsBasicOrEnum< ElementType > Struct Template Reference . . . 122
10.43.1 Detailed Description . . . 122
10.44 Key Class Reference . . . 122
10.44.1 Detailed Description . . . 123
10.44.2 Constructor & Destructor Documentation . . . 123
10.44.2.1 Key . . . 123
10.44.3 Member Function Documentation . . . 124
10.44.3.1 getName . . . 124
10.44.3.2 setName . . . 124
10.45 KeyInSingle Class Reference . . . 124
10.45.1 Detailed Description . . . 126
10.45.2 Member Function Documentation . . . 126

10.45.2.1 parse . . . 126
10.45.2.2 pop . . . 126
10.45.2.3 isEmpty . . . 126
10.46 KeysInMatrix Class Reference . . . 127
10.46.1 Detailed Description . . . 128
10.46.2 Member Function Documentation . . . 128
10.46.2.1 parse . . . 128
10.46.2.2 pop . . . 128
10.46.2.3 isEmpty . . . 129
10.46.2.4 check . . . 129
10.47 KeysInVector Class Reference . . . 129
10.47.1 Detailed Description . . . 131
10.47.2 Member Function Documentation . . . 131
10.47.2.1 parse . . . 131
10.47.2.2 pop . . . 131
10.47.2.3 isEmpty . . . 132
10.48 Leq< ElementType > Class Template Reference . . . 132
10.48.1 Detailed Description . . . 132
10.48.2 Constructor & Destructor Documentation . . . 133
10.48.2.1 Leq . . . 133
10.48.3 Member Function Documentation . . . 133
10.48.3.1 check . . . 133
10.48.3.2 getBound . . . 133
10.48.3.3 getName . . . 133
10.49 Less< ElementType > Class Template Reference . . . 133
10.49.1 Detailed Description . . . 134
10.49.2 Constructor & Destructor Documentation . . . 134
10.49.2.1 Less . . . 134
10.49.3 Member Function Documentation . . . 134
10.49.3.1 check . . . 134
10.49.3.2 getBound . . . 135
10.49.3.3 getName . . . 135
10.50 Level Class Reference . . . 135
10.50.1 Detailed Description . . . 136
10.50.2 Constructor & Destructor Documentation . . . 136
10.50.2.1 Level . . . 136

10.50.3 Member Function Documentation . . . 136
10.50.3.1 getName . . . 136
10.50.3.2 getName . . . 136
10.50.3.3 setName . . . 137
10.51 LexicalToolKit Class Reference . . . 137
10.51.1 Detailed Description . . . 138
10.51.2 Member Enumeration Documentation . . . 138
10.51.2.1 IO . . . 138
10.51.3 Constructor & Destructor Documentation . . . 138
10.51.3.1 LexicalToolKit . . . 138
10.51.4 Member Function Documentation . . . 138
10.51.4.1 set . . . 138
10.51.4.2 isEnabled . . . 139
10.51.4.3 convert . . . 139
10.51.5 Member Data Documentation . . . 140
10.51.5.1 endInitList . . . 140
10.52 Logger Class Reference . . . 140
10.52.1 Detailed Description . . . 140
10.52.2 Member Function Documentation . . . 141
10.52.2.1 getLevel . . . 141
10.52.2.2 setLevel . . . 141
10.52.2.3 log . . . 141
10.53 LoggerImpl< PrefixPolicy, ColorPolicy, SafetyPolicy > Class Template Reference . . . 141
10.53.1 Detailed Description . . . 143
10.53.2 Member Function Documentation . . . 143
10.53.2.1 getLevel . . . 143
10.53.2.2 setLevel . . . 144
10.53.2.3 log . . . 144
10.53.2.4 sendHeader . . . 144
10.53.2.5 process . . . 144
10.53.2.6 send . . . 145
10.54 Logic Class Reference . . . 145
10.54.1 Detailed Description . . . 145
10.55 LogicError Class Reference . . . 145
10.55.1 Detailed Description . . . 145
10.56 Matrix< ElementType > Class Template Reference . . . 145

10.56.1 Detailed Description . . . 147
10.56.2 Constructor & Destructor Documentation . . . 147
10.56.2.1 Matrix . . . 147
10.56.2.2 Matrix . . . 147
10.56.3 Member Function Documentation . . . 147
10.56.3.1 checkSize . . . 147
10.56.3.2 getName . . . 148
10.56.3.3 getName . . . 148
10.56.3.4 setName . . . 148
10.57 Matrix Class Reference . . . 148
10.57.1 Detailed Description . . . 150
10.57.2 Constructor & Destructor Documentation . . . 150
10.57.2.1 Matrix . . . 150
10.57.3 Member Function Documentation . . . 151
10.57.3.1 getNRows . . . 151
10.57.3.2 getNCols . . . 151
10.57.3.3 resize . . . 151
10.57.3.4 operator() . . . 151
10.57.3.5 operator() . . . 152
10.57.3.6 operator== . . . 152
10.57.3.7 transposeVector . . . 152
10.57.4 Friends And Related Function Documentation . . . 152
10.57.4.1 intrusive_ptr_add_ref . . . 152
10.57.4.2 intrusive_ptr_release . . . 153
10.58 Message Class Reference . . . 153
10.58.1 Detailed Description . . . 154
10.58.2 Constructor & Destructor Documentation . . . 155
10.58.2.1 Message . . . 155
10.58.3 Member Function Documentation . . . 155
10.58.3.1 getLevel . . . 155
10.58.3.2 getString . . . 155
10.58.3.3 getPrefix . . . 155
10.58.3.4 getColor . . . 155
10.58.3.5 operator& . . . 156
10.59 MessageImpl< id > Class Template Reference . . . 156
10.59.1 Detailed Description . . . 158

10.59.2 Constructor & Destructor Documentation . . . 158
10.59.2.1 MessageImpl . . . 158
10.59.3 Member Function Documentation . . . 159
10.59.3.1 getPrefix . . . 159
10.59.3.2 getColor . . . 159
10.59.3.3 getLevel . . . 159
10.59.3.4 getString . . . 159
10.59.3.5 operator& . . . 159
10.60MonotoneBoundedVector< ElementType, Monotone, Bound1, Bound2 > Class Template Reference . . . 160
10.60.1 Detailed Description . . . 162
10.60.2 Constructor & Destructor Documentation . . . 163
10.60.2.1 MonotoneBoundedVector . . . 163
10.60.2.2 MonotoneBoundedVector . . . 163
10.60.2.3 MonotoneBoundedVector . . . 163
10.60.3 Member Function Documentation . . . 163
10.60.3.1 checkSoFar . . . 163
10.60.3.2 checkSize . . . 164
10.60.3.3 getName . . . 164
10.60.3.4 getName . . . 164
10.60.3.5 setName . . . 164
10.60.3.6 map . . . 164
10.61NoBound< ElementType > Class Template Reference . . . 165
10.61.1 Detailed Description . . . 165
10.61.2 Member Function Documentation . . . 165
10.61.2.1 check . . . 165
10.61.2.2 getBound . . . 165
10.61.2.3 getName . . . 166
10.62NoMap< OutputType > Class Template Reference . . . 166
10.62.1 Detailed Description . . . 167
10.62.2 Member Function Documentation . . . 167
10.62.2.1 map . . . 167
10.62.2.2 getName . . . 167
10.63NonMonotone Class Reference . . . 167
10.63.1 Detailed Description . . . 168
10.63.2 Member Function Documentation . . . 168
10.63.2.1 check . . . 168

10.63.2.2 getName . . . 168
10.64 Repository::NotFound Class Reference . . . 168
10.64.1 Detailed Description . . . 169
10.65 Nothing Class Reference . . . 169
10.65.1 Detailed Description . . . 169
10.66 NullDeleter Class Reference . . . 169
10.66.1 Detailed Description . . . 169
10.67 ObjectMap< ObjectType > Class Template Reference . . . 169
10.67.1 Detailed Description . . . 170
10.67.2 Member Function Documentation . . . 170
10.67.2.1 map . . . 170
10.67.2.2 getName . . . 171
10.68 ObjectPtr Class Reference . . . 171
10.68.1 Detailed Description . . . 171
10.69 Parent< T > Struct Template Reference . . . 171
10.69.1 Detailed Description . . . 171
10.70 PartialMap< OutputType > Class Template Reference . . . 172
10.70.1 Detailed Description . . . 173
10.70.2 Member Function Documentation . . . 173
10.70.2.1 map . . . 173
10.70.2.2 get . . . 174
10.70.2.3 getName . . . 174
10.71 Pattern Class Reference . . . 174
10.71.1 Detailed Description . . . 175
10.71.2 Member Function Documentation . . . 176
10.71.2.1 parse . . . 176
10.71.2.2 pop . . . 176
10.71.2.3 isEmpty . . . 176
10.72 Positive Class Reference . . . 176
10.72.1 Detailed Description . . . 178
10.72.2 Constructor & Destructor Documentation . . . 178
10.72.2.1 Positive . . . 178
10.72.3 Member Function Documentation . . . 178
10.72.3.1 checkOutput . . . 178
10.72.3.2 getName . . . 178
10.72.3.3 getName . . . 178

10.72.3.4 setName . . . 178
10.73 ProcessNow Class Reference . . . 179
10.73.1 Detailed Description . . . 180
10.73.2 Constructor & Destructor Documentation . . . 180
10.73.2.1 ProcessNow . . . 180
10.73.3 Member Function Documentation . . . 180
10.73.3.1 getName . . . 180
10.73.3.2 getName . . . 180
10.73.3.3 setName . . . 180
10.74 Processor Class Reference . . . 180
10.74.1 Detailed Description . . . 181
10.74.2 Member Function Documentation . . . 181
10.74.2.1 getName . . . 181
10.74.2.2 getResult . . . 182
10.75 Processor Class Reference . . . 182
10.75.1 Detailed Description . . . 183
10.75.2 Constructor & Destructor Documentation . . . 183
10.75.2.1 Processor . . . 183
10.75.3 Member Function Documentation . . . 183
10.75.3.1 getName . . . 183
10.75.3.2 getName . . . 183
10.75.3.3 setName . . . 184
10.76 ProcessorInstantiator< Tag > Class Template Reference . . . 184
10.76.1 Detailed Description . . . 184
10.76.2 Member Function Documentation . . . 184
10.76.2.1 getInstance . . . 184
10.77 ProcessorMngr Class Reference . . . 184
10.77.1 Detailed Description . . . 185
10.77.2 Member Function Documentation . . . 185
10.77.2.1 add . . . 185
10.77.2.2 getProcessor . . . 186
10.77.2.3 getCommand . . . 186
10.77.2.4 getCmdList . . . 186
10.77.3 Member Data Documentation . . . 186
10.77.3.1 dummy_ . . . 186
10.78 Queue Class Reference . . . 186

10.78.1 Detailed Description . . . 187
10.78.2 Member Function Documentation . . . 187
10.78.2.1 isEmpty . . . 187
10.78.2.2 pop . . . 187
10.78.2.3 remove . . . 188
10.78.2.4 putBack . . . 188
10.79 Pattern::QueueRaii Class Reference . . . 188
10.79.1 Detailed Description . . . 188
10.79.2 Constructor & Destructor Documentation . . . 188
10.79.2.1 QueueRaii . . . 188
10.80 DataSet::Record Struct Reference . . . 188
10.80.1 Detailed Description . . . 189
10.81 Report Class Reference . . . 189
10.81.1 Detailed Description . . . 189
10.82 Repository Class Reference . . . 189
10.82.1 Detailed Description . . . 190
10.82.2 Member Function Documentation . . . 190
10.82.2.1 add . . . 190
10.82.2.2 erase . . . 191
10.82.2.3 clear . . . 191
10.82.2.4 find . . . 191
10.82.2.5 getDataSet . . . 191
10.82.2.6 getSize . . . 192
10.82.2.7 getList . . . 192
10.83 Result Class Reference . . . 192
10.83.1 Detailed Description . . . 193
10.83.2 Constructor & Destructor Documentation . . . 193
10.83.2.1 Result . . . 193
10.83.2.2 Result . . . 193
10.83.2.3 Result . . . 193
10.83.3 Member Function Documentation . . . 193
10.83.3.1 operator= . . . 193
10.83.3.2 getObjectPtr . . . 194
10.83.3.3 getObjectPtr . . . 194
10.83.3.4 getValue . . . 194
10.83.3.5 getValue . . . 194

10.84 RuntimeError Class Reference . . . 194
10.84.1 Detailed Description . . . 194
10.85 Saver< SavedType > Class Template Reference . . . 194
10.85.1 Detailed Description . . . 195
10.85.2 Constructor & Destructor Documentation . . . 195
10.85.2.1 Saver . . . 195
10.85.2.2 Saver . . . 195
10.85.3 Member Function Documentation . . . 196
10.85.3.1 getSaved . . . 196
10.86 Single< ElementType > Class Template Reference . . . 196
10.86.1 Detailed Description . . . 197
10.86.2 Constructor & Destructor Documentation . . . 197
10.86.2.1 Single . . . 197
10.86.3 Member Function Documentation . . . 197
10.86.3.1 getName . . . 197
10.86.3.2 getName . . . 197
10.86.3.3 setName . . . 198
10.87 Single Class Reference . . . 198
10.87.1 Detailed Description . . . 200
10.87.2 Constructor & Destructor Documentation . . . 200
10.87.2.1 Single . . . 200
10.87.2.2 Single . . . 200
10.87.2.3 Single . . . 200
10.87.3 Member Function Documentation . . . 200
10.87.3.1 operator= . . . 200
10.87.3.2 getSize . . . 201
10.87.3.3 resize . . . 201
10.87.3.4 resize . . . 201
10.87.3.5 operator() . . . 201
10.87.3.6 operator() . . . 202
10.87.3.7 operator() . . . 202
10.87.3.8 operator() . . . 202
10.87.3.9 transpose . . . 202
10.87.3.10 getNRows . . . 202
10.87.3.11 getNCols . . . 202
10.87.3.12 operator== . . . 203

10.87.3.13 transposeVector . . . 203
10.88 StdMatrix< ElementType > Class Template Reference . . . 203
10.88.1 Detailed Description . . . 204
10.88.2 Member Typedef Documentation . . . 205
10.88.2.1 InputType_ . . . 205
10.88.2.2 OutputType_ . . . 205
10.88.3 Constructor & Destructor Documentation . . . 205
10.88.3.1 StdMatrix . . . 205
10.88.4 Member Function Documentation . . . 205
10.88.4.1 isEmpty . . . 205
10.88.4.2 pop . . . 205
10.88.4.3 insert . . . 206
10.88.4.4 getOutput . . . 206
10.88.4.5 mustUpdate . . . 206
10.89 StdSingle< ElementType > Class Template Reference . . . 206
10.89.1 Detailed Description . . . 207
10.89.2 Member Typedef Documentation . . . 207
10.89.2.1 InputType_ . . . 207
10.89.2.2 OutputType_ . . . 208
10.89.3 Constructor & Destructor Documentation . . . 208
10.89.3.1 StdSingle . . . 208
10.89.4 Member Function Documentation . . . 208
10.89.4.1 isEmpty . . . 208
10.89.4.2 pop . . . 208
10.89.4.3 insert . . . 209
10.89.4.4 getOutput . . . 209
10.89.4.5 mustUpdate . . . 209
10.90 StdVector< ElementType > Class Template Reference . . . 209
10.90.1 Detailed Description . . . 210
10.90.2 Member Typedef Documentation . . . 210
10.90.2.1 InputType_ . . . 210
10.90.2.2 OutputType_ . . . 210
10.90.3 Constructor & Destructor Documentation . . . 211
10.90.3.1 StdVector . . . 211
10.90.4 Member Function Documentation . . . 211
10.90.4.1 isEmpty . . . 211

10.90.4.2 pop . . . 211
10.90.4.3 insert . . . 211
10.90.4.4 getOutput . . . 211
10.90.4.5 mustUpdate . . . 212
10.91 StrictlyDecreasing Class Reference . . . 212
10.91.1 Detailed Description . . . 212
10.91.2 Member Function Documentation . . . 212
10.91.2.1 check . . . 212
10.91.2.2 getName . . . 213
10.92 StrictlyIncreasing Class Reference . . . 213
10.92.1 Detailed Description . . . 213
10.92.2 Member Function Documentation . . . 213
10.92.2.1 check . . . 213
10.92.2.2 getName . . . 214
10.93 StrictlyPositive Class Reference . . . 214
10.93.1 Detailed Description . . . 215
10.93.2 Constructor & Destructor Documentation . . . 215
10.93.2.1 StrictlyPositive . . . 215
10.93.3 Member Function Documentation . . . 215
10.93.3.1 checkOutput . . . 215
10.93.3.2 getName . . . 215
10.93.3.3 getName . . . 216
10.93.3.4 setName . . . 216
10.94 Table Class Reference . . . 216
10.94.1 Detailed Description . . . 217
10.94.2 Member Function Documentation . . . 217
10.94.2.1 parse . . . 217
10.94.2.2 pop . . . 218
10.94.2.3 isEmpty . . . 218
10.95 Traits< ElementType, ConverterType, MapType > Class Template Reference . . . 218
10.95.1 Detailed Description . . . 219
10.95.2 Constructor & Destructor Documentation . . . 220
10.95.2.1 Traits . . . 220
10.95.3 Member Function Documentation . . . 220
10.95.3.1 getName . . . 220
10.95.3.2 getName . . . 220

10.95.3.3 setName . . . 221
10.96 TypeName< Type > Struct Template Reference . . . 221
10.96.1 Detailed Description . . . 221
10.97 Value Class Reference . . . 221
10.97.1 Detailed Description . . . 222
10.97.2 Constructor & Destructor Documentation . . . 222
10.97.2.1 Value . . . 222
10.97.2.2 Value . . . 222
10.97.2.3 Value . . . 222
10.97.3 Member Function Documentation . . . 223
10.97.3.1 operator= . . . 223
10.97.3.2 operator== . . . 223
10.97.3.3 get . . . 223
10.97.3.4 get . . . 223
10.98 Variant Class Reference . . . 223
10.98.1 Detailed Description . . . 225
10.98.2 Constructor & Destructor Documentation . . . 225
10.98.2.1 Variant . . . 225
10.98.2.2 Variant . . . 225
10.98.2.3 Variant . . . 225
10.98.2.4 Variant . . . 225
10.98.3 Member Function Documentation . . . 226
10.98.3.1 operator= . . . 226
10.98.3.2 operator= . . . 226
10.98.3.3 is . . . 226
10.98.3.4 get . . . 227
10.98.3.5 get . . . 227
10.98.4 Friends And Related Function Documentation . . . 227
10.98.4.1 operator<< . . . 227
10.99 Vector Class Reference . . . 228
10.99.1 Detailed Description . . . 230
10.99.2 Constructor & Destructor Documentation . . . 231
10.99.2.1 Vector . . . 231
10.99.2.2 Vector . . . 231
10.99.3 Member Function Documentation . . . 231
10.99.3.1 getSize . . . 231

10.99.3.2 resize . . . 232
10.99.3.3 operator() . . . 232
10.99.3.4 operator() . . . 232
10.99.3.5 transpose . . . 232
10.99.3.6 getNRows . . . 232
10.99.3.7 getNCols . . . 233
10.99.3.8 resize . . . 233
10.99.3.9 operator() . . . 233
10.99.3.10 operator() . . . 233
10.99.3.11 operator== . . . 234
10.99.3.12 transposeVector . . . 234
10.100 Vector< ElementType > Class Template Reference . . . 234
10.100.1 Detailed Description . . . 236
10.100.2 Constructor & Destructor Documentation . . . 236
10.100.2.1 Vector . . . 236
10.100.3 Member Function Documentation . . . 236
10.100.3.1 checkSize . . . 236
10.100.3.2 getName . . . 237
10.100.3.3 getName . . . 237
10.100.3.4 setName . . . 237
10.101 VectorOutput Class Reference . . . 237
10.101.1 Detailed Description . . . 238
10.101.2 Member Enumeration Documentation . . . 238
10.101.2.1 Flag . . . 238
10.101.3 Member Function Documentation . . . 238
10.101.3.1 get . . . 238
10.101.3.2 getName . . . 239
10.101.3.3 getName . . . 239
10.101.3.4 setName . . . 239
10.102 VectorOutputBase Struct Reference . . . 239
10.102.1 Detailed Description . . . 240
10.102.2 Member Enumeration Documentation . . . 240
10.102.2.1 Flag . . . 240
10.103 Warning Class Reference . . . 240
10.103.1 Detailed Description . . . 241
10.104 WindowsConsole Class Reference . . . 241

10.104.1 Detailed Description . . . 242
10.104.2 Member Function Documentation . . . 242
10.104.2.1 send . . . 242
10.104.2.2 getLevel . . . 243
10.104.2.3 setLevel . . . 243
10.104.2.4 log . . . 243
10.104.2.5 sendHeader . . . 243
10.105 XtermColor Class Reference . . . 244
10.105.1 Detailed Description . . . 244
10.106 XtermConsole Class Reference . . . 244
10.106.1 Detailed Description . . . 246
10.106.2 Member Function Documentation . . . 246
10.106.2.1 send . . . 246
10.106.2.2 getLevel . . . 247
10.106.2.3 setLevel . . . 247
10.106.2.4 log . . . 247
10.106.2.5 sendHeader . . . 247

11 File Documentation **249**

11.1 keyvalue/bridge/Bridge.h File Reference . . . 249
11.1.1 Detailed Description . . . 250
11.2 keyvalue/bridge/key/Device.h File Reference . . . 250
11.2.1 Detailed Description . . . 251
11.3 keyvalue/bridge/key/Global.h File Reference . . . 251
11.3.1 Detailed Description . . . 252
11.4 keyvalue/util/Global.h File Reference . . . 252
11.4.1 Detailed Description . . . 253
11.5 keyvalue/bridge/key/Level.h File Reference . . . 254
11.5.1 Detailed Description . . . 254
11.6 keyvalue/bridge/Register.h File Reference . . . 254
11.6.1 Detailed Description . . . 255
11.7 keyvalue/extern/Ptime.h File Reference . . . 255
11.7.1 Detailed Description . . . 255
11.8 keyvalue/extern/SharedPtr.h File Reference . . . 256
11.8.1 Detailed Description . . . 256
11.9 keyvalue/extern/String.h File Reference . . . 256
11.9.1 Detailed Description . . . 257

11.10keyvalue/extern/Vector.h File Reference 257
11.10.1 Detailed Description 258
11.11keyvalue/key/generic/Vector.h File Reference 258
11.11.1 Detailed Description 259
11.12keyvalue/value/Vector.h File Reference 259
11.12.1 Detailed Description 261
11.13keyvalue/frontend/FrontEnd.h File Reference 261
11.13.1 Detailed Description 262
11.14keyvalue/frontend/LexicalToolKit.h File Reference 262
11.14.1 Detailed Description 264
11.14.2 Define Documentation 264
11.14.2.1 KV_LEXICAL_TOOL_KIT_PAIR 264
11.15keyvalue/frontend/LexicalToolKitPairs.h File Reference 264
11.15.1 Detailed Description 264
11.16keyvalue/frontend/Queue.h File Reference 264
11.16.1 Detailed Description 265
11.17keyvalue/key/Checker.h File Reference 265
11.17.1 Detailed Description 267
11.18keyvalue/key/converter/StdMatrix.h File Reference 267
11.18.1 Detailed Description 268
11.19keyvalue/key/converter/StdSingle.h File Reference 268
11.19.1 Detailed Description 269
11.20keyvalue/key/converter/StdVector.h File Reference 269
11.20.1 Detailed Description 270
11.21keyvalue/key/generic/bound/Geq.h File Reference 271
11.21.1 Detailed Description 271
11.22keyvalue/key/generic/bound/Greater.h File Reference 271
11.22.1 Detailed Description 272
11.23keyvalue/key/generic/bound/Leq.h File Reference 272
11.23.1 Detailed Description 273
11.24keyvalue/key/generic/bound/Less.h File Reference 273
11.24.1 Detailed Description 273
11.25keyvalue/key/generic/bound/NoBound.h File Reference 273
11.25.1 Detailed Description 274
11.26keyvalue/key/generic/Bounded.h File Reference 274
11.26.1 Detailed Description 276

11.27keyvalue/key/generic/Matrix.h File Reference 276
11.27.1 Detailed Description 276
11.28keyvalue/value/Matrix.h File Reference 276
11.28.1 Detailed Description 278
11.29keyvalue/key/generic/monotone/Decreasing.h File Reference 278
11.29.1 Detailed Description 278
11.30keyvalue/key/generic/monotone/Increasing.h File Reference 279
11.30.1 Detailed Description 279
11.31keyvalue/key/generic/monotone/NonMonotone.h File Reference 279
11.31.1 Detailed Description 279
11.32keyvalue/key/generic/monotone/StrictlyDecreasing.h File Reference 280
11.32.1 Detailed Description 280
11.33keyvalue/key/generic/monotone/StrictlyIncreasing.h File Reference 280
11.33.1 Detailed Description 281
11.34keyvalue/key/generic/MonotoneBoundedVector.h File Reference 281
11.34.1 Detailed Description 281
11.35keyvalue/key/generic/Positive.h File Reference 281
11.35.1 Detailed Description 282
11.36keyvalue/key/generic/Single.h File Reference 282
11.36.1 Detailed Description 283
11.37keyvalue/value/Single.h File Reference 283
11.37.1 Detailed Description 285
11.38keyvalue/key/generic/StrictlyPositive.h File Reference 285
11.38.1 Detailed Description 285
11.39keyvalue/key/Key.h File Reference 286
11.39.1 Detailed Description 287
11.40keyvalue/key/map/Default.h File Reference 287
11.40.1 Detailed Description 288
11.41keyvalue/key/map/FlagMap.h File Reference 288
11.41.1 Detailed Description 289
11.42keyvalue/key/map/NoMap.h File Reference 289
11.42.1 Detailed Description 291
11.43keyvalue/key/map/ObjectMap.h File Reference 291
11.43.1 Detailed Description 292
11.44keyvalue/key/map/PartialMap.h File Reference 292
11.44.1 Detailed Description 293

11.45keyvalue/key/specific/Export.h File Reference 293
11.45.1 Detailed Description 294
11.46keyvalue/key/specific/Imports.h File Reference 294
11.46.1 Detailed Description 295
11.47keyvalue/key/specific/ProcessNow.h File Reference 295
11.47.1 Detailed Description 296
11.48keyvalue/key/specific/Processor.h File Reference 296
11.48.1 Detailed Description 297
11.49keyvalue/mngt/Processor.h File Reference 297
11.49.1 Detailed Description 299
11.50keyvalue/key/specific/VectorOutput.h File Reference 299
11.50.1 Detailed Description 300
11.51keyvalue/key/Traits.h File Reference 300
11.51.1 Detailed Description 301
11.52keyvalue/keyvalue.h File Reference 301
11.52.1 Detailed Description 301
11.53keyvalue/mngt/Builder.h File Reference 301
11.53.1 Detailed Description 302
11.54keyvalue/mngt/Calculator.h File Reference 302
11.54.1 Detailed Description 303
11.55keyvalue/mngt/Command.h File Reference 303
11.55.1 Detailed Description 304
11.56keyvalue/mngt/DataSet.h File Reference 305
11.56.1 Detailed Description 306
11.57keyvalue/mngt/DeclareBuilder.h File Reference 306
11.57.1 Detailed Description 306
11.58keyvalue/mngt/DeclareCalculator.h File Reference 306
11.58.1 Detailed Description 306
11.59keyvalue/mngt/process.h File Reference 306
11.59.1 Detailed Description 307
11.60keyvalue/mngt/ProcessorMngr.h File Reference 307
11.60.1 Detailed Description 308
11.61keyvalue/mngt/Repository.h File Reference 309
11.61.1 Detailed Description 310
11.62keyvalue/pattern/KeyInSingle.h File Reference 310
11.62.1 Detailed Description 310

11.63keyvalue/pattern/KeysInMatrix.h File Reference 311
11.63.1 Detailed Description 311
11.64keyvalue/pattern/KeysInVector.h File Reference 311
11.64.1 Detailed Description 312
11.65keyvalue/pattern/Pattern.h File Reference 312
11.65.1 Detailed Description 314
11.66keyvalue/pattern/Table.h File Reference 314
11.66.1 Detailed Description 314
11.67keyvalue/sys/exception/Exception.h File Reference 315
11.67.1 Detailed Description 316
11.68keyvalue/sys/exception/KV_ASSERT.h File Reference 316
11.68.1 Detailed Description 317
11.68.2 Define Documentation 317
11.68.2.1 KV_ASSERT 317
11.69keyvalue/sys/logger/ConsoleColors.h File Reference 317
11.69.1 Detailed Description 318
11.70keyvalue/sys/logger/ConsoleLogger.h File Reference 318
11.70.1 Detailed Description 319
11.71keyvalue/sys/logger/FileLogger.h File Reference 319
11.71.1 Detailed Description 320
11.72keyvalue/sys/logger/Logger.h File Reference 320
11.72.1 Detailed Description 322
11.73keyvalue/sys/logger/LoggerImpl.h File Reference 322
11.73.1 Detailed Description 323
11.74keyvalue/sys/logger/policy/AddPrefix.h File Reference 323
11.74.1 Detailed Description 324
11.75keyvalue/sys/logger/policy/ForwardToGlobalLogger.h File Reference 324
11.75.1 Detailed Description 326
11.76keyvalue/sys/logger/policy/IgnoreColor.h File Reference 326
11.76.1 Detailed Description 327
11.77keyvalue/sys/logger/policy/IgnoreFailure.h File Reference 327
11.77.1 Detailed Description 328
11.78keyvalue/sys/logger/policy/IgnorePrefix.h File Reference 328
11.78.1 Detailed Description 328
11.79keyvalue/sys/logger/policy/XtermColor.h File Reference 329
11.79.1 Detailed Description 330

11.80keyvalue/sys/logger/StdLogger.h File Reference 330
11.80.1 Detailed Description 330
11.81keyvalue/sys/logger/WindowsConsole.h File Reference 330
11.81.1 Detailed Description 332
11.82keyvalue/sys/logger/XtermConsole.h File Reference 332
11.82.1 Detailed Description 333
11.83keyvalue/sys/message/Message.h File Reference 333
11.83.1 Detailed Description 334
11.84keyvalue/sys/message/MessageImpl.h File Reference 334
11.84.1 Detailed Description 336
11.85keyvalue/util/IsBasic.h File Reference 336
11.85.1 Detailed Description 337
11.86keyvalue/util/Lexical.h File Reference 337
11.86.1 Detailed Description 338
11.87keyvalue/util/NullDeleter.h File Reference 338
11.87.1 Detailed Description 338
11.88keyvalue/util/Saver.h File Reference 338
11.88.1 Detailed Description 339
11.89keyvalue/util/tenor2ptime.h File Reference 339
11.89.1 Detailed Description 340
11.90keyvalue/util/Util.h File Reference 340
11.90.1 Detailed Description 341
11.91keyvalue/value/Nothing.h File Reference 341
11.91.1 Detailed Description 342
11.92keyvalue/value/Parent.h File Reference 342
11.92.1 Detailed Description 342
11.93keyvalue/value/Result.h File Reference 343
11.93.1 Detailed Description 344
11.94keyvalue/value/TypeName.h File Reference 344
11.94.1 Detailed Description 346
11.95keyvalue/value/Value.h File Reference 346
11.95.1 Detailed Description 347
11.96keyvalue/value/Variant.h File Reference 347
11.96.1 Detailed Description 349

Chapter 1

KeyValue

1.1 Introduction

This is the reference manual of *KeyValue* library. Here you will find detailed information for all classes, namespaces, files, macros, etc.

Before reading this document, we recommend the reading of *KeyValue* user's manual.

Chapter 2

Todo List

Class Global< ObjectType > Consider thread-safety issues.

Chapter 3

Directory Hierarchy

3.1 Directories

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

keyvalue . . . 31
bridge . . . 22
key . . . 29
extern . . . 25
frontend . . . 26
key . . . 28
converter . . . 23
generic . . . 27
bound . . . 21
monotone . . . 37
map . . . 34
specific . . . 40
mngt . . . 36
pattern . . . 38
sys . . . 42
exception . . . 24
logger . . . 32
policy . . . 39
message . . . 35
util . . . 43
value . . . 44

Chapter 4

Namespace Index

4.1 Namespace List

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

keyvalue (All functionalities of *KeyValue* library are inside this `namespace`) 47
keyvalue::exception (All exception functionalities are inside this `namespace`) 50
keyvalue::frontend (All `class`es and functions needed by frontends belong to this namespace) 51
keyvalue::key (All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`) 52
keyvalue::logger (All logger functionalities are inside this namespace) 55
keyvalue::pattern (Key-value pattern recognition `class` are members of this namespace) . . . 56
keyvalue::tag (All Processor tags are member of this `namespace`) 58
keyvalue::util (Utility `class`es and functions are member of this `namespace`) 58
keyvalue::value (All *KeyValue* containers belong to this namespace) 60

Chapter 5

Class Index

5.1 Class Hierarchy

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

AddPrefix . . . 63
LoggerImpl< AddPrefix, XtermColor > . . . 141
XtermConsole . . . 244
LoggerImpl< AddPrefix, XtermColor, ForwardToGlobalLogger > . . . 141
WindowsConsole . . . 241
LoggerImpl<> . . . 141
FileLogger . . . 101
Bridge . . . 67
BuilderFrom< ObjectType, InputType > . . . 72
BuilderFromVariant< ObjectType > . . . 73
Checker< ConverterType, InputType > . . . 77
Checker< ConverterType, value::Matrix > . . . 78
Checker< ConverterType, value::Single > . . . 80
Checker< ConverterType, value::Vector > . . . 81
Checker< ConverterType< MapType< ElementType >::OutputType_ > > . . . 77
Traits< ElementType, ConverterType, MapType > . . . 218
Checker< StdMatrix< Default< ElementType >::OutputType_ > > . . . 77
Traits< ElementType, StdMatrix > . . . 218
Matrix< ElementType > . . . 145
Checker< StdSingle< Default< bool >::OutputType_ > > . . . 77
Traits< bool > . . . 218
Export . . . 99
Global . . . 112
ProcessNow . . . 179
Checker< StdSingle< Default< double >::OutputType_ > > . . . 77
Traits< double, StdSingle > . . . 218
Bounded< double, Geq > . . . 64
Positive . . . 176
Bounded< double, Greater > . . . 64
StrictlyPositive . . . 214
Checker< StdSingle< Default< ElementType >::OutputType_ > > . . . 77

Traits< ElementType > . . . 218
Single< ElementType > . . . 196
Traits< ElementType, StdSingle > . . . 218
Bounded< ElementType, Bound1, Bound2 > . . . 64
Checker< StdSingle< Default< logger::Logger::Device >::OutputType_ > > . . . 77
Traits< logger::Logger::Device, StdSingle > . . . 218
Device . . . 92
Checker< StdSingle< Default< string >::OutputType_ > > . . . 77
Traits< string > . . . 218
Processor . . . 182
Checker< StdSingle< Default< unsigned int >::OutputType_ > > . . . 77
Traits< unsigned int > . . . 218
Level . . . 135
Checker< StdSingle< Default< VectorOutputBase::Flag >::OutputType_ > > . . . 77
Traits< VectorOutputBase::Flag, StdSingle > . . . 218
VectorOutput . . . 237
Checker< StdVector< Default< ElementType >::OutputType_ > > . . . 77
Traits< ElementType, StdVector > . . . 218
Vector< ElementType > . . . 234
Checker< StdVector< Default< string >::OutputType_ > > . . . 77
Traits< string, StdVector > . . . 218
Imports . . . 118
Checker< StdVector< NoMap< ElementType >::OutputType_ > > . . . 77
Traits< ElementType, StdVector, NoMap > . . . 218
MonotoneBoundedVector< ElementType, Monotone, Bound1, Bound2 > . . . 160
ConsoleLogger . . . 85
DataSet . . . 85
Debug . . . 90
Decreasing . . . 90
Default< OutputType > . . . 91
Default< bool > . . . 91
Traits< bool > . . . 218
Default< double > . . . 91
Traits< double, StdSingle > . . . 218
Default< ElementType > . . . 91
Traits< ElementType > . . . 218
Traits< ElementType, StdMatrix > . . . 218
Traits< ElementType, StdSingle > . . . 218
Traits< ElementType, StdVector > . . . 218
Default< logger::Logger::Device > . . . 91
Traits< logger::Logger::Device, StdSingle > . . . 218
Default< string > . . . 91
Traits< string > . . . 218
Traits< string, StdVector > . . . 218
Default< unsigned int > . . . 91
Traits< unsigned int > . . . 218
Default< VectorOutputBase::Flag > . . . 91
Traits< VectorOutputBase::Flag, StdSingle > . . . 218

DefaultMap< OutputType, isBasic > . . . 91
Error . . . 94
Exception . . . 94
ExceptionImpl< StdExcept, MessageType > . . . 96
LexicalToolKit::Failure . . . 100
XtermConsole::FileRaii . . . 103
FlagMap< OutputType > . . . 104
ForwardToGlobalLogger . . . 106
LoggerImpl< AddPrefix, XtermColor > . . . 141
LoggerImpl< AddPrefix, XtermColor, ForwardToGlobalLogger > . . . 141
LoggerImpl<> . . . 141
FrontEnd . . . 107
Geq< ElementType > . . . 108
Geq< double > . . . 108
Global< ObjectType > . . . 109
DataSet::Graph . . . 113
Greater< ElementType > . . . 114
Greater< double > . . . 114
LexicalToolKit::Helper< From, To > . . . 115
IgnoreColor . . . 116
LoggerImpl<> . . . 141
IgnoreFailure . . . 117
IgnorePrefix . . . 118
Increasing . . . 120
Info . . . 121
IsBasic< ElementType > . . . 121
IsBasicOrEnum< ElementType > . . . 122
Key . . . 122
Traits< ElementType, ConverterType, MapType > . . . 218
Traits< bool > . . . 218
Traits< double, StdSingle > . . . 218
Traits< ElementType > . . . 218
Traits< ElementType, StdMatrix > . . . 218
Traits< ElementType, StdSingle > . . . 218
Traits< ElementType, StdVector > . . . 218
Traits< ElementType, StdVector, NoMap > . . . 218
Traits< logger::Logger::Device, StdSingle > . . . 218
Traits< string > . . . 218
Traits< string, StdVector > . . . 218
Traits< unsigned int > . . . 218
Traits< VectorOutputBase::Flag, StdSingle > . . . 218
Leq< ElementType > . . . 132
Less< ElementType > . . . 133
LexicalToolKit . . . 137
Logger . . . 140
LoggerImpl< PrefixPolicy, ColorPolicy, SafetyPolicy > . . . 141
LoggerImpl< AddPrefix, XtermColor > . . . 141
LoggerImpl< AddPrefix, XtermColor, ForwardToGlobalLogger > . . . 141
LoggerImpl<> . . . 141
Logic . . . 145
LogicError . . . 145
Matrix . . . 148

Vector . . . 228
Single . . . 198
Message . . . 153
MessageImpl< id > . . . 156
NoBound< ElementType > . . . 165
NoBound< double > . . . 165
NoMap< OutputType > . . . 166
NoMap< ElementType > . . . 166
Traits< ElementType, StdVector, NoMap > . . . 218
NonMonotone . . . 167
Repository::NotFound . . . 168
Nothing . . . 169
NullDeleter . . . 169
ObjectMap< ObjectType > . . . 169
ObjectPtr . . . 171
Parent< T > . . . 171
PartialMap< OutputType > . . . 172
Pattern . . . 174
KeyInSingle . . . 124
KeysInMatrix . . . 127
KeysInVector . . . 129
Table . . . 216
Processor . . . 180
Builder< ObjectType > . . . 68
Calculator< Tag > . . . 74
Command . . . 83
ProcessorInstantiator< Tag > . . . 184
ProcessorMngr . . . 184
Queue . . . 186
Pattern::QueueRaii . . . 188
DataSet::Record . . . 188
Report . . . 189
Repository . . . 189
Result . . . 192
RuntimeError . . . 194
Saver< SavedType > . . . 194
StdMatrix< ElementType > . . . 203
StdSingle< ElementType > . . . 206
StdVector< ElementType > . . . 209
StrictlyDecreasing . . . 212
StrictlyIncreasing . . . 213
TypeName< Type > . . . 221
Value . . . 221
Variant . . . 223
VectorOutputBase . . . 239
VectorOutput . . . 237
Warning . . . 240
XtermColor . . . 244
LoggerImpl< AddPrefix, XtermColor > . . . 141
LoggerImpl< AddPrefix, XtermColor, ForwardToGlobalLogger > . . . 141

Chapter 6

Class Index

6.1 Class List

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

AddPrefix (Defines a policy for Message prefixes, which is, to send the prefix to the underlying Logger's device) . . . 63
Bounded< ElementType, Bound1, Bound2 > (Key for bounded values) . . . 64
Bridge (Bridge's specific data) . . . 67
Builder< ObjectType > (Concrete Builder) . . . 68
BuilderFrom< ObjectType, InputType > (Base `class` for Builders that can build from *InputType*) . . . 72
BuilderFromVariant< ObjectType > (Base `class` for Builders that can build from value::Variant) . . . 73
Calculator< Tag > (Concrete Calculator) . . . 74
Checker< ConverterType, InputType > (Primary key checker `template class`) . . . 77
Checker< ConverterType, value::Matrix > (Specialization of Checker for *InputType* equal to value::Matrix) . . . 78
Checker< ConverterType, value::Single > (Specialization of Checker for *InputType* equal to value::Single) . . . 80
Checker< ConverterType, value::Vector > (Specialization of Checker for *InputType* equal to value::Vector) . . . 81
Command (Command interface) . . . 83
ConsoleLogger (Implements a console Logger) . . . 85
DataSet (A named map from keys to values) . . . 85
Debug (MessageImpl instantiation used for debug messages) . . . 90
Decreasing (Decreasing monotone `class`) . . . 90
Default< OutputType > (Meta-function which defines Traits' default map type) . . . 91
DefaultMap< OutputType, isBasic > (Meta-function which defines Traits' default map type) . . . 91
Device (Device key) . . . 92
Error (MessageImpl instantiation used for error messages) . . . 94
Exception (Base protocol `class` for all exceptions) . . . 94
ExceptionImpl< StdExcept, MessageType > (Concrete `template class` which implements Exception's pure virtual methods) . . . 96
Export (Export key) . . . 99
LexicalToolKit::Failure (This `class` is thrown when LexicalToolKit::convert() fails) . . . 100
FileLogger (Implements a file Logger) . . . 101
XtermConsole::FileRaii (XtermConsole's helper `class` for file management) . . . 103

FlagMap< OutputType > (Maps names to constants) . . . 104
ForwardToGlobalLogger (Defines a policy to apply when the Logger fails to process a Message, namely, forward it to GlobalLogger) . . . 106
FrontEnd (Front-end's specific data) . . . 107
Geq< ElementType > (Greater-than-or-equal-to bound `class`) . . . 108
Global< ObjectType > (Manager of global objects) . . . 109
Global (Global key) . . . 112
DataSet::Graph (Simplified dependency graph) . . . 113
Greater< ElementType > (Greater-than bound `class`) . . . 114
LexicalToolKit::Helper< From, To > (Helper `class` of LexicalToolKit) . . . 115
IgnoreColor (Defines a policy for Message colors, which is, to ignore them) . . . 116
IgnoreFailure (Defines a policy to apply when the logger fails, which is, to ignore the failure) . . . 117
IgnorePrefix (Defines a policy for Message prefixes, which is, to ignore them) . . . 118
Imports (Imports key) . . . 118
Increasing (Increasing monotone `class`) . . . 120
Info (MessageImpl instantiation used for info messages) . . . 121
IsBasic< ElementType > (This meta-function checks if a type is either `bool`, `double`, `string`, `ptime` or `unsigned int` or not) . . . 121
IsBasicOrEnum< ElementType > (This meta-function checks if a type is either `bool`, `double`, `string`, `ptime`, `unsigned int` or enum or not) . . . 122
Key (Base for all real keys) . . . 122
KeyInSingle (Pattern where a value::Single object, recognized as a key, is followed by any value::Value) . . . 124
KeysInMatrix (Pattern made by a value::Matrix where all elements either in the first row or in the first column are keys) . . . 127
KeysInVector (Pattern made by a value::Vector whose entries are keys followed by a value::Vector or value::Matrix) . . . 129
Leq< ElementType > (Less-than-or-equal-to bound `class`) . . . 132
Less< ElementType > (Less-than bound `class`) . . . 133
Level (Level key) . . . 135
LexicalToolKit (Manager of all lexical converters needed for front-end input/output) . . . 137
Logger (Protocol `class` which defines the interface for all types of logger) . . . 140
LoggerImpl< PrefixPolicy, ColorPolicy, SafetyPolicy > (Abstract `class` which implements main methods of concrete Loggers) . . . 141
Logic (MessageImpl instantiation used for logic error messages) . . . 145
LogicError (Base `class` for all *KeyValue* exceptions occurring at development time) . . . 145
Matrix< ElementType > (Generic key whose converter type is StdMatrix) . . . 145
Matrix (Matrix of `Variant`s) . . . 148
Message (Abstract `class` which defines the interface for all types of message) . . . 153
MessageImpl< id > (Implementation of `class` Message) . . . 156
MonotoneBoundedVector< ElementType, Monotone, Bound1, Bound2 > (Key for monotone bounded vectors) . . . 160
NoBound< ElementType > (NoBound bound `class`) . . . 165
NoMap< OutputType > (Identity map (aka, NoMap)) . . . 166
NonMonotone (NonMonotone `class`) . . . 167
Repository::NotFound (Exception thrown by get() when a DataSet is not found) . . . 168
Nothing (Empty `class` to represent empty data) . . . 169
NullDeleter (A `shared_ptr` deleter that does not do anything) . . . 169
ObjectMap< ObjectType > (Maps names to objects) . . . 169
ObjectPtr (Pointer to objects) . . . 171
Parent< T > (Primary Parent `template` meta-function) . . . 171
PartialMap< OutputType > (Partially maps names to constants) . . . 172
Pattern (Protocol `class` which defines the interface for pattern recognition `classes`) . . . 174
Positive (A general key for positive values) . . . 176

ProcessNow (ProcessNow key) . . . 179
Processor (Protocol `class` which defines Processor interface) . . . 180
Processor (Processor key) . . . 182
ProcessorInstantiator< Tag > (Instantiate a processor given its *Tag*) . . . 184
ProcessorMngr (The processor manager) . . . 184
Queue (Protocol `class` which defines the Queue interface) . . . 186
Pattern::QueueRaii (This `class` implements RAII for a frontend::Queue) . . . 188
DataSet::Record (A wrapper around a value::Value) . . . 188
Report (MessageImpl instantiation used for reporting results) . . . 189
Repository (The Repository of DataSets) . . . 189
Result (A single-valued container for ObjectPtr or Value) . . . 192
RuntimeError (Base `class` for all *KeyValue* exceptions occurring at runtime) . . . 194
Saver< SavedType > (Saves a variable reference at construction time to restore/set its value at destruction time) . . . 194
Single< ElementType > (Generic key whose converter type is StdSingle) . . . 196
Single (A 1 x 1 Matrix) . . . 198
StdMatrix< ElementType > (Converter from value::Matrix to `shared_ptr<std::vector<std::vector<`*ElementType*`>>>`) . . . 203
StdSingle< ElementType > (Converter from value::Single to *ElementType*) . . . 206
StdVector< ElementType > (Converter from value::Vector to `shared_ptr<std::vector<`*ElementType*`>>`) . . . 209
StrictlyDecreasing (StrictlyDecreasing monotone `class`) . . . 212
StrictlyIncreasing (StrictlyIncreasing monotone `class`) . . . 213
StrictlyPositive (A general key whose value is a strictly positive number) . . . 214
Table (Pattern where a value::Matrix contains the row, column and table keys) . . . 216
Traits< ElementType, ConverterType, MapType > (Real keys must derive from adequate instantiations of this `template class`) . . . 218
TypeName< Type > (Meta function which returns the name of type) . . . 221
Value (A single-valued container for Single, Vector or Matrix) . . . 221
Variant (Single-value multi-type container) . . . 223
Vector (A m x 1 or 1 x n Matrix) . . . 228
Vector< ElementType > (Generic key whose converter type is StdVector) . . . 234
VectorOutput (VectorOutput key) . . . 237
VectorOutputBase (VectorOutput's base `class`) . . . 239
Warning (MessageImpl instantiation used for warning messages) . . . 240
WindowsConsole (This `class` implements a Windows console Logger) . . . 241
XtermColor (Defines a policy for Message color, which is, to set the underlying device of the Logger to print in Message's color) . . . 244
XtermConsole (Implements an xterm console Logger) . . . 244

Chapter 7

File Index

7.1 File List

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

keyvalue/keyvalue.h (Main page of *KeyValue*'s reference manual) 301
keyvalue/bridge/Bridge.h (Declaration of `class` Bridge) 249
keyvalue/bridge/Register.h (List of reserved processors) 254
keyvalue/bridge/key/Device.h (Declaration of `class` Device) 250
keyvalue/bridge/key/Global.h (Declaration of `class` Global) 251
keyvalue/bridge/key/Level.h (Declaration of `class` Level) 254
keyvalue/extern/Ptime.h (Export `boost::ptime` to `namespace` keyvalue) 255
keyvalue/extern/SharedPtr.h (Export `boost::shared_ptr` to `namespace` keyvalue) . . . 256
keyvalue/extern/String.h (Export `std::string` to `namespace` keyvalue) 256
keyvalue/extern/Vector.h (Export `std::vector` to `namespace` keyvalue) 257
keyvalue/frontend/FrontEnd.h (Declaration of `class` FrontEnd) 261
keyvalue/frontend/LexicalToolKit.h (Definition of `class` LexicalToolKit) 262
keyvalue/frontend/LexicalToolKitPairs.h (X Macro helper for `class` LexicalToolkit) 264
keyvalue/frontend/Queue.h (Declaration of `class` Queue) 264
keyvalue/key/Checker.h (Declaration of `template class` Checker) 265
keyvalue/key/Key.h (Declaration and implementation of `class` Key) 286
keyvalue/key/Traits.h (Declaration and implementation of `template class` Traits) 300
keyvalue/key/converter/StdMatrix.h (Declaration and implementation of `template class` StdMatrix) 267
keyvalue/key/converter/StdSingle.h (Declaration and implementation of `template class` StdSingle) 268
keyvalue/key/converter/StdVector.h (Declaration and implementation of `template class` StdVector) 269
keyvalue/key/generic/Bounded.h (Declaration and implementation of `template class` Bounded) 274
keyvalue/key/generic/Matrix.h (Declaration and implementation of `template class` Matrix (generic key)) 276
keyvalue/key/generic/MonotoneBoundedVector.h (Declaration and implementation of `template class` MonotoneBoundedVector) 281
keyvalue/key/generic/Positive.h (Declaration of `class` Positive) 281
keyvalue/key/generic/Single.h (Declaration and implementation of `template class` Single (generic key)) 282
keyvalue/key/generic/StrictlyPositive.h (Declaration of `class` StrictlyPositive) 285

keyvalue/key/generic/Vector.h (Declaration and implementation of template class Vector (generic key)) . . . 258
keyvalue/key/generic/bound/Geq.h (Declaration of template class Geq) . . . 271
keyvalue/key/generic/bound/Greater.h (Declaration of template class Greater) . . . 271
keyvalue/key/generic/bound/Leq.h (Declaration of template class Leq) . . . 272
keyvalue/key/generic/bound/Less.h (Declaration of template class Less) . . . 273
keyvalue/key/generic/bound/NoBound.h (Declaration of template class NoBound) . . . 273
keyvalue/key/generic/monotone/Decreasing.h (Declaration of class Decreasing) . . . 278
keyvalue/key/generic/monotone/Increasing.h (Declaration of class Increasing) . . . 279
keyvalue/key/generic/monotone/NonMonotone.h (Declaration of class NonMonotone) . . . 279
keyvalue/key/generic/monotone/StrictlyDecreasing.h (Declaration of class StrictlyDecreasing) 280
keyvalue/key/generic/monotone/StrictlyIncreasing.h (Declaration of class StrictlyIncreasing) 280
keyvalue/key/map/Default.h (Declaration and implementation of template class Default) 287
keyvalue/key/map/FlagMap.h (Declaration and implementation of template class FlagMap) . . . 288
keyvalue/key/map/NoMap.h (Declaration of template class NoMap) . . . 289
keyvalue/key/map/ObjectMap.h (Declaration and implementation of template class ObjectMap) . . . 291
keyvalue/key/map/PartialMap.h (Declaration of template class PartialMap) . . . 292
keyvalue/key/specific/Export.h (Declaration of class Export) . . . 293
keyvalue/key/specific/Imports.h (Declaration of class Imports) . . . 294
keyvalue/key/specific/ProcessNow.h (Declaration of class ProcessNow) . . . 295
keyvalue/key/specific/Processor.h (Declaration of class Processor (key)) . . . 296
keyvalue/key/specific/VectorOutput.h (Declaration of class VectorOutput) . . . 299
keyvalue/mngt/Builder.h (Declaration of primary template class Builder) . . . 301
keyvalue/mngt/**BuilderFrom.h** . . . **??**
keyvalue/mngt/Calculator.h (Declaration of primary template class Calculator) . . . 302
keyvalue/mngt/Command.h (Declaration of class Command) . . . 303
keyvalue/mngt/DataSet.h (Declaration of class DataSet) . . . 305
keyvalue/mngt/DeclareBuilder.h (A helper file to declare Builders) . . . 306
keyvalue/mngt/DeclareCalculator.h (A helper file to declare Calculators) . . . 306
keyvalue/mngt/process.h (Declaration of function keyvalue::process()) . . . 306
keyvalue/mngt/Processor.h (Declaration of class Processor) . . . 297
keyvalue/mngt/**ProcessorInstantiator.h** . . . **??**
keyvalue/mngt/ProcessorMngr.h (Declaration of class ProcessorMngr) . . . 307
keyvalue/mngt/Repository.h (Declaration of class Repository) . . . 309
keyvalue/pattern/KeyInSingle.h (Declaration of class KeyInSingle) . . . 310
keyvalue/pattern/KeysInMatrix.h (Declaration of class KeysInMatrix) . . . 311
keyvalue/pattern/KeysInVector.h (Declaration of class KeysInVector) . . . 311
keyvalue/pattern/Pattern.h (Declaration of class Pattern and functions keyvalue::pattern::isKey() and keyvalue::pattern::getKey()) . . . 312
keyvalue/pattern/Table.h (Declaration of class Table) . . . 314
keyvalue/sys/exception/Exception.h (Declaration of classes Exception, RuntimeError and LogicError) . . . 315
keyvalue/sys/exception/KV_ASSERT.h (Implementation of macro KV_ASSERT) . . . 316
keyvalue/sys/logger/ConsoleColors.h (Definition of enum Color) . . . 317
keyvalue/sys/logger/ConsoleLogger.h (Declaration of class ConsoleLogger) . . . 318
keyvalue/sys/logger/FileLogger.h (Declaration of class FileLogger) . . . 319
keyvalue/sys/logger/Logger.h (Declaration of class Logger) . . . 320
keyvalue/sys/logger/LoggerImpl.h (Declaration and implementation of template class LoggerImpl) . . . 322
keyvalue/sys/logger/StdLogger.h (Declaration of class StdLogger) . . . 330
keyvalue/sys/logger/WindowsConsole.h (Declaration of class WindowsConsole) . . . 330
keyvalue/sys/logger/XtermConsole.h (Declaration of class XtermConsole) . . . 332

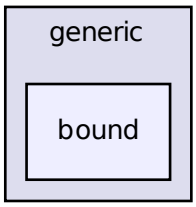
keyvalue/sys/logger/policy/AddPrefix.h (Declaration of class AddPrefix) 323
keyvalue/sys/logger/policy/ForwardToGlobalLogger.h (Declaration of class ForwardToGlobalLogger) 324
keyvalue/sys/logger/policy/IgnoreColor.h (Declaration of class IgnoreColor) 326
keyvalue/sys/logger/policy/IgnoreFailure.h (Declaration of class IgnoreFailure) 327
keyvalue/sys/logger/policy/IgnorePrefix.h (Declaration of class IgnorePrefix) 328
keyvalue/sys/logger/policy/XtermColor.h (Declaration of class XtermColor) 329
keyvalue/sys/message/Message.h (Declaration of class Message) 333
keyvalue/sys/message/MessageImpl.h (Declaration of classes MessageImpl, Error, Info, Warning, Report and Debug) 334
keyvalue/util/Global.h (Declaration of template class Global) 252
keyvalue/util/IsBasic.h (Declaration and implementation of template class IsBasic) . . . 336
keyvalue/util/Lexical.h (Declaration of template function Lexical) 337
keyvalue/util/NullDeleter.h (Declaration of class NullDeleter) 338
keyvalue/util/Saver.h (Definition and implementation of template class Saver) 338
keyvalue/util/tenor2ptime.h (Declaration of function tenor2ptime()) 339
keyvalue/util/Util.h (Documentation of namespace util) 340
keyvalue/value/Matrix.h (Declaration of class Matrix (container)) 276
keyvalue/value/Nothing.h (Declaration of class Nothing) 341
keyvalue/value/Parent.h (Declaration of template class Parent) 342
keyvalue/value/Result.h (Declaration of classes Result and ObjectPtr) 343
keyvalue/value/Single.h (Declaration of class Single (container)) 283
keyvalue/value/TypeName.h (Definition of template class TypeName) 344
keyvalue/value/Value.h (Declaration and (partial) implementation of class Value) 346
keyvalue/value/Variant.h (Declaration and (partial) implementation of class Variant) 347
keyvalue/value/Vector.h (Declaration of class Vector (container)) 259

Chapter 8

Directory Documentation

8.1 keyvalue/key/generic/bound/ Directory Reference

Directory dependency graph for keyvalue/key/generic/bound/:



Files

- file Geq.h

  Declaration of `template class` *Geq.*

- file Greater.h

  Declaration of `template class` *Greater.*

- file Leq.h

  Declaration of `template class` *Leq.*

- file Less.h

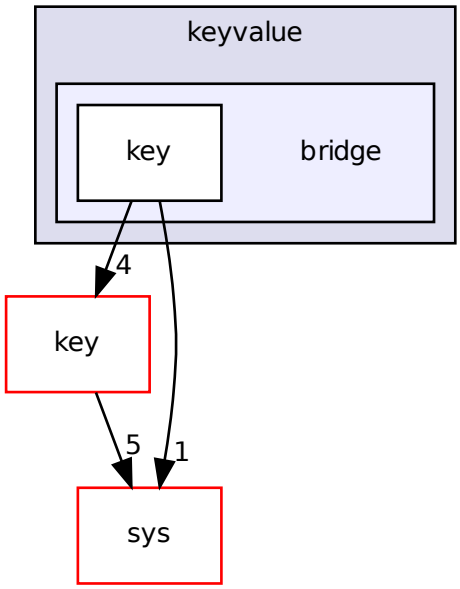
  Declaration of `template class` *Less.*

- file NoBound.h

Declaration of `template class` *NoBound.*

8.2 keyvalue/bridge/ Directory Reference

Directory dependency graph for keyvalue/bridge/:



Directories

- directory key

Files

- file Bridge.h

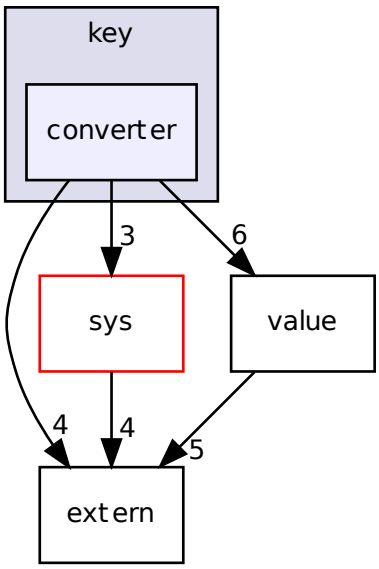
  Declaration of `class` *Bridge.*

- file Register.h

  List of reserved processors.

8.3 keyvalue/key/converter/ Directory Reference

Directory dependency graph for keyvalue/key/converter/:



Files

- file StdMatrix.h

  Declaration and implementation of `template class` *StdMatrix.*

- file StdSingle.h

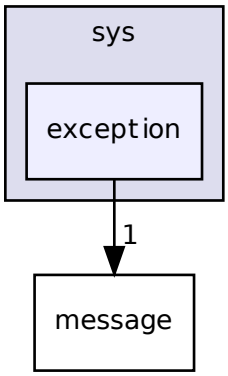
  Declaration and implementation of `template class` *StdSingle.*

- file StdVector.h

  Declaration and implementation of `template class` *StdVector.*

8.4 keyvalue/sys/exception/ Directory Reference

Directory dependency graph for keyvalue/sys/exception/:



Files

- file Exception.h

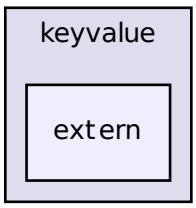
  Declaration of `classes` Exception, RuntimeError and LogicError.

- file KV_ASSERT.h

  Implementation of macro KV_ASSERT.

8.5 keyvalue/extern/ Directory Reference

Directory dependency graph for keyvalue/extern/:



Files

- file Ptime.h

  Export `boost::ptime` *to* `namespace` *keyvalue.*

- file SharedPtr.h

  Export `boost::shared_ptr` *to* `namespace` *keyvalue.*

- file String.h

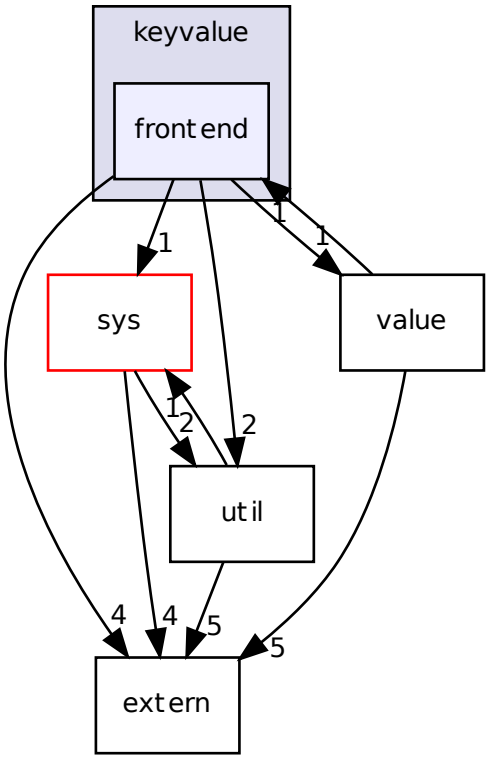
  Export `std::string` *to* `namespace` *keyvalue.*

- file Vector.h

  Export `std::vector` *to* `namespace` *keyvalue.*

8.6 keyvalue/frontend/ Directory Reference

Directory dependency graph for keyvalue/frontend/:



Files

- file FrontEnd.h

  Declaration of `class` *FrontEnd.*

- file LexicalToolKit.h

  Definition of `class` *LexicalToolKit.*

- file LexicalToolKitPairs.h

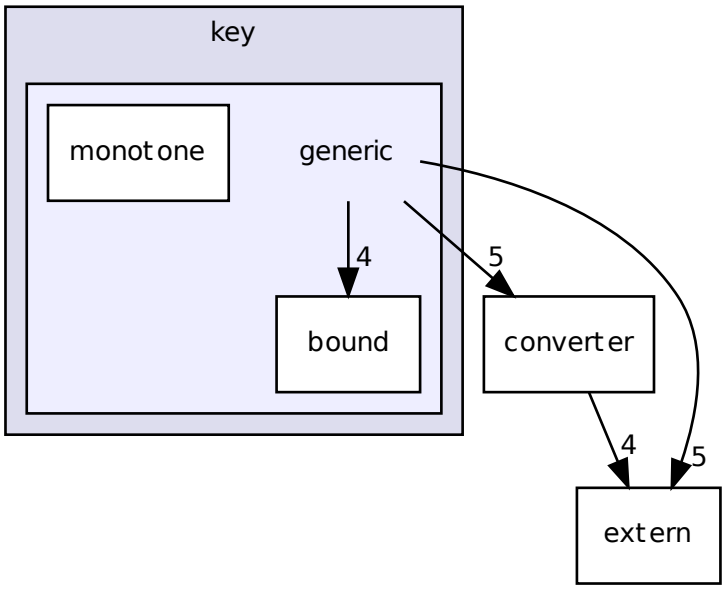
  X Macro helper for `class` *LexicalToolkit.*

- file Queue.h

  Declaration of `class` *Queue.*

8.7 keyvalue/key/generic/ Directory Reference

Directory dependency graph for keyvalue/key/generic/:



Directories

- directory bound
- directory monotone

Files

- file Bounded.h

  Declaration and implementation of `template class` *Bounded.*

- file Matrix.h

  Declaration and implementation of `template class` *Matrix (generic key).*

- file MonotoneBoundedVector.h

  Declaration and implementation of `template class` *MonotoneBoundedVector.*

- file Positive.h

  Declaration of `class` *Positive.*

- file Single.h

  Declaration and implementation of `template class` *Single (generic key).*

- file StrictlyPositive.h

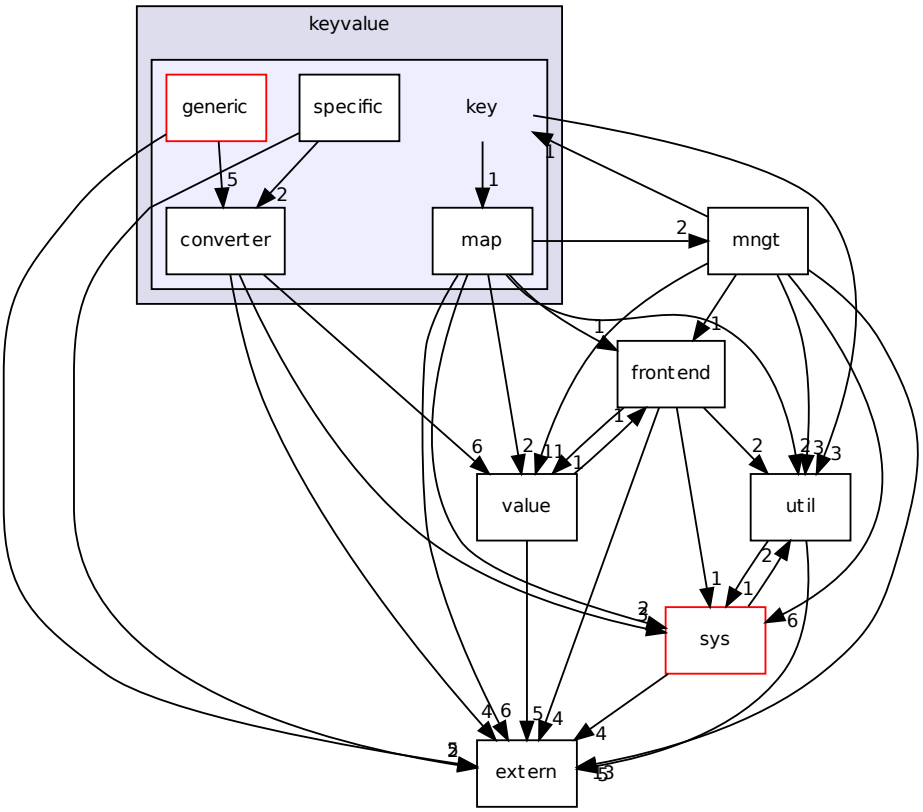
  Declaration of `class` *StrictlyPositive.*

- file Vector.h

  Declaration and implementation of `template class` *Vector (generic key).*

8.8 keyvalue/key/ Directory Reference

Directory dependency graph for keyvalue/key/:



Directories

- directory converter
- directory generic

- directory map
- directory specific

Files

- file Checker.h

  Declaration of `template class` Checker.

- file Key.h

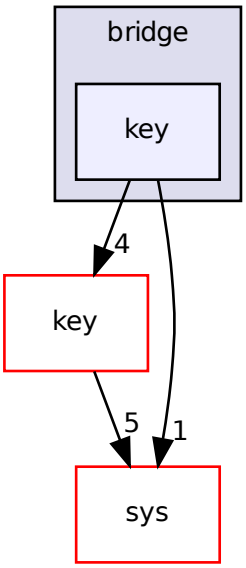
  Declaration and implementation of `class` Key.

- file Traits.h

  Declaration and implementation of `template class` Traits.

8.9 keyvalue/bridge/key/ Directory Reference

Directory dependency graph for keyvalue/bridge/key/:



Files

- file Device.h

  Declaration of `class` Device.

- file Global.h

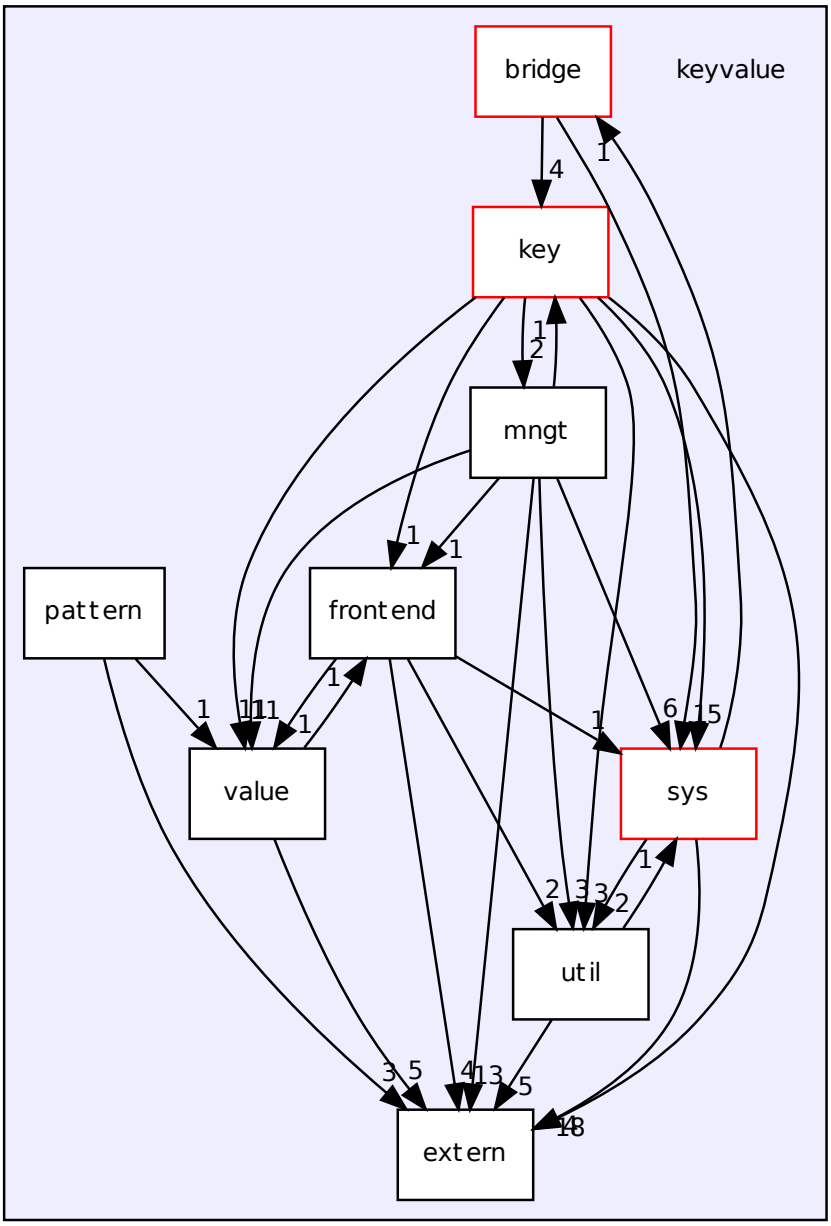
Declaration of `class` *Global.*

- file Level.h

Declaration of `class` *Level.*

8.10 keyvalue/ Directory Reference

Directory dependency graph for keyvalue/:



Directories

- directory bridge
- directory extern
- directory frontend
- directory key
- directory mngt
- directory pattern
- directory sys
- directory util
- directory value

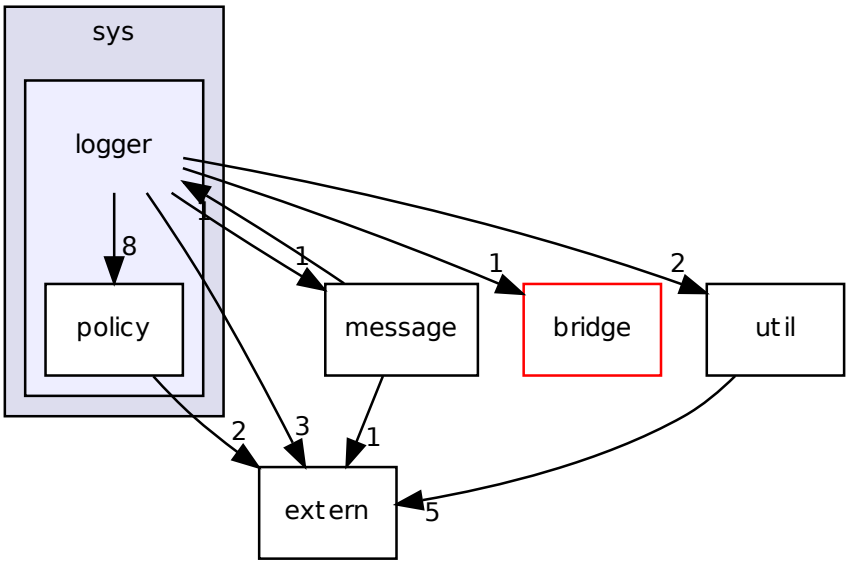
Files

- file keyvalue.h

  Main page of KeyValue's reference manual.

8.11 keyvalue/sys/logger/ Directory Reference

Directory dependency graph for keyvalue/sys/logger/:



Directories

- directory policy

Files

- file ConsoleColors.h

  Definition of `enum` *Color.*

- file ConsoleLogger.h

  Declaration of `class` *ConsoleLogger.*

- file FileLogger.h

  Declaration of `class` *FileLogger.*

- file Logger.h

  Declaration of `class` *Logger.*

- file LoggerImpl.h

  Declaration and implementation of `template class` *LoggerImpl.*

- file StdLogger.h

  Declaration of `class` *StdLogger.*

- file WindowsConsole.h

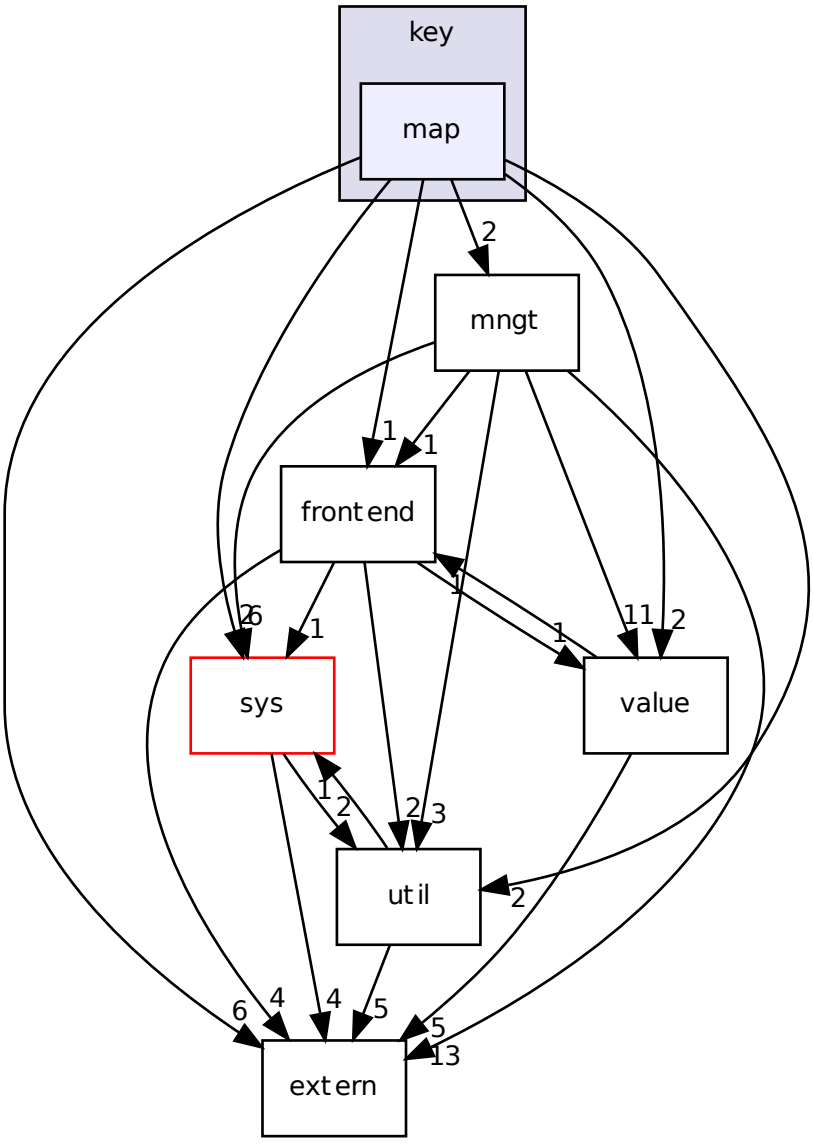
  Declaration of `class` *WindowsConsole.*

- file XtermConsole.h

  Declaration of `class` *XtermConsole.*

8.12 keyvalue/key/map/ Directory Reference

Directory dependency graph for keyvalue/key/map/:



Files

- file Default.h

  Declaration and implementation of `template class` *Default.*

- file FlagMap.h

  Declaration and implementation of `template class` *FlagMap.*

- file NoMap.h

  Declaration of `template class` *NoMap.*

- file ObjectMap.h

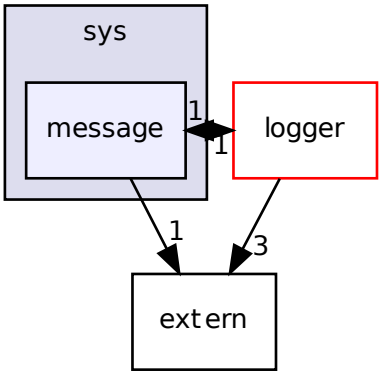
  Declaration and implementation of `template class` *ObjectMap.*

- file PartialMap.h

  Declaration of `template class` *PartialMap.*

8.13 keyvalue/sys/message/ Directory Reference

Directory dependency graph for keyvalue/sys/message/:



Files

- file Message.h

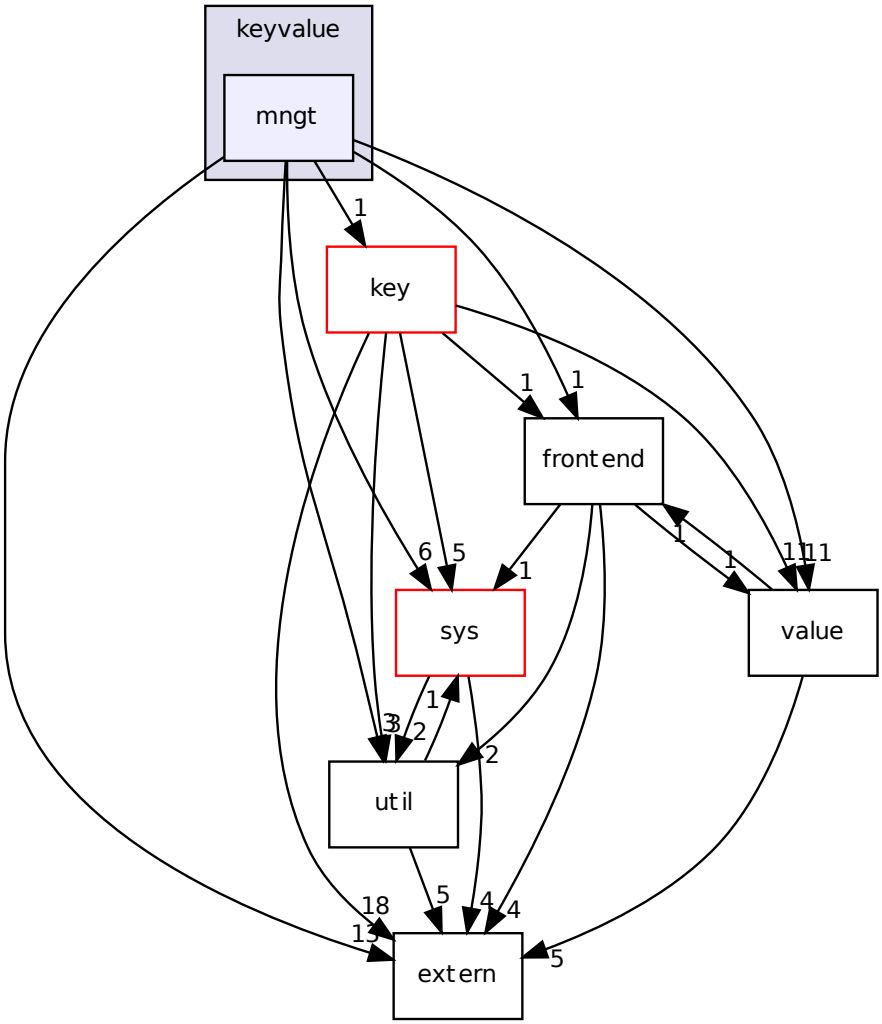
  Declaration of `class` *Message.*

- file MessageImpl.h

  Declaration of `classes` *MessageImpl, Error, Info, Warning, Report and Debug.*

8.14 keyvalue/mngt/ Directory Reference

Directory dependency graph for keyvalue/mngt/:



Files

- file Builder.h

  Declaration of primary `template class` *Builder.*

- file **BuilderFrom.h**
- file Calculator.h

  Declaration of primary `template class` *Calculator.*

- file Command.h

  Declaration of `class` *Command.*

- file DataSet.h

  Declaration of `class` *DataSet.*

- file DeclareBuilder.h

  A helper file to declare Builders.

- file DeclareCalculator.h

  A helper file to declare Calculators.

- file process.h

  Declaration of function keyvalue::process().

- file Processor.h

  Declaration of `class` *Processor.*

- file **ProcessorInstantiator.h**
- file ProcessorMngr.h

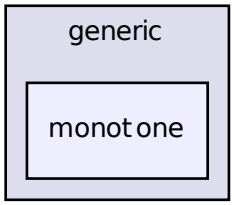
  Declaration of `class` *ProcessorMngr.*

- file Repository.h

  Declaration of `class` *Repository.*

8.15 keyvalue/key/generic/monotone/ Directory Reference

Directory dependency graph for keyvalue/key/generic/monotone/:



Files

- file Decreasing.h

  Declaration of `class` *Decreasing.*

- file Increasing.h

  Declaration of `class` *Increasing.*

- file NonMonotone.h

  Declaration of `class` *NonMonotone.*

- file StrictlyDecreasing.h

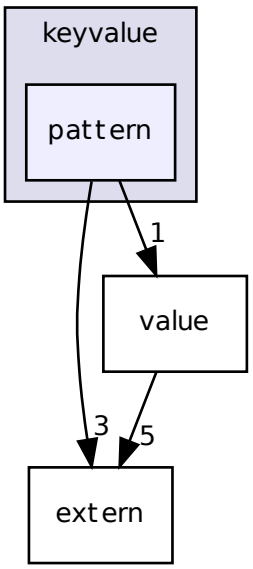
  Declaration of `class` *StrictlyDecreasing.*

- file StrictlyIncreasing.h

  Declaration of `class` *StrictlyIncreasing.*

8.16 keyvalue/pattern/ Directory Reference

Directory dependency graph for keyvalue/pattern/:



Files

- file KeyInSingle.h

  Declaration of `class` *KeyInSingle.*

- file KeysInMatrix.h

  Declaration of `class` *KeysInMatrix.*

- file KeysInVector.h

  Declaration of `class` *KeysInVector.*

- file Pattern.h

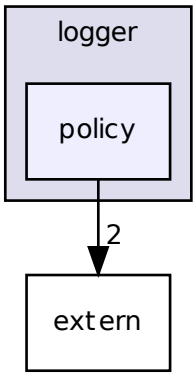
  Declaration of `class` *Pattern and functions keyvalue::pattern::isKey() and keyvalue::pattern::getKey().*

- file Table.h

  Declaration of `class` *Table.*

8.17 keyvalue/sys/logger/policy/ Directory Reference

Directory dependency graph for keyvalue/sys/logger/policy/:



Files

- file AddPrefix.h

  Declaration of `class` *AddPrefix.*

- file ForwardToGlobalLogger.h

  Declaration of `class` *ForwardToGlobalLogger.*

- file IgnoreColor.h

  Declaration of `class` *IgnoreColor.*

- file IgnoreFailure.h

  Declaration of `class` *IgnoreFailure.*

- file IgnorePrefix.h

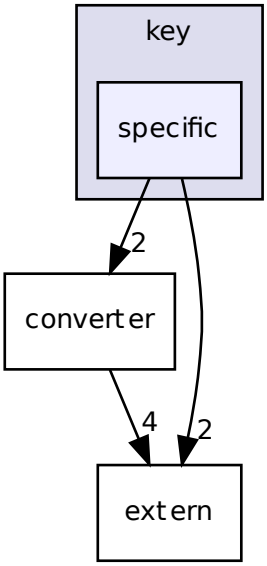
  Declaration of `class` *IgnorePrefix.*

- file XtermColor.h

  Declaration of `class` *XtermColor.*

8.18 keyvalue/key/specific/ Directory Reference

Directory dependency graph for keyvalue/key/specific/:



Files

- file Export.h

  Declaration of `class` *Export.*

- file Imports.h

  Declaration of `class` *Imports.*

- file ProcessNow.h

  Declaration of `class` *ProcessNow.*

- file Processor.h

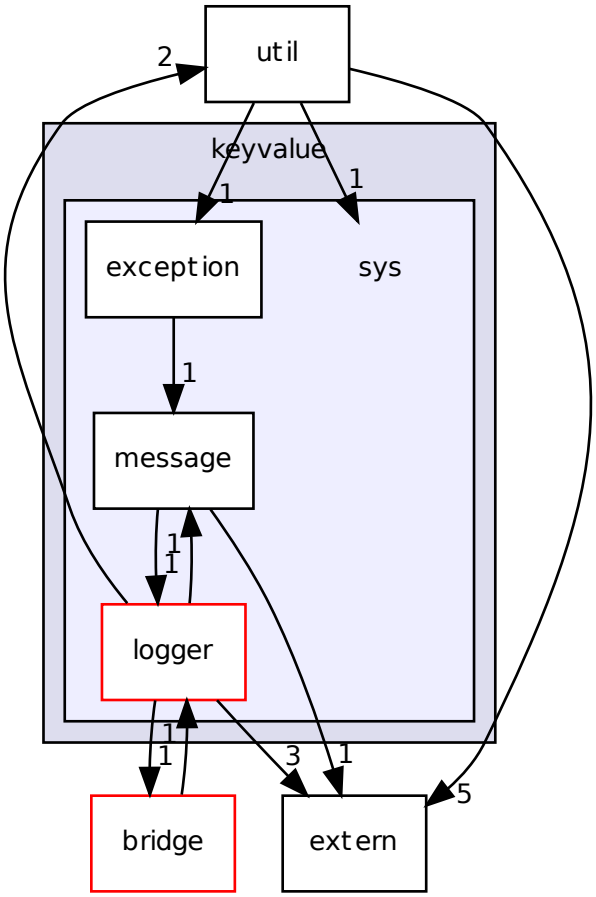
  Declaration of `class` *Processor (key).*

- file VectorOutput.h

  Declaration of `class` *VectorOutput.*

8.19 keyvalue/sys/ Directory Reference

Directory dependency graph for keyvalue/sys/:

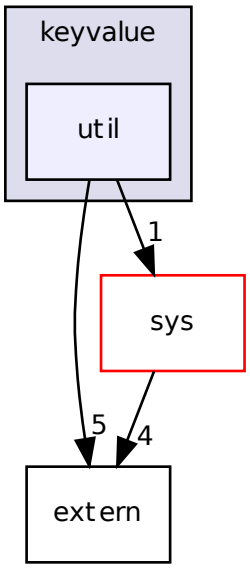


Directories

- directory exception
- directory logger
- directory message

8.20 keyvalue/util/ Directory Reference

Directory dependency graph for keyvalue/util/:



Files

- file Global.h

  Declaration of `template class` *Global.*

- file IsBasic.h

  Declaration and implementation of `template class` *IsBasic.*

- file Lexical.h

  Declaration of `template` *function Lexical.*

- file NullDeleter.h

  Declaration of `class` *NullDeleter.*

- file Saver.h

  Definition and implementation of `template class` *Saver.*

- file tenor2ptime.h

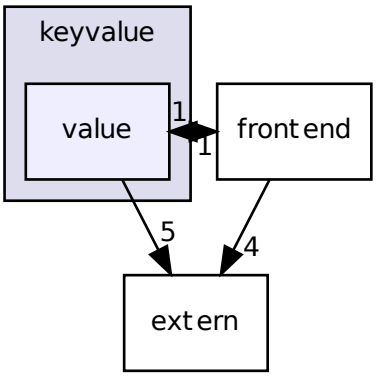
  Declaration of function tenor2ptime().

- file Util.h

  Documentation of `namespace` *util.*

8.21 keyvalue/value/ Directory Reference

Directory dependency graph for keyvalue/value/:



Files

- file Matrix.h

  Declaration of `class` *Matrix (container).*

- file Nothing.h

  Declaration of `class` *Nothing.*

- file Parent.h

  Declaration of `template class` *Parent.*

- file Result.h

  Declaration of `classes` *Result and ObjectPtr.*

- file Single.h

  Declaration of `class` *Single (container).*

- file TypeName.h

  Definition of `template class` *TypeName.*

- file Value.h

Declaration and (partial) implementation of `class` *Value.*

- file Variant.h

  Declaration and (partial) implementation of `class` *Variant.*

- file Vector.h

  Declaration of `class` *Vector (container).*

Chapter 9

Namespace Documentation

9.1 keyvalue Namespace Reference

All functionalities of *KeyValue* library are inside this `namespace`.

Namespaces

- namespace key

  All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

- namespace frontend

  All `classes` and functions needed by frontends belong to this namespace.

- namespace util

  Utility `classes` and functions are member of this `namespace`.

- namespace value

  All KeyValue containers belong to this namespace.

- namespace tag

  All Processor tags are member of this `namespace`.

- namespace pattern

  Key-value pattern recognition `class` are members of this namespace.

- namespace exception

  All exception functionalities are inside this `namespace`.

- namespace logger

  All logger functionalities are inside this namespace.

Classes

- class Bridge

  Bridge's specific data.

- class Builder

  Concrete Builder.

- class BuilderFromVariant

  Base `class` for Builders that can build from value::Variant.

- class BuilderFrom

  Base `class` for Builders that can build from InputType.

- class Calculator

  Concrete Calculator.

- class Command

  Command interface.

- class DataSet

  A named map from keys to values.

- class Processor

  Protocol `class` which defines Processor interface.

- class ProcessorInstantiator

  Instantiate a processor given its Tag.

- class ProcessorMngr

  The processor manager.

- class Repository

  The Repository of DataSets.

- class Message

  Abstract `class` which defines the interface for all types of message.

- class MessageImpl

  Implementation of `class` Message.

Typedefs

- typedef MessageImpl< 0 > **Error**
- typedef MessageImpl< 1 > **Logic**
- typedef MessageImpl< 2 > **Info**
- typedef MessageImpl< 3 > **Warning**
- typedef MessageImpl< 4 > **Report**
- typedef MessageImpl< 5 > **Debug**

Functions

- value::Value process (frontend::Queue &data)

  Processes a frontend::Queue.

- value::Value process (shared_ptr< DataSet > data)

  Processes a DataSet.

- std::ostream & operator<< (std::ostream &os, const Message &message)

  ostream operator<<() for Message.

9.1.1 Detailed Description

All functionalities of *KeyValue* library are inside this `namespace`. This `class` implements a standard Logger.

9.1.2 Function Documentation

9.1.2.1 value::Value keyvalue::process (frontend::Queue & *data*)

Processes a frontend::Queue.

This function creates a DataSet from a frontend::Queue. The DataSet gets the frontend::Queue's name after leading and trailing white spaces are removed. Key-value patterns are parsed at this stage.

Then the DataSet is sent to process(shared_ptr<DataSet>).

Parameters

data : the original frontend::Queue.

Returns

The DataSet's name or the result of DataSet processing.

9.1.2.2 value::Value keyvalue::process (shared_ptr< DataSet > *data*)

Processes a DataSet.

This function adds a DataSet to the Repository and, if requested, process it.

By default, the DataSet is not processed, unless its name is empty or key 'ProcessNow' is 'TRUE'.

Parameters

data : The DataSet to be added to the Repository;

Returns

When the DataSet is processed, the result of this processing is returned. Otherwise DataSet's name is returned.

9.1.2.3 std::ostream& keyvalue::operator<< (std::ostream & *os*, const Message & *message*)

ostream operator<<() for Message.

9.2 keyvalue::exception Namespace Reference

All exception functionalities are inside this `namespace`.

Classes

- class Exception

  Base protocol `class` for all exceptions.

- class ExceptionImpl

  Concrete `template` `class` which implements Exception's pure virtual methods.

Typedefs

- typedef ExceptionImpl< std::runtime_error, Error > **RuntimeError**
- typedef ExceptionImpl< std::logic_error, Logic > **LogicError**

Functions

- std::ostream & operator<< (std::ostream &os, const Exception &exception)

  ostream operator<<() for Exception.

- template<typename LhsType , typename RhsType >
  LhsType & operator& (const LhsType &lhs, const RhsType &rhs)

  Non member operator& for exceptions.

9.2.1 Detailed Description

All exception functionalities are inside this `namespace`. This namespace was created mainly to surround operator& which, otherwise, could conflict with the standard bitwise *and* operator, `&`.

The main `classes` inside this `namespace` are exception::Exception, exception::RuntimeError and exception::LogicError. They are all exported to the outer `namespace`, namely, keyvalue.

9.2.2 Function Documentation

9.2.2.1 std::ostream& keyvalue::exception::operator<< (std::ostream & *os*, const Exception & *exception*)

ostream operator<<() for Exception.

9.2.2.2 LhsType& keyvalue::exception::operator& (const LhsType & *lhs*, const RhsType & *rhs*)

Non member operator& for exceptions.

This function appends data to an ExceptionImpl by forwarding the call to ExceptionImpl::operator&. Additionally, this function provides two services:

- Cast the `const`-ness away: Consider this typical use of operator&:

```
throw RuntimeError() & "Invalid price."
```

Notice that the RuntimeError object is a temporary and, thus, automatically qualified as `const`. This `const`-ness must be casted away inside this function since the 'temporary' must change.

- Prevent up casting: Exception `class`es derived from any instantiation of ExceptionImpl inherit ExceptionImpl::operator& which returns a reference to the base `class`. This causes a problem when throwing derived objects which make use of ExceptionImpl::operator&. For instance, recall that Repository::NotFound derives from RuntimeError which is an instantiation of ExceptionImpl and consider the following piece of code:

```
try {
  throw Repository::NotFound() & "Foo";
}
catch(Repository::NotFound&) {
  // do something
}
```

If the operator& above is the ExceptionImpl's member, then the result of `Repository::NotFound() & "Foo"` is an ExceptionImpl and not a Repository::NotFound. Hence, the `catch` block will not be executed.

The type of the left operand of operator& given here is a `template` parameter. Hence, this function provides a better type match to Repository::NotFound than ExceptionImpl::operator& because the latter needs to up cast the left operand. Moreover, this function can return the same type of its left operand. We conclude that this function takes precedence over ExceptionImpl::operator& and its result has same type as its left operand. Therefore, the issue described above vanishes.

Last but not least. This `class` is inside the `namespace` exception. Thanks to Koenig look-up, it prevents name clashes with the bitwise 'AND' operator &.

Parameters

LhsType : (`template` parameter) Type of left operand;

RhsType : (`template` parameter) Type of right operand;

lhs : Left operand;

rhs : Left operand.

Returns

A non `const` reference to the result of lhs.operator&(rhs).

9.3 keyvalue::frontend Namespace Reference

All `class`es and functions needed by frontends belong to this namespace.

Classes

- class FrontEnd

  Front-end's specific data.

- class LexicalToolKit

  Manager of all lexical converters needed for front-end input/output.

- class Queue

  Protocol `class` *which defines the Queue interface.*

9.3.1 Detailed Description

All `class`es and functions needed by frontends belong to this namespace.

9.4 keyvalue::key Namespace Reference

All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

Classes

- class Device

  Device key.

- class Global

  Global key.

- class Level

  Level key.

- class Checker

  Primary key checker `template class`.

- class Checker< ConverterType, value::Single >

  Specialization of Checker for InputType equal to value::Single.

- class Checker< ConverterType, value::Vector >

  Specialization of Checker for InputType equal to value::Vector.

- class Checker< ConverterType, value::Matrix >

  Specialization of Checker for InputType equal to value::Matrix.

- class StdMatrix

  Converter from value::Matrix to `shared_ptr<std::vector<std::vector<ElementType>>>`.

- class StdSingle

  Converter from value::Single to ElementType.

- class StdVector

  Converter from value::Vector to `shared_ptr<std::vector<ElementType>>`.

- class Geq

  Greater-than-or-equal-to bound `class`.

- class Greater

  Greater-than bound `class`.

- class Leq

  Less-than-or-equal-to bound `class`.

- class Less

  Less-than bound `class`.

- class NoBound

  NoBound bound `class`.

- class Bounded

  Key for bounded values.

- class Matrix

  Generic key whose converter type is StdMatrix.

- class Decreasing

  Decreasing monotone `class`.

- class Increasing

  Increasing monotone `class`.

- class NonMonotone

  NonMonotone `class`.

- class StrictlyDecreasing

  StrictlyDecreasing monotone `class`.

- class StrictlyIncreasing

  StrictlyIncreasing monotone `class`.

- class MonotoneBoundedVector

  Key for monotone bounded vectors.

- class Positive

  A general key for positive values.

- class Single

  Generic key whose converter type is StdSingle.

- class StrictlyPositive

A general key whose value is a strictly positive number.

- class Vector

  Generic key whose converter type is StdVector.

- class Key

  Base for all real keys.

- struct DefaultMap

  Meta-function which defines Traits' default map type.

- struct Default

  Meta-function which defines Traits' default map type.

- class FlagMap

  Maps names to constants.

- class NoMap

  Identity map (aka, NoMap).

- class ObjectMap

  Maps names to objects.

- class PartialMap

  Partially maps names to constants.

- class Export

  Export key.

- class Imports

  Imports key.

- class ProcessNow

  ProcessNow key.

- class Processor

  Processor key.

- struct VectorOutputBase

  VectorOutput's base `class`.

- class VectorOutput

  VectorOutput key.

- class Traits

  Real keys must derive from adequate instantiations of this `template class`.

9.4.1 Detailed Description

All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

9.5 keyvalue::logger Namespace Reference

All logger functionalities are inside this namespace.

Classes

- class ConsoleLogger

  Implements a console Logger.

- class FileLogger

  Implements a file Logger.

- class Logger

  Protocol `class` which defines the interface for all types of logger.

- class LoggerImpl

  Abstract `class` which implements main methods of concrete Loggers.

- class AddPrefix

  Defines a policy for Message prefixes, which is, to send the prefix to the underlying Logger's device.

- class ForwardToGlobalLogger

  Defines a policy to apply when the Logger fails to process a Message, namely, forward it to GlobalLogger.

- class IgnoreColor

  Defines a policy for Message colors, which is, to ignore them.

- class IgnoreFailure

  Defines a policy to apply when the logger fails, which is, to ignore the failure.

- class IgnorePrefix

  Defines a policy for Message prefixes, which is, to ignore them.

- class XtermColor

  Defines a policy for Message color, which is, to set the underlying device of the Logger to print in Message's color.

- class WindowsConsole

  This `class` implements a Windows console Logger.

- class XtermConsole

  Implements an xterm console Logger.

Enumerations

- enum Color {

  black = 0, **red**, **green**, **yellow**,

  blue, **pink**, **cyan**, **white**,

grey, **bold_red**, **bold_green**, **bold_yellow**,
bold_blue, **bold_pink**, **bold_cyan**, **bold_white** }

Console colors.

Functions

- void resetGlobal ()

  Resets util::Global<Logger> to its default (StdLogger).

9.5.1 Detailed Description

All logger functionalities are inside this namespace.

9.5.2 Enumeration Type Documentation

9.5.2.1 enum Color

Console colors.

9.5.3 Function Documentation

9.5.3.1 void keyvalue::logger::resetGlobal ( )

Resets util::Global<Logger> to its default (StdLogger).

WARNING: If this function is called by ∗utilGlobal<Logger>, the caller may cease to exist immediately (it suicides). In this case the caller can no longer access any of its non static members.

9.6 keyvalue::pattern Namespace Reference

Key-value pattern recognition `class` are members of this namespace.

Classes

- class KeyInSingle

  Pattern where a value::Single object, recognized as a key, is followed by any value::Value.

- class KeysInMatrix

  Pattern made by a value::Matrix where all elements either in the first row or in the first column are keys.

- class KeysInVector

  Pattern made by a value::Vector whose entries are keys followed by a value::Vector or value::Matrix.

- class Pattern

  Protocol `class` which defines the interface for pattern recognition `classes`.

- class Table

  Pattern where a value::Matrix contains the row, column and table keys.

Functions

- bool isKey (const value::Variant &variant)

  Checks if variant may be considered as a key.

- string getKey (const value::Variant &variant)

  Gets the key held by variant.

9.6.1 Detailed Description

Key-value pattern recognition `class` are members of this namespace.

9.6.2 Function Documentation

9.6.2.1 bool keyvalue::pattern::isKey (const value::Variant & *variant*)

Checks if *variant* may be considered as a key.

The check is successful if the following conditions hold:

- it contains a value::Single holding a `string` *S*;
- excluding trailing spaces, *S* ends in " =" (space + equal sign);
- excluding the ending '=', *S* has a non space character.

Parameters

variant : value::Variant to be checked.

Returns

If *variant* may be considered as a key, then this method returns true. Otherwise it returns `false`.

9.6.2.2 string keyvalue::pattern::getKey (const value::Variant & *variant*)

Gets the key held by *variant*.

More precisely, the key is what remains after the following operations are performed in the `string` held by *variant* :

- the last '=' (equal sign) is replaced by ' ' (space);
- trailing spaces are removed;
- leading spaces are removed.

The debug version of this method checks if *variant* may be considered as a key (by calling isKey()) and throws an exception when the check fails. The release version of this method has undefined behavior when *variant* may not be considered as a key.

Parameters

variant : value::Variant to be checked.

Returns

The key held by *variant*.

Exceptions

LogicError : (Debug built only) If *variant* may not be considered as a key.

9.7 keyvalue::tag Namespace Reference

All Processor tags are member of this `namespace`.

9.7.1 Detailed Description

All Processor tags are member of this `namespace`.

9.8 keyvalue::util Namespace Reference

Utility `classes` and functions are member of this `namespace`.

Classes

- class Global

  Manager of global objects.

- struct IsBasic

  This meta-function checks if a type is either `bool`, `double`, `string`, `ptime` or `unsigned int` or not.

- struct IsBasicOrEnum

  This meta-function checks if a type is either `bool`, `double`, `string`, `ptime`, `unsigned int` or enum or not.

- class NullDeleter

  A `shared_ptr` deleter that does not do anything.

- class Saver

  Saves a variable reference at construction time to restore/set its value at destruction time.

Functions

- template<typename From , typename To >
  To Lexical (const From &input)

  Converts from type From to type To.

- ptime tenor2ptime (const ptime &start, const string &tenor)

  Convert a tenor string (from a start date) into the corresponding date.

9.8.1 Detailed Description

Utility `classes` and functions are member of this `namespace`.

9.8.2 Function Documentation

9.8.2.1 To keyvalue::util::Lexical (const From & *input*)

Converts from type *From* to type *To*.

Parameters

From : (`template` parameter) The input type;

To : (`template` parameter) The output type;

input : Data to be converted.

Return values

For <*From*, *To*> = <`double`, `bool`> : It returns `true` if *input* != 0.0. Otherwise, it returns `false`.

For <*From*, *To*> = <`bool`, `double`> : It returns 1.0 if *input* = `true` and 0.0 if *input* = `false`.

For <*From*, *To*> = <`string`, `bool`> : It returns `true` if *input* is either "TRUE", "True", "true", "YES", "Yes", "yes", "Y" or "y". It returns `false` if *input* is either "FALSE", "False", "false", "NO", "No", "no", "N" or "n".

For <*From*, *To*> = <`bool`, `string`> : It returns "True" if *input* = `true` and "False" if *input* = `false`.

For <*From*, *To*> = <`string`, `double`> : It returns the `double` represented by `string` *input*.

For <*From*, *To*> = <`double`, `string`> : It returns the `string` representation of *input*.

throw RuntimeError : If required conversion is invalid.

9.8.2.2 ptime keyvalue::util::tenor2ptime (const ptime & *start*, const string & *tenor*)

Convert a tenor string (from a start date) into the corresponding date.

A *tenor* is a string of the form *nP*, where *n* is a positive integer and *P* is a `char` defining a period of time. Possible values for *P* are:

- 'd' or 'D' for day;
- 'w' or 'W' for week;

- 'm' or 'M' for month; or
- 'y' or 'Y' for year.

This function returns the date which follows *start* by the amount of time defined by the *tenor*.

Parameters

start : The start date;
tenor : The tenor.

Returns

A ptime value corresponding to the *tenor* amount of time from *start* date.

Exceptions

RuntimeError : When an invalid tenor is provided.

9.9 keyvalue::value Namespace Reference

All *KeyValue* containers belong to this namespace.

Classes

- class Matrix
  Matrix of `Variant`s.
- class Nothing
  Empty `class` to represent empty data.
- struct Parent
  Primary Parent `template` meta-function.
- class Result
  A single-valued container for ObjectPtr or Value.
- class Single
  A 1 x 1 Matrix.
- struct TypeName
  Meta function which returns the name of type.
- class Value
  A single-valued container for Single, Vector or Matrix.
- class Variant
  Single-value multi-type container.
- class Vector
  A m x 1 or 1 x n Matrix.

Typedefs

- typedef shared_ptr< void > **ObjectPtr**

Functions

- void intrusive_ptr_add_ref (Matrix::Impl *pimpl)
- void intrusive_ptr_release (Matrix::Impl *pimpl)
- std::ostream & operator<< (std::ostream &os, const Matrix &matrix)

  ostream operator<<() for Matrix.

- std::ostream & operator<< (std::ostream &os, const Nothing &nothing)

  ostream operator<<() for Nothing.

- std::ostream & operator<< (std::ostream &os, const Result &rhs)

  ostream operator<<() for Result.

- std::ostream & operator<< (std::ostream &os, const Single &rhs)

  ostream operator<<() for Single.

- std::ostream & operator<< (std::ostream &os, const Value &rhs)

  ostream operator<<() for Value.

- std::ostream & operator<< (std::ostream &os, const Variant &rhs)

  ostream operator<<() for Variant.

- std::ostream & operator<< (std::ostream &os, const Vector &vector)

  ostream operator<<() for Vector.

9.9.1 Detailed Description

All *KeyValue* containers belong to this namespace.

9.9.2 Function Documentation

9.9.2.1 void keyvalue::value::intrusive_ptr_add_ref (Matrix::Impl * *pimpl*)

headerfile Matrix.h "keyvalue/value/Matrix.h"

brief Increment Matrix reference counting.

See documentation of intrusive_ptr in the Smart Ptr (Boost) library.

param Pointer to Matrix::Impl storing the counter.

9.9.2.2 void keyvalue::value::intrusive_ptr_release (Matrix::Impl * *pimpl*)

headerfile Matrix.h "keyvalue/value/Matrix.h"

brief Decrement Matrix reference counting.

See documentation of intrusive_ptr in the Smart Ptr (Boost) library.

param Pointer to Matrix::Impl storing the counter.

9.9.2.3 std::ostream& keyvalue::value::operator<< (std::ostream & *os,* const Matrix & *matrix*)

ostream operator<<() for Matrix.

9.9.2.4 std::ostream& keyvalue::value::operator<< (std::ostream & *os,* const Nothing & *nothing*)

ostream operator<<() for Nothing.

9.9.2.5 std::ostream& keyvalue::value::operator<< (std::ostream & *os,* const Result & *rhs*)

ostream operator<<() for Result.

9.9.2.6 std::ostream& keyvalue::value::operator<< (std::ostream & *os,* const Single & *rhs*)

ostream operator<<() for Single.

9.9.2.7 std::ostream& keyvalue::value::operator<< (std::ostream & *os,* const Value & *rhs*)

ostream operator<<() for Value.

9.9.2.8 std::ostream& keyvalue::value::operator<< (std::ostream & *os,* const Variant & *rhs*)

ostream operator<<() for Variant.

9.9.2.9 std::ostream& keyvalue::value::operator<< (std::ostream & *os,* const Vector & *vector*)

ostream operator<<() for Vector.

Chapter 10

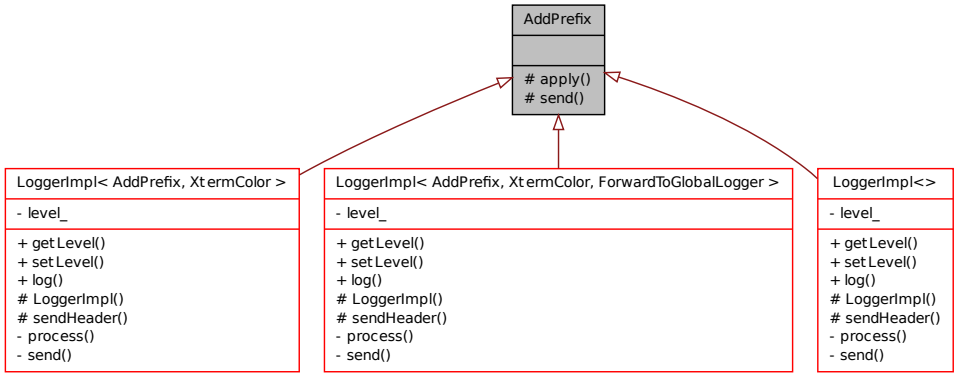
Class Documentation

10.1 AddPrefix Class Reference

Defines a policy for Message prefixes, which is, to send the prefix to the underlying Logger's device.

`#include <keyvalue/sys/logger/policy/AddPrefix.h>`

Inheritance diagram for AddPrefix:



Protected Member Functions

- bool **apply** (const Message &message)
- virtual bool **send** (const string &message)=0

10.1.1 Detailed Description

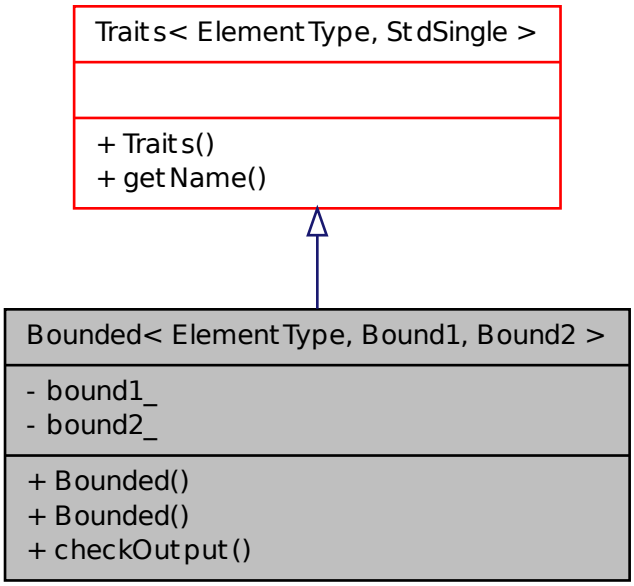
Defines a policy for Message prefixes, which is, to send the prefix to the underlying Logger's device.

10.2 Bounded< ElementType, Bound1, Bound2 > Class Template Reference

Key for bounded values.

`#include <keyvalue/key/generic/Bounded.h>`

Inheritance diagram for Bounded< ElementType, Bound1, Bound2 >:



Public Types

- enum
- typedef Traits< ElementType, StdSingle, Default > **Traits_**
- typedef StdSingle< typename Default< ElementType >::OutputType_ > **ConverterType_**
- typedef ConverterType_::InputType_ **InputType_**
- typedef ConverterType_::OutputType_ **OutputType_**

Public Member Functions

- Bounded (const string &name, const ElementType &bound1)

  Constructs a key accepting values which are bounded either from below or from above.

- Bounded (const string &name, const ElementType &bound1, const ElementType &bound2)

  Constructs a key accepting values which are bounded both from below and from above.

- void checkOutput (ElementType element) const

  Compares element against the bound(s).

- string getName () const

  Gets Key's name.

- string getName () const

  Gets name.

- void setName (const string &name)

  Sets name.

Private Attributes

- const Bound1< ElementType > **bound1_**
- const Bound2< ElementType > **bound2_**

10.2.1 Detailed Description

template<typename ElementType, template< typename > class Bound1, template< typename > class Bound2 = NoBound> class keyvalue::key::Bounded< ElementType, Bound1, Bound2 >

Key for bounded values. This is a `template class` for keys corresponding to bounded values.

This `template` is used, for instance, to define keys accepting `double` values in intervals. More generally, a value *x* of type *ElementType* will be accepted if, and only if,

- *bound1_*::check(*x*) = `true`; and
- *bound2_*::check(*x*) = `true`;

where *bound1_* and *bound2_* are instantiations (for *ElementType*) of bound `template classes` *Bound1* and *Bound2*, resp.

Bound `template classes` implement the method *check* whose signature is

```
bool
check(const  ElementType& x) const;
```

This method compares its argument against a bound fixed at construction time and returns `true` or `false` depending on whether the argument is in the bound or not.

The following bound `template classes` are implemented:

- NoBound (used only as a default value for *Bound2*);
- Less;
- Leq;
- Greater;

- Geq.

Bound1 cannot be NoBound and, by default, *Bound2* is NoBound.

The two bounds are passed to constructors. If *Bound2* is NoBound, then only *bound1* must be provided (otherwise the compiler will complain).

Example 1: Consider a key for weights. It accepts only `double` values (strictly) greater than 0.0, the `template` instantiation and the key definition would be

```
Bounded<double, Greater> weight("Weight", 0.0);
```

Example 2: A key for probabilities may accept only double values in the interval [0,1]:

```
Bounded<double, Geq, Leq> probability("Probability", 0.0, 1.0);
```

Parameters

ElementType : (`template` parameter) See description above;

Comp1 : (`template` parameter) 1st comparison `class`;

Comp2 : (`template` parameter) 2nd comparison `class`.

10.2.2 Constructor & Destructor Documentation

10.2.2.1 Bounded (const string & *name,* const ElementType & *bound1*)

Constructs a key accepting values which are bounded either from below or from above.

Parameters

name : The key name;

bound1 : Bound which values are compared against (through *Bound1*).

10.2.2.2 Bounded (const string & *name,* const ElementType & *bound1,* const ElementType & *bound2*)

Constructs a key accepting values which are bounded both from below and from above.

Parameters

name : The key name;

bound1 : First bound which values are compared against (through by *Bound1*);

bound2 : Second bound which values are compared against (through by *Bound2*).

10.2.3 Member Function Documentation

10.2.3.1 void checkOutput (ElementType *element*) const

Compares *element* against the bound(s).

Parameters

element : Element to be checked.

Exceptions

RuntimeError : If *element* is out of bound(s).

10.2.3.2 string getName ( ) const [inherited]

Gets Key's name.

Returns

The Key's name.

10.2.3.3 string getName ( ) const [inherited]

Gets name.

Returns

The name.

10.2.3.4 void setName (const string & *name*) [inherited]

Sets name.

Parameters

name : The name.

10.3 Bridge Class Reference

Bridge's specific data.

`#include <keyvalue/bridge/Bridge.h>`

Public Member Functions

- const char * getCoreLibraryName () const
  Gets core library's name.
- const char * getSimpleInfo () const
  Gets an info message about the core library.
- const char * getCompleteInfo () const
  Gets an info message about the core library.

10.3.1 Detailed Description

Bridge's specific data. This is used to get information on the core library. Only the declaration is provided. Bridge developers must write the implementation.

Use util::Global to access the global Bridge.

10.3.2 Member Function Documentation

10.3.2.1 const char* getCoreLibraryName ( ) const

Gets core library's name.

The name must be a single word, otherwise, some front-ends will get in trouble.

Returns

The core library's name.

10.3.2.2 const char* getSimpleInfo ( ) const

Gets an info message about the core library.

The message must be one line long. For larger messages use method getCompleteInfo().

Returns

The info message.

10.3.2.3 const char* getCompleteInfo ( ) const

Gets an info message about the core library.

Returns

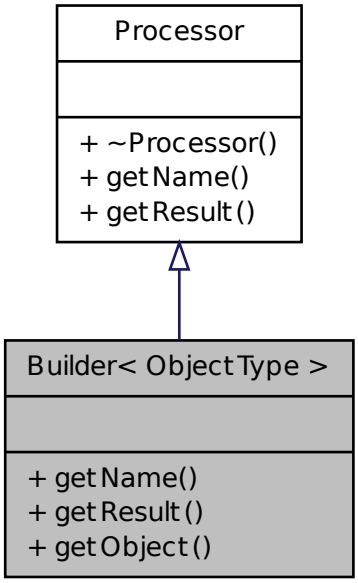
The info message.

10.4 Builder< ObjectType > Class Template Reference

Concrete Builder.

```
#include <keyvalue/mngt/DeclareBuilder.h>
```

Inheritance diagram for Builder< ObjectType >:



Public Member Functions

- const char * getName () const

  Gets Builder's name.

- value::Result getResult (const DataSet &data) const

  Gets the object built from a DataSet.

- shared_ptr< ObjectType > getObject (const DataSet &data) const

  Gets the object built from a DataSet.

10.4.1 Detailed Description

template<typename ObjectType> class keyvalue::Builder< ObjectType >

Concrete Builder. Each Builder is a specialization of this `template class` which is parameterized on *ObjectType*, the type of object built.

Builder<*OutputType*> is associated to a type *Tag* which also uniquely identifies it. Actually, this extra type seems to be redundant since *ObjectType* also uniquely identifies the Builder. However some C++ technicalities and the way we want to do the processor registration into the ProcessorMngr enforce the use of *Tag*.

Builder<*OutputType*> derives from Processor and, additionally, it might derive from other `class`es depending on whether it has certain features or not. For instance, some Builders are also commands and, in that case, they must `public`ly derive from Command. On the same way, some Builders can build from value::Variant input (see documentation of BuilderFromVariant::getObject() for an example) in which case they must publicly derive from BuilderFromVariant<*OutputType*>.

Rather than declare Builder<*OutputType*> from scratch, providing all its base `class`es and declaring all its methods, the helper file keyvalue/mngt/DeclareBuilder.h should be used. This file expects some preprocessor macros to be `#define`d or not and based on that, it makes the correct declaration. Therefore, Builder writers must only `#define` those macros and then `#include` this file to get the right declaration. After that they must provide the implementations.

The macros are:

- *OBJECT_TYPE* : the fully qualified type of object built by the builder (e.g., logger::Logger). DeclareBuilder.h provides the declaration of Builder<*OBJECT_TYPE*>.
- *TAG* : The tag which uniquely identifies the builder. In addition to the Builder<*OBJECT_TYPE*>, DeclareBuilder.h also declares `class` *TAG* (for which no implementation is required) inside `namespace` tag. Builder<*OBJECT_TYPE*> `public`ly derives from ProcessorInstantiator<tag::*TAG*>.
- *COMMAND* : This macro must be `#define`d if, and only if, Builder<*OBJECT_TYPE*> is a command, in which case, it `public`ly derives from Command and must implement its pure `virtual` methods.
- *BUILDS_FROM_BOOL* : This macro must be `#define`d if, and only if, Builder<*OBJECT_TYPE*> is able to build the object from a `bool` input. In that case, Builder<*OBJECT_TYPE*> `public`ly derives from BuilderFrom<OBJECT_TYPE, `bool`> and must implement its pure `virtual` method.
- *BUILDS_FROM_DOUBLE* : Similar to *BUILDS_FROM_BOOL* for `double` input.
- *BUILDS_FROM_PTIME* : Similar to *BUILDS_FROM_BOOL* for `ptime` input.
- *BUILDS_FROM_STRING* : Similar to *BUILDS_FROM_BOOL* for `string` input.

The macros above must be `#define`d before DeclareBuilder.h is `#include`d and will be `#undef`ined at the end of that file.

Builder<*OBJECT_TYPE*> also derives from Processor and must implement its two `virtual` methods. However, DeclareBuilder.h provides the implementation for Builder<*OBJECT_TYPE*>::getResult() which just forwards the call to Builder<*OBJECT_TYPE*>::getObject(). Hence, only Builder<*OBJECT_TYPE*>::getName() needs to be implemented.

An example of how DeclareBuilder.h is used is given in keyvalue/bridge/processor/Logger.cpp:

```
namespace keyvalue {

#define  OBJECT_TYPE logger::Logger
#define  TAG         Logger
#include "keyvalue/mngt/DeclareBuilder.h"

*--------------------------------------------------------------------------
```

```
 * getName()
 *--------------------------------------------------------------------------*/

const char*
Builder<logger::Logger>::getName() const {
  return "Logger";
}

// ...

} // namespace keyvalue
```

10.4.2 Member Function Documentation

10.4.2.1 const char* getName ( ) const [virtual]

Gets Builder's name.

File DeclareBuilder.h does not provide the implementation of this method. Builder writers must do it.

Returns

The name.

Implements Processor.

10.4.2.2 value::Result getResult (const DataSet & *data*) const [virtual]

Gets the object built from a DataSet.

As said in Processor::getResult() documentation, this is a low level method which should be avoided in favor of DataSet::process().

File DeclareBuilder.h provides the implementation of this method which simply forwards the call to getObject().

Parameters

data : DataSet to be processed.

Returns

The object built.

Implements Processor.

10.4.2.3 shared_ptr<ObjectType> getObject (const DataSet & *data*) const

Gets the object built from a DataSet.

Similarly to getResult() this is a low level method which should be avoided in favor of DataSet::process().

File DeclareBuilder.h does not provide the implementation of this method. Builder writers must do it.

Parameters

data : DataSet to be processed.

Returns

The object built.

10.5 BuilderFrom< ObjectType, InputType > Class Template Reference

Base `class` for Builders that can build from *InputType*.

`#include <BuilderFrom.h>`

Public Member Functions

- virtual shared_ptr< ObjectType > getObject (const InputType &data) const =0

  Gets the object built by the derived Builder.

10.5.1 Detailed Description

template<typename ObjectType, typename InputType> class keyvalue::BuilderFrom< ObjectType, InputType >

Base `class` for Builders that can build from *InputType*. Here, *InputType* must be

- `bool`;
- `double`;
- `ptime`; or
- `string`.

Parameters

ObjectType : (`template` parameter) Type of object built by the derived Builder.

InputType : (`template` parameter) Type of input which the derived Builder can build from.

10.5.2 Member Function Documentation

10.5.2.1 virtual shared_ptr<ObjectType> getObject (const InputType & *data*) const [pure virtual]

Gets the object built by the derived Builder.

This is a pure `virtual` method which must be implemented by the derived Builder.

Parameters

data : The input data.

Returns

The object built by the derived Builder.

10.6 BuilderFromVariant< ObjectType > Class Template Reference

Base `class` for Builders that can build from value::Variant.

`#include <keyvalue/mngt/BuilderFrom.h>`

Public Member Functions

- shared_ptr< ObjectType > getObject (const value::Variant &data) const

  Gets the object built by the derived Builder.

Private Member Functions

- virtual const char * **getName** () const =0
- template<typename InputType >
  shared_ptr< ObjectType > build (const value::Variant &data) const

  Called by getObject.

10.6.1 Detailed Description

template<typename ObjectType> class keyvalue::BuilderFromVariant< ObjectType >

Base `class` for Builders that can build from value::Variant. Any Builder which can build from value::Variant must derive from this `class`. Additionally, it also must derive from at least one instantiation of `template class` BuilderFrom.

Parameters

ObjectType : (`template` parameter) Type of object build by the derived Builder.

10.6.2 Member Function Documentation

10.6.2.1 shared_ptr< ObjectType > getObject (const value::Variant & *data*) const

Gets the object built by the derived Builder.

In general, the data that builders need to perform their duties is so rich that must be stored in a DataSet. Nevertheless, in some particular cases, a single value::Variant might be enough. For instance, consider a builder that creates a curve given a few points on it. Normally, this processor needs the set of points together with interpolator and extrapolator methods. In this general case, a DataSet is necessary to hold all this information. However, when the curve is known to be constant, then a single number - the constant - is enough to build the whole curve. Rather than creating a DataSet to store a single `double` value, it would

be more convenient if the processor could accept just this value (or more generally, a value::Variant). That is the reason for this method.

Given the content type *InputType* of the provided value::Variant and assuming that the dynamic type of the derived object is BuilderFrom<*ObjectType*, *InputType*>, this method forwards its call to BuilderFrom<*ObjectType*, *InputType*>::getObject().

Parameters

data : The input value::Variant.

Returns

The object built by the derived Builder.

Exceptions

LogicError : If the dynamic type of the derived object is not BuilderFrom<*ObjectType*, *InputType*>.

10.6.2.2 shared_ptr< ObjectType > build (const value::Variant & *data*) const [private]

Called by getObject.

See the documentation of getObject(). Actually, that method only detects *InputType* and then call this one.

Parameters

data : The input value::Variant.

Returns

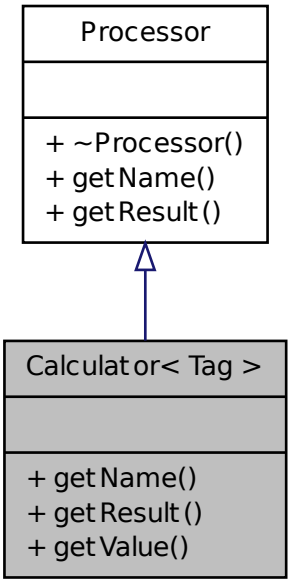
The object built by the derived Builder.

@ throw LogicError : If the dynamic type of `this` is not BuildFrom<*ObjectType*, `InputType`>.

10.7 Calculator< Tag > Class Template Reference

Concrete Calculator.

Inheritance diagram for Calculator< Tag >:



Public Member Functions

- const char * getName () const

  Gets Calculator's name.

- value::Result getResult (const DataSet &data) const

  Gets the value::Value calculated from a DataSet.

- value::Value getValue (const DataSet &data) const

  Gets the value::Value calculated from a DataSet.

10.7.1 Detailed Description

template<typename Tag> class keyvalue::Calculator< Tag >

Concrete Calculator. Each Calculator is a specialization of this `template class` which is parameterized on a type *Tag*. This type uniquely identifies the Calculator.

Calculator<*Tag*> `public`ly derives from Processor and, additionally, it might be derived from other `class`es depending on whether it has certain features or not. For instance, some Calculators are also commands and, in that case, they must `public`ly derive from Command.

Rather than declare Calculator<*Tag*> from scratch, providing all its base `classes` and declaring all its methods, the helper file keyvalue/mngt/DeclareCalculator.h should be used. This file expects some preprocessor macros to be `#defined` or not and based on that, it makes the correct declaration. Therefore, Calculator writers must only `#define` those macros and then `#include` this file to get the right declaration. After that they must provide the implementations.

The macros are:

- *TAG* : The tag which uniquely identifies the calculator. DeclareCalculator.h declares `class` *TAG* (for which no implementation is required) inside `namespace` tag. The specialization Calculator<tag::*TAG*> `publicly` derives from ProcessorInstantiator<tag::*TAG*>.

- *COMMAND* : This macro must b#`#defined` if and only if Calculator<tag::*TAG*> is a command, in which case, it `publicly` derives from Command and must implement its pure `virtual` methods.

The macros above must be `#defined` before DeclareCalculator.h is `#included` and will be `#undefined` at the end of that file.

Calculator<tag::*TAG*> also derives from Processor and must implement its two `virtual` methods. However, DeclareCalculator.h provides the implementation for Calculator<tag::*TAG*>::getResult() which just forwards the call to Calculator<tag::*TAG*>::getValue(). Hence, only Calculator<tag::*TAG*>::getName() needs to be implemented.

An example of how DeclareCalculator.h is used is given in keyvalue/bridge/processor/DeleteDataSets.cpp:

```
namespace keyvalue {

#define  TAG DeleteDataSets
#define  COMMAND
#include "keyvalue/mngt/DeclareCalculator.h"

/*-------------------------------------------------------------------------
 * getName()
 *-----------------------------------------------------------------------*/

const char*
Calculator<tag::DeleteDataSets>::getName() const {
  return "DeleteDataSets";
}

// ...

} // namespace keyvalue
```

10.7.2 Member Function Documentation

10.7.2.1 const char* getName ( ) const [virtual]

Gets Calculator's name.

File DeclareCalculator.h does not provide the implementation of this method. Calculator writers must do it.

Returns

The name.

Implements Processor.

10.7.2.2 value::Result getResult (const DataSet & *data*) const [virtual]

Gets the value::Value calculated from a DataSet.

As said in Processor::getResult() documentation, this is a low level method which should be avoided in favor of DataSet::process().

File DeclareCalculator.h provides the implementation of this method which simply forwards the call to getValue().

Parameters

data : DataSet to be processed.

Returns

The resulting value::Value.

Implements Processor.

10.7.2.3 value::Value getValue (const DataSet & *data*) const

Gets the value::Value calculated from a DataSet.

Similarly to getResult() this is a low level method which should be avoided in favor of DataSet::process().

File DeclareCalculator.h does not provide the implementation of this method. Calculator writers must do it.

Parameters

data : DataSet to be processed.

Returns

The resulting value::Value.

10.8 Checker< ConverterType, InputType > Class Template Reference

Primary key checker `template class`.

`#include <keyvalue/key/Checker.h>`

10.8.1 Detailed Description

template<typename ConverterType, typename InputType = typename ConverterType::InputType_-> class keyvalue::key::Checker< ConverterType, InputType >

Primary key checker `template class`. As said in Traits documentation, in some circumstances, a value associated to a key must pass some sanity check. This test is one step of the value processing as explained in Record::getProcessedValue() and is done through this `template class`.

It is easy to see that a specialization for each input type is needed. Indeed, a value::Vector, for instance, may be rejected or validated according to its dimension. On the other hand, checking dimensionality for a value::Single is meaningless. This `template class` takes the type parameter *ConverterType* from which it can recover the input type.

Checks are done by specializations methods. Real keys are derived from Traits which, in turn, derives from instantiations of this `template class`. Therefore, one can set up the checks to be performed by overhiding Check's methods. Bear in mind that, default implementations validate all values.

Parameters

ConverterType : (`template` parameter) Defines the container converter.

InputType : Converter's input type. Provided default value must not be changed.

10.9 Checker< ConverterType, value::Matrix > Class Template Reference

Specialization of Checker for *InputType* equal to value::Matrix.

`#include <keyvalue/key/Checker.h>`

Public Types

- typedef ConverterType::OutputType_ **OutputType_**

Public Member Functions

- void checkInput (const value::Matrix &input) const

  Check input.

- void checkSoFar (const ConverterType &container) const

  Check conversion process so far.

- void checkOutput (const OutputType_ &output) const

  Check output.

Private Member Functions

- virtual string getName () const =0

  Get Key's name.

- virtual void checkSize (size_t nRows, size_t nCols) const

  Check input size.

10.9.1 Detailed Description

template<typename ConverterType> class keyvalue::key::Checker< ConverterType, value::Matrix >

Specialization of Checker for *InputType* equal to value::Matrix.

10.9.2 Member Function Documentation

10.9.2.1 void checkInput (const value::Matrix & *input*) const

Check input.

The current implementation calls checkSize().

Parameters

input : Data to be checked.

10.9.2.2 void checkSoFar (const ConverterType & *container*) const

Check conversion process so far.

Parameters

container : The container.

10.9.2.3 void checkOutput (const OutputType_ & *output*) const

Check output.

Parameters

output : Data to be checked.

10.9.2.4 virtual string getName ( ) const [private, pure virtual]

Get Key's name.

Returns

The Key's name.

10.9.2.5 void checkSize (size_t *nRows*, size_t *nCols*) const [private, virtual]

Check input size.

Parameters

nRows : Number of input rows;
nCols : Number of input columns.

10.10 Checker< ConverterType, value::Single > Class Template Reference

Specialization of Checker for *InputType* equal to value::Single.

`#include <keyvalue/key/Checker.h>`

Public Types

- typedef ConverterType::OutputType_ **OutputType_**

Public Member Functions

- void checkInput (const value::Single &input) const

  Checks input.

- void checkSoFar (const ConverterType &container) const

  Checks conversion process so far.

- void checkOutput (const OutputType_ &output) const

  Checks output.

Private Member Functions

- virtual string getName () const =0

  Gets Key's name.

10.10.1 Detailed Description

template<typename ConverterType> class keyvalue::key::Checker< ConverterType, value::Single >

Specialization of Checker for *InputType* equal to value::Single.

10.10.2 Member Function Documentation

10.10.2.1 void checkInput (const value::Single & *input*) const

Checks input.

Parameters

input : Data to be checked.

10.10.2.2 void checkSoFar (const ConverterType & *container*) const

Checks conversion process so far.

Although this method is very likely to be useless when input data is value::Single, this is not the case for other types of input data. For sake of generality, this must be implemented.

Parameters

container : The container.

10.10.2.3 void checkOutput (const OutputType_ & *output*) const

Checks output.

Parameters

output : Data to be checked.

10.10.2.4 virtual string getName ( ) const [private, pure virtual]

Gets Key's name.

Returns

The Key's name.

10.11 Checker< ConverterType, value::Vector > Class Template Reference

Specialization of Checker for *InputType* equal to value::Vector.

`#include <keyvalue/key/Checker.h>`

Public Types

- typedef ConverterType::OutputType_ **OutputType_**

Public Member Functions

- void checkInput (const value::Vector &input) const

  Check input.

- void checkSoFar (const ConverterType &container) const

  Check conversion process so far.

- void checkOutput (const OutputType_ &output) const

  Check output.

Private Member Functions

- virtual string getName () const =0
  Get Key's name.
- virtual void checkSize (size_t size) const
  Check input size.

10.11.1 Detailed Description

template<typename ConverterType> class keyvalue::key::Checker< ConverterType, value::Vector >

Specialization of Checker for *InputType* equal to value::Vector.

10.11.2 Member Function Documentation

10.11.2.1 void checkInput (const value::Vector & *input*) const

Check input.

The current implementation calls checkSize().

Parameters

input : Data to be checked.

10.11.2.2 void checkSoFar (const ConverterType & *container*) const

Check conversion process so far.

Parameters

container : The container.

10.11.2.3 void checkOutput (const OutputType_ & *output*) const

Check output.

Parameters

output : Data to be checked.

10.11.2.4 virtual string getName ( ) const [private, pure virtual]

Get Key's name.

Returns

The Key's name.

10.11.2.5 void checkSize (size_t *size*) const [private, virtual]

Check input size.

Parameters

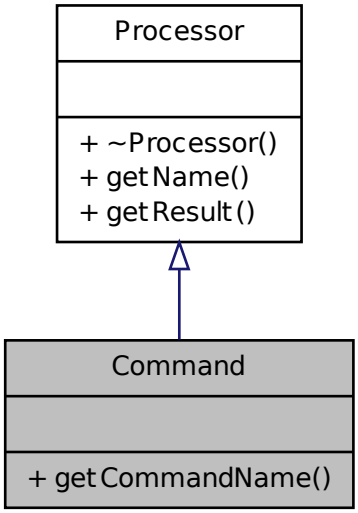
size : Input size.

10.12 Command Class Reference

Command interface.

`#include <keyvalue/mngt/Command.h>`

Inheritance diagram for Command:



Public Member Functions

- virtual const char * getCommandName () const =0

  Gets Command's name.

- virtual const char * getName () const =0

  Gets Processor's name.

- virtual value::Result getResult (const DataSet &data) const =0

  Gets result of processing a DataSet.

10.12.1 Detailed Description

Command interface. A Processor able to process an empty DataSet might be a Command.

For instance, if the Processor is Builder<*OutputType*>, for some *ObjectType*, then Builder<*OutputType*>::getObject(const DataSet& data) might do its job ignoring *data*. Another way is when the method does look up values in an empty *data* but, failing to find any, can still do its job considering default values for the searched keys (with or without intervention of DataSet Default).

In either cases above, the Builder<*OutputType*> might be declared as a command by `public`ly deriving from this `class`. Similar arguments hold for a Calculator<*Tag*>. The `public` derivation can be done directly but the use of DeclareBuilder.h or DeclareCalculator.h are preferable. (See Builder and Calculator for details.)

When a Processor is a Command some front-ends might take advantage of this fact and provide to their users some shortcut or menu entry to call the Processor without asking for additional user's input.

10.12.2 Member Function Documentation

10.12.2.1 virtual const char* getCommandName ( ) const [pure virtual]

Gets Command's name.

Returns

The name.

10.12.2.2 virtual const char* getName ( ) const [pure virtual, inherited]

Gets Processor's name.

Returns

The name.

Implemented in Builder< ObjectType >, and Calculator< Tag >.

10.12.2.3 virtual value::Result getResult (const DataSet & *data*) const [pure virtual, inherited]

Gets result of processing a DataSet.

This is a low level method which bypasses the memoization technique implemented by DataSet (see DataSet::mustUpdate()). Therefore, if *processor* is a Processor and *data* is a DataSet, then rather than

```
processor.getResult(data);
```

one should use

```
data.process(processor);
```

Actually, the latter calls the former adding memoization support.

Parameters

data : DataSet to be processed.

Returns

The result.

Implemented in Builder< ObjectType >, and Calculator< Tag >.

10.13 ConsoleLogger Class Reference

Implements a console Logger.

`#include <keyvalue/sys/logger/ConsoleLogger.h>`

Public Member Functions

- **ConsoleLogger** (unsigned int level, string title)

10.13.1 Detailed Description

Implements a console Logger. Essentially this `class` is either a WindowsConsole or a XtermConsole, depending on the OS.

10.14 DataSet Class Reference

A named map from keys to values.

`#include <keyvalue/mngt/DataSet.h>`

Classes

- struct Graph

  Simplified dependency graph.

- struct Record

  A wrapper around a value::Value.

Public Member Functions

- DataSet (const string &name)

  Constructs an empty DataSet with a given name.

- string getName () const

  Gets DataSet's name.

- void add (const string &key, const value::Value &value)

  Adds a key-value pair to the DataSet.

- template<typename KeyType >
  KeyType::OutputType_ * find (const KeyType &key) const

  Checks if key can be found.

- template<typename KeyType >
  KeyType::OutputType_ getValue (const KeyType &key) const

  Gets and processes value associated to a key.

- value::Result process () const

  Processes a DataSet.

- value::Result process (const Processor &processor) const

  Processes a DataSet provided the Processor.

- bool mustUpdate () const

  Checks if there is need for redoing the processing.

Private Types

- typedef std::map< string, shared_ptr< Record > > **MapType_**
- typedef MapType_::value_type **ValueType_**

Private Member Functions

- Graph getGraph (const string &key, bool start=true) const

  Gets the simplified dependency graph of a key.

- Graph **getGraph** (shared_ptr< Record > first, const string &key) const
- value::Result **doProcess** (const Processor &processor) const

Private Attributes

- string **name_**
- MapType_ **map_**
- bool **mustUpdate_**
- value::Result **result_**

Friends

- std::ostream & operator<< (std::ostream &os, const DataSet &dataSet)

  ostream operator<<() for DataSet.

10.14.1 Detailed Description

A named map from keys to values.

10.14.2 Constructor & Destructor Documentation

10.14.2.1 DataSet (const string & *name*) [explicit]

Constructs an empty DataSet with a given name.

Parameters

name : The DataSet's name.

10.14.3 Member Function Documentation

10.14.3.1 string getName ( ) const

Gets DataSet's name.

Returns

The name of this DataSet.

10.14.3.2 void add (const string & *key*, const value::Value & *value*)

Adds a key-value pair to the DataSet.

Parameters

key : The key;

value : The value.

10.14.3.3 KeyType::OutputType_ * find (const KeyType & *key*) const

Checks if key can be found.

This method checks if a key can be resolved. If resolution fails, then a null pointer is returned.

On the other hand. if resolution succeeds, this method processes the value associated to the key and in case of success a pointer to the processed value is returned. If the processing fails an exception indicating the error will be thrown.

This method is similar to getValue() below since both give access to the processed value corresponding to a specific key. The difference is how they report resolution failure. While find() returns a null pointer, getValue() throws an exception.

This method gives to the user the ability to set default values for unsolved keys:

```
// DataSet data;
// ...
//  const key::Single<bool> keyIsRegular("IsRegular");
```

```
bool isRegular(true); // Set default value
if (bool* ptr = data.find(keyIsRegular))
  isRegular = *ptr;
```

Parameters

key : The key to be checked.

Returns

If the key can be found, this method returns a pointer to the corresponding processed value. Otherwise it returns a null pointer.

Set dependency graph.

Memoization.

Check input container type.

Process raw value.

Cache output.

10.14.3.4 KeyType::OutputType_ getValue (const KeyType & *key*) const

Gets and processes value associated to a key.

Parameters

KeyType : (`template` parameter) Key's dynamic type;

key : Key associated to the value to be retrieved.

Returns

Processed value which is associated to *key*.

10.14.3.5 value::Result process ( ) const

Processes a DataSet.

If key::Processor can be resolved for this DataSet, then the corresponding Processor (retrieved from ProcessorMngr) will be passed to process(const Processor& processor).

Returns

Result of processing or a null pointer.

Exceptions

RuntimeError : If key::Processor cannot be resolved or the processor is not registered.

10.14.3.6 value::Result process (const Processor & *processor*) const

Processes a DataSet provided the Processor.

Given a Processor *processor*, if key::Processor can be resolved for this DataSet and its corresponding value matches processor's name (*i.e.* `processor.getName()`), then this DataSet will be processed by *processor* and the result will be returned. Otherwise, an exception will be thrown.

Parameters

processor : The Processor for this DataSet.

Returns

The result of processing.

ProcessDefault : When one tries to process "Default" DataSet.

Exceptions

RuntimeError : When the processing cannot be performed (see description above).

10.14.3.7 bool mustUpdate ( ) const

Checks if there is need for redoing the processing.

The two process() methods, pass this DataSet to the relevant Processor which will query for values through find() and getValue(). The memoization technique is applied: The DataSet will cache those values along the processing result. If the same processing is requested again, then the same values will be retrieved a second time.

The Processor can call this method which will compare the new and old (cached) set of values. If they agree this method returns `false` indicating that the result of previous processing is up to date and the Processor may return immediately.

Returns

This method returns `true` if cached result of previous processing is out of date. Otherwise, it returns `false`.

10.14.3.8 Graph getGraph (const string & *key*, bool *start* = `true`) const `[private]`

Gets the simplified dependency graph of a key.

If a key K_1 depends on K_2 which, in turn depends on K_3, and so on, up to K_n, then this method goes through all this dependency graph and returns a simplified version of it containing a pointer to K_1 and another to K_n. Additionally, when walking through the graph this method checks if any node has changed since the last time this graph was calculated.

Parameters

key : The starting key (K_1 above);

start : This method is recursive and must be able to detected the root call (the called from outside this function) from the others (called by itself). This flag by default is set to `true` to indicate calling from outside.

10.14.4 Friends And Related Function Documentation

10.14.4.1 std::ostream& operator<< (std::ostream & *os*, const DataSet & *dataSet*) [friend]

ostream operator<<() for DataSet.

10.15 Debug Class Reference

MessageImpl instantiation used for debug messages.

`#include <keyvalue/sys/message/MessageImpl.h>`

10.15.1 Detailed Description

MessageImpl instantiation used for debug messages. `typedef MessageImpl<4> Debug;`

10.16 Decreasing Class Reference

Decreasing monotone `class`.

`#include <keyvalue/key/generic/monotone/Decreasing.h>`

Static Public Member Functions

- template<typename ElementType >
  static bool check (const ElementType &x, const ElementType &y)

  Performs the comparison.

- static const char * getName ()

  Gets type name.

10.16.1 Detailed Description

Decreasing monotone `class`. Given two consecutive values of a sequence, *x*[i] and *x*[i+1], this `class` checks if *x*[i] >= *x*[i+1].

10.16.2 Member Function Documentation

10.16.2.1 static bool check (const ElementType & *x*, const ElementType & *y*) [static]

Performs the comparison.

Parameters

x : 1st value to be checked;

y : 2nd value to be checked.

ElementType : (`template` parameter) Type of x and y. Accepted types are `double`, `ptime`, `string` and `unsigned int`. (Other types will generate a link error.)

Returns

($x >= y$).

10.16.2.2 static const char* getName ( ) [static]

Gets type name.

Returns

"decreasing".

10.17 Default< OutputType > Struct Template Reference

Meta-function which defines Traits' default map type.

`#include <keyvalue/key/map/Default.h>`

10.17.1 Detailed Description

template<typename OutputType> struct keyvalue::key::Default< OutputType >

Meta-function which defines Traits' default map type. When *OutputType* is a basic type, then NoMap is selected. When it is an `enum` type, then FlagMap is selected. Otherwise, ObjectMap is chosen.

Parameters

OutputType : (`template` parameter) See Traits documentation.

10.18 DefaultMap< OutputType, isBasic > Struct Template Reference

Meta-function which defines Traits' default map type.

`#include <keyvalue/key/map/Default.h>`

10.18.1 Detailed Description

template<typename OutputType, int isBasic> struct keyvalue::key::DefaultMap< OutputType, isBasic >

Meta-function which defines Traits' default map type. The selected map is defined by parameter *Type*. If it is 0 then NoMap<*OutputType*> is selected. If it is 1, then FlagMap<*OutputType*> is selected. Finally if it is 2, then ObjectMap<*OutputType*> is chosen.

Parameters

OutputType : (`template` parameter) See Traits documentation;

Type : (`template` parameter) Defines if *OutputType* is basic, `enum` or object type. It has a default value which must not be changed.

Return values

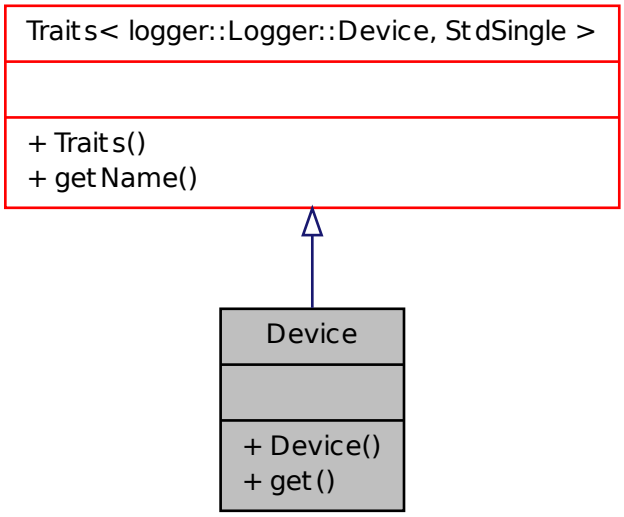
Map_ : The selected map.

10.19 Device Class Reference

Device key.

`#include <keyvalue/bridge/key/Device.h>`

Inheritance diagram for Device:



Public Types

- enum
- typedef Traits< logger::Logger::Device, StdSingle, Default > **Traits_**
- typedef StdSingle< typename Default< logger::Logger::Device >::OutputType_ > **ConverterType_**
- typedef ConverterType_::InputType_ **InputType_**
- typedef ConverterType_::OutputType_ **OutputType_**

Public Member Functions

- logger::Logger::Device get (const string &name) const

  Partially maps names into flags.

- string getName () const

  Gets Key's name.

- string getName () const

  Gets name.

- void setName (const string &name)

  Sets name.

10.19.1 Detailed Description

Device key. Key's name is "Device".

10.19.2 Member Function Documentation

10.19.2.1 logger::Logger::Device get (const string & *name*) const

Partially maps names into flags.

Maps "Console", "File", and "Standard", resp., to *Console*, *File*, and *Standard*.

Parameters

name : Name to be mapped;

Returns

Flag corresponding to name.

Exceptions

RuntimeError : If *name* is unknown.

10.19.2.2 string getName ( ) const [inherited]

Gets Key's name.

Returns

The Key's name.

10.19.2.3 string getName ( ) const [inherited]

Gets name.

Returns

The name.

10.19.2.4 void setName (const string & *name*) [inherited]

Sets name.

Parameters

name : The name.

10.20 Error Class Reference

MessageImpl instantiation used for error messages.

`#include <keyvalue/sys/message/MessageImpl.h>`

10.20.1 Detailed Description

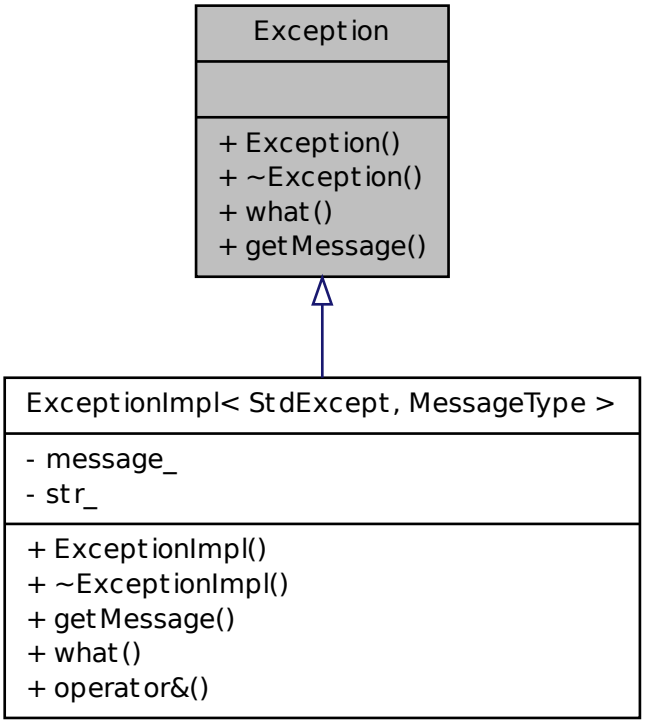
MessageImpl instantiation used for error messages. `typedef MessageImpl<0> Error;`

10.21 Exception Class Reference

Base protocol `class` for all exceptions.

`#include <keyvalue/sys/exception/Exception.h>`

Inheritance diagram for Exception:



Public Member Functions

- Exception ()

  Constructor (of course!).

- virtual const char * what () const =0

  Gets an explanatory C `string` *message about the exception.*

- virtual const Message & getMessage () const =0

  Gets the inner Message object stored by the exception.

10.21.1 Detailed Description

Base protocol `class` for all exceptions.

10.21.2 Constructor & Destructor Documentation

10.21.2.1 Exception ( )

Constructor (of course!).

Tip: Since all *KeyValue* exceptions are derived from this `class`, whenever an exception is created, this constructor is called. Hence, when debugging, if you wish to trace a thrown exception, then place a breakpoint inside the constructor and take a look at the call stack.

10.21.3 Member Function Documentation

10.21.3.1 virtual const char* what ( ) const [pure virtual]

Gets an explanatory C `string` message about the exception.

This methods exists for compatibility with `std::exception::what()`. Through a pointer or reference to either Exception or `std::exception` it is possible to get the same explanatory message.

By maintaining compatibility with `std::exception::what()`, *KeyValue* exceptions can be catch by many already implemented exception handlers. However, it is preferable to use *KeyValue's* exception management. In particular the method getMessage() is a better alternative to this one.

Returns

A C null terminated `char` string with a explanatory message about the exception.

Implemented in ExceptionImpl< StdExcept, MessageType >.

10.21.3.2 virtual const Message& getMessage ( ) const [pure virtual]

Gets the inner Message object stored by the exception.

Recall that Message objects carry more than just an explanatory text about the exception. It also contains extra information like its prefix and color, which allows a nicer presentation by a logger::Logger.

Returns

The inner Message object.

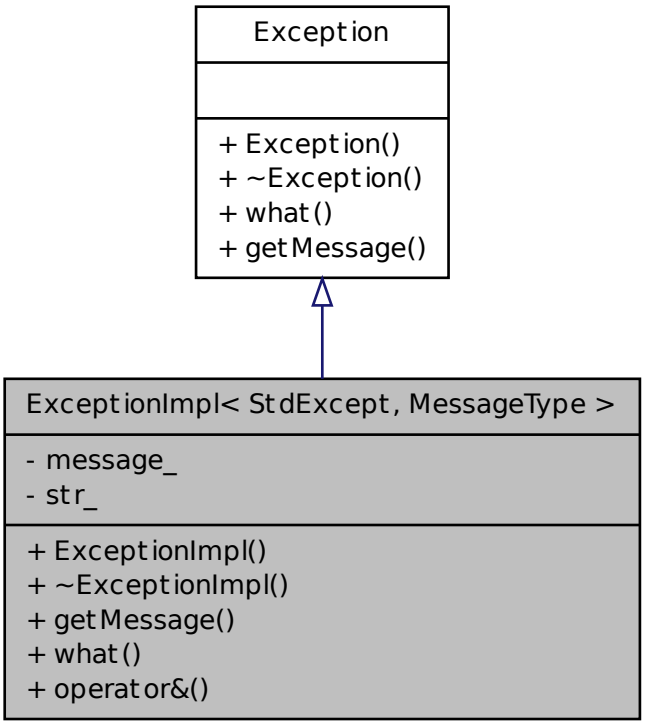
Implemented in ExceptionImpl< StdExcept, MessageType >.

10.22 ExceptionImpl< StdExcept, MessageType > Class Template Reference

Concrete `template class` which implements Exception's pure virtual methods.

`#include <keyvalue/sys/exception/Exception.h>`

Inheritance diagram for ExceptionImpl< StdExcept, MessageType >:



Public Member Functions

- const Message & getMessage () const

  Gets the inner Message.

- const char * what () const throw ()

  Gets a C string describing the exception.

- template<typename DataType >
  ExceptionImpl & operator& (const DataType &data)

  Appends formated data to a given exception.

Private Attributes

- MessageType **message_**
- string **str_**

10.22.1 Detailed Description

template<typename StdExcept, typename MessageType> class keyvalue::exception::ExceptionImpl< StdExcept, MessageType >

Concrete `template class` which implements Exception's pure virtual methods.

Parameters

StdExcept : (`template` parameter) Standard exception `class` that this `class` derives from (either `std::runtime_error` or `std::logic_error`);

MessageType : (`template` parameter) Type of inner Message object that might be gotten by getMessage().

10.22.2 Member Function Documentation

10.22.2.1 const Message & getMessage ( ) const [virtual]

Gets the inner Message.

Implements Exception.

10.22.2.2 const char * what ( ) const throw () [virtual]

Gets a C string describing the exception.

Implements Exception.

10.22.2.3 ExceptionImpl< StdExcept, MessageType > & operator& (const DataType & *data*)

Appends formated data to a given exception.

Recall that this `class` has a Message member. This method appends *data* to that member (see Message::operator&).

The typical use for this method is as follows:

```
if (price <= 0.0)
  throw RuntimeError() & "Invalid price. Expecting a positive number. Got " & price;
```

Parameters

DataType : (`template` parameter) The type of data to be appended. This parameter is automatically deduced by the compiler and may be ignored by the user.

exception : The exception that data should be appended to;

data : The data to be appended.

Returns

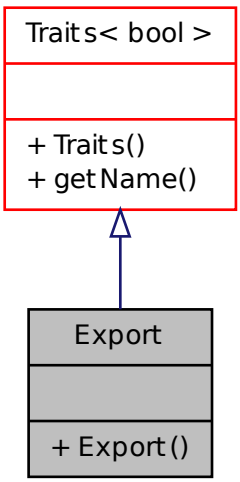
The exception with appended data.

10.23 Export Class Reference

Export key.

`#include <keyvalue/key/specific/Export.h>`

Inheritance diagram for Export:



Public Types

- enum
- typedef Traits< bool, StdSingle, Default > **Traits_**
- typedef StdSingle< typename Default< bool >::OutputType_ > **ConverterType_**
- typedef ConverterType_::InputType_ **InputType_**
- typedef ConverterType_::OutputType_ **OutputType_**

Public Member Functions

- Export ()

  Constructs key and sets its name.

- string getName () const

  Gets Key's name.

- string getName () const

  Gets name.

- void setName (const string &name)

Sets name.

10.23.1 Detailed Description

Export key. Key's name is "Export".

10.23.2 Constructor & Destructor Documentation

10.23.2.1 Export ( )

Constructs key and sets its name.

10.23.3 Member Function Documentation

10.23.3.1 string getName ( ) const [inherited]

Gets Key's name.

Returns

The Key's name.

10.23.3.2 string getName ( ) const [inherited]

Gets name.

Returns

The name.

10.23.3.3 void setName (const string & *name*) [inherited]

Sets name.

Parameters

name : The name.

10.24 LexicalToolKit::Failure Class Reference

This `class` is thrown when LexicalToolKit::convert() fails.

`#include <keyvalue/frontend/LexicalToolKit.h>`

Inherits exception::RuntimeError.

10.24.1 Detailed Description

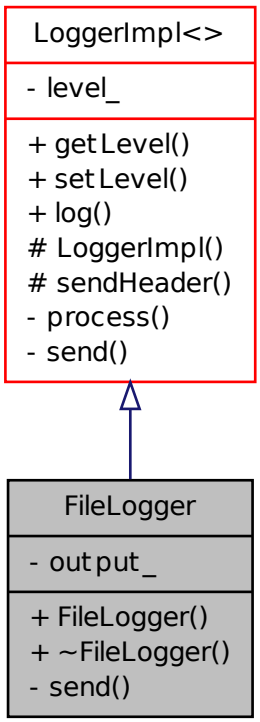
This `class` is thrown when LexicalToolKit::convert() fails.

10.25 FileLogger Class Reference

Implements a file Logger.

`#include <keyvalue/sys/logger/FileLogger.h>`

Inheritance diagram for FileLogger:



Public Types

- typedef LoggerImpl< AddPrefix, IgnoreColor, ForwardToGlobalLogger > **LoggerImpl_**
- enum **Device** { **Console**, **File**, **Standard** }

Public Member Functions

- **FileLogger** (unsigned int level, const string &fileName)
- unsigned int **getLevel** () const
- virtual unsigned int getLevel () const =0

  Gets Logger's current level.

- void **setLevel** (unsigned int level)

- virtual void setLevel (unsigned int level)=0

  Sets Logger's current level.

- bool **log** (const Message &message)
- virtual bool log (const Message &message)=0

  Logs a Message.

Protected Member Functions

- bool sendHeader ()

  Sends the header message.

- virtual bool **send** (const string &message)=0

Private Member Functions

- bool send (const string &message)

  Sends a raw `string` message to the Logger's underlying device.

Private Attributes

- std::ofstream **output_**

10.25.1 Detailed Description

Implements a file Logger.

10.25.2 Member Function Documentation

10.25.2.1 bool send (const string & *message*) [private, virtual]

Sends a raw `string` message to the Logger's underlying device.

Parameters

message : Message to be sent.

Returns

This method returns `false` if it fails. Otherwise, it returns `true`.

Implements LoggerImpl<>.

10.25.2.2 virtual unsigned int getLevel ( ) const [pure virtual, inherited]

Gets Logger's current level.

Returns

The level.

Implemented in LoggerImpl< PrefixPolicy, ColorPolicy, SafetyPolicy >.

10.25.2.3 virtual void setLevel (unsigned int *level*) [pure virtual, inherited]

Sets Logger's current level.

Parameters

level : New level.

Implemented in LoggerImpl< PrefixPolicy, ColorPolicy, SafetyPolicy >.

10.25.2.4 virtual bool log (const Message & *message*) [pure virtual, inherited]

Logs a Message.

This method may fail if the implemented Logger cannot process the message (e.g. a console Logger that has no longer a console window).

Parameters

message : Message to be logged.

Returns

This method returns `true` if it is successful.

Implemented in LoggerImpl< PrefixPolicy, ColorPolicy, SafetyPolicy >.

10.25.2.5 bool sendHeader ( ) [protected, inherited]

Sends the header message.

This method sends (throw send()) an opening header message to the underlying device of the Logger.

Returns

This method returns what print() does.

10.26 XtermConsole::FileRaii Class Reference

XtermConsole's helper `class` for file management.

Public Member Functions

- **FileRaii** (const int fd=-1)
- FileRaii & **operator=** (const int fd)
- void **close** ()
- int **getFd** () const

Private Attributes

- int **fd_**

10.26.1 Detailed Description

XtermConsole's helper `class` for file management. This private `class` deals with (C) low level file descriptors needed by XtermConsole.

10.27 FlagMap< OutputType > Class Template Reference

Maps names to constants.

`#include <keyvalue/key/map/FlagMap.h>`

Public Types

- typedef OutputType **OutputType_**

Public Member Functions

- OutputType map (const value::Variant &variant) const

  Performs the mapping.

- virtual OutputType get (const string &name) const =0

  Performs the mapping from `string` to OutputType.

Private Member Functions

- virtual string getName () const =0

  Gets Key's name.

10.27.1 Detailed Description

template<typename OutputType> class keyvalue::key::FlagMap< OutputType >

Maps names to constants. A `string` is mapped to a flag (normally an `enum`). Consider, for instance, key::Device which specifies a logger::Logger type to be build. Users provide a `string` either "Console",

"File" or "Standard". However, inside keyvalue, devices are designated by key::Device::Flag. An error occurs when a `string` cannot be mapped to a valid output.

Traits derived `class`es which uses this map must implement a method to perform the mapping. This method has the following signature

```
OutputType get(const string& name) const;
```

Parameters

OutputType : (`template` parameter) Output type.

Return values

OutputType_ : Same as *OutputType*.

10.27.2 Member Function Documentation

10.27.2.1 OutputType map (const value::Variant & *variant*) const

Performs the mapping.

This method converts the input from value::Variant to `string` and calls get().

Parameters

variant : A value::Variant containing the input `string` to be mapped.

Returns

The mapping result.

10.27.2.2 virtual OutputType get (const string & *name*) const [pure virtual]

Performs the mapping from `string` to *OutputType*.

Must be implemented by derived real keys.

Parameters

name : The `string` to be mapped.

Returns

The mapped value.

10.27.2.3 virtual string getName ( ) const [private, pure virtual]

Gets Key's name.

Returns

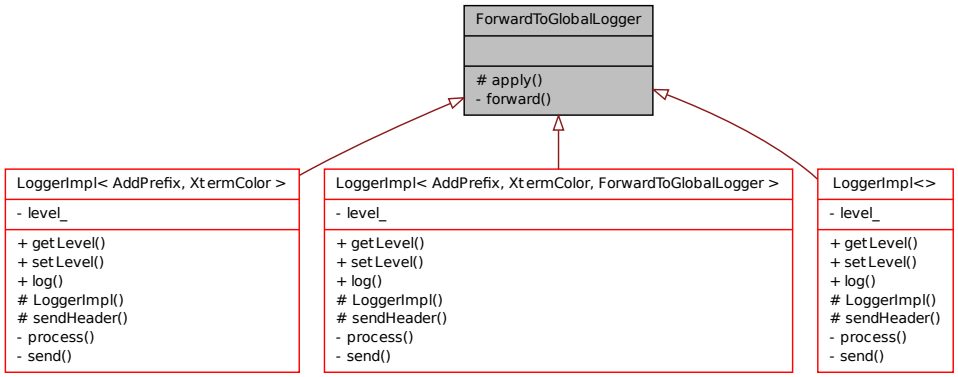
The Key's name.

10.28 ForwardToGlobalLogger Class Reference

Defines a policy to apply when the Logger fails to process a Message, namely, forward it to GlobalLogger.

`#include <keyvalue/sys/logger/policy/ForwardToGlobalLogger.h>`

Inheritance diagram for ForwardToGlobalLogger:



Protected Member Functions

- bool apply (const Logger *logger, const Message &message) const

  Applies ForwardToGlobalLogger policy.

Private Member Functions

- void forward (const Logger *logger, const Message &message) const

  Forwards a Message to GlobalLogger.

10.28.1 Detailed Description

Defines a policy to apply when the Logger fails to process a Message, namely, forward it to GlobalLogger. If the GlobalLogger is the one that fails, it is very likely that it can no longer receive Messages. For that reason, this policy resets GlobalLogger to its default(which supposedly never fails) before the forwarding.

10.28.2 Member Function Documentation

10.28.2.1 bool apply (const Logger * *logger*, const Message & *message*) const [protected]

Applies ForwardToGlobalLogger policy.

Returns

This method returns `false` to indicate the previous failure.

10.28.2.2 void forward (const Logger * *logger,* const Message & *message*) const [private]

Forwards a Message to GlobalLogger.

This method receives a (raw) pointer to the Logger that has failed. If this pointer points to GlobalLogger's current registered Logger, then this method resets the GlobalLogger to its default Logger (which supposedly never fails).

Parameters

logger : A pointer to the Logger which has failed;

message : The message to be forwarded.

10.29 FrontEnd Class Reference

Front-end's specific data.

`#include <keyvalue/frontend/FrontEnd.h>`

Public Member Functions

- virtual string getPath () const =0

  Gets the front-end's execution path.

- LexicalToolKit & lexicalToolKit ()

  Access the FrontEnd's lexical tool kit.

Protected Attributes

- LexicalToolKit **lexicalToolKit_**

10.29.1 Detailed Description

Front-end's specific data. Use util::Global to access the global FrontEnd.

10.29.2 Member Function Documentation

10.29.2.1 virtual string getPath ( ) const [pure virtual]

Gets the front-end's execution path.

Returns

The path.

10.29.2.2 LexicalToolKit& lexicalToolKit ( )

Access the FrontEnd's lexical tool kit.

Returns

The lexical tool kit.

10.30 Geq< ElementType > Class Template Reference

Greater-than-or-equal-to bound `class`.

`#include <keyvalue/key/generic/bound/Geq.h>`

Public Member Functions

- Geq (const ElementType &bound)
  Constructs and sets bound.
- bool check (const ElementType &x) const
  Performs the check.
- const ElementType & getBound () const
  Gets bound value.
- const char * getName () const
  Gets comparison name.

Private Attributes

- const ElementType **bound_**

10.30.1 Detailed Description

template<typename ElementType> class keyvalue::key::Geq< ElementType >

Greater-than-or-equal-to bound `class`. Given a value *x* this `class` checks if *x* >= *bound*, where *bound* is fixed at construction time.

Parameters

ElementType : (`template` parameter) Type of *bound*. Accepted types are `double`, `ptime`, `string` and `unsigned int`. (Other types will generate a link error.)

10.30.2 Constructor & Destructor Documentation

10.30.2.1 Geq (const ElementType & *bound*) [explicit]

Constructs and sets bound.

Parameters

bound : The bound value.

10.30.3 Member Function Documentation

10.30.3.1 bool check (const ElementType & *x*) const

Performs the check.

Parameters

x : Value to be checked.

Returns

x >= *bound*.

10.30.3.2 const ElementType& getBound ( ) const

Gets bound value.

Returns

The bound.

10.30.3.3 const char* getName ( ) const

Gets comparison name.

Returns

This method returns "greater than or equal to" unless *ElementType* is `ptime`, in which case it returns "latter than or equal to".

10.31 Global< ObjectType > Class Template Reference

Manager of global objects.

`#include <keyvalue/util/Global.h>`

Static Public Member Functions

- static shared_ptr< ObjectType > get ()

  Gets registered object.

- static void set (shared_ptr< ObjectType > object)

  Registers a new object.

Private Member Functions

- Global ()

  Constructor.

- template<>
  Global ()

Static Private Member Functions

- static Global & getInstance ()

  Gets the unique instance of this `class`.

Private Attributes

- shared_ptr< ObjectType > **object_**

10.31.1 Detailed Description

template<typename ObjectType> class keyvalue::util::Global< ObjectType >

Manager of global objects. Some objects could be global variables and some types could be singletons (*e.g.* Repository). However, both global variables and singletons are heavily criticized since they introduce state into the application making testing harder. Additionally, static/global objects adds complications to multi-threaded applications. This `template class` is an attempt to deal with this dilemma.

The instantiation for a given type manages a pointer to an object of this type and provides a global access to it.

Normally, users change the managed at most once, at initialization time. Testers, however might change it to point to mock objects.

Each instantiation of this `template class` is a singleton which is very simple and supposedly bug free and (hopefully in the future) thread-safe. However, the unique instance is not accessible externally. Only `static` methods, giving access to the pointer managed by the unique instance, are available.

Parameters

ObjectType : The type of object managed.

Todo

Consider thread-safety issues.

10.31.2 Constructor & Destructor Documentation

10.31.2.1 Global ( ) `[private]`

Constructor.

The general implementation calls *ObjectType*'s default constructor to build the object that the managed pointer points to. This behavior can be changed by specializations. For instance,

Global<logger::Logger>::Global() sets the initial object to be a logger::StdLogger with some specified level.

We emphasize that when the constructor's default behavior is not adequate for one's needs, then only the constructor needs to be declared and implemented by specializations. (See keyvalue/sys/logger/Logger.h and keyvalue/sys/logger/Logger.cpp.)

10.31.2.2 Global ( ) [private]

Recall that the default implementation of Global<ObjectType>'s constructor build the global *ObjectType* by calling its default constructor. Since frontend::FrontEnd is abstract the default behavior is not adequate. Therefore, we need to declare and implement a specialization.

Here the specialization is declared but not implemented. Actually its implementation comes togheter with frontend::FrontEnd's (concrete) derived `class`.

10.31.3 Member Function Documentation

10.31.3.1 shared_ptr< ObjectType > get ( ) [static]

Gets registered object.

Debug build throws an exception if the internal pointer has not been previously initialized. Release build has undefined behavior in this case.

Returns

A pointer to registered object.

Exceptions

keyvalue::LogicError (Debug build only) If internal pointer has not been initialized.

10.31.3.2 void set (shared_ptr< ObjectType > *object*) [static]

Registers a new object.

Parameters

object : Pointer to the object to be registered.

10.31.3.3 Global< ObjectType > & getInstance ( ) [static, private]

Gets the unique instance of this `class`.

Returns

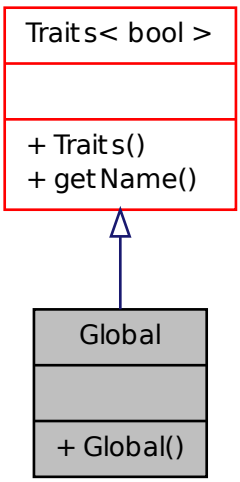
A reference to the unique instance.

10.32 Global Class Reference

Global key.

`#include <keyvalue/bridge/key/Global.h>`

Inheritance diagram for Global:



Public Types

- enum
- typedef Traits< bool, StdSingle, Default > **Traits_**
- typedef StdSingle< typename Default< bool >::OutputType_ > **ConverterType_**
- typedef ConverterType_::InputType_ **InputType_**
- typedef ConverterType_::OutputType_ **OutputType_**

Public Member Functions

- Global ()

  Constructs key and set its name.

- string getName () const

  Gets Key's name.

- string getName () const

  Gets name.

- void setName (const string &name)

Sets name.

10.32.1 Detailed Description

Global key. Key's name is "Global".

10.32.2 Constructor & Destructor Documentation

10.32.2.1 Global ( )

Constructs key and set its name.

10.32.3 Member Function Documentation

10.32.3.1 string getName ( ) const [inherited]

Gets Key's name.

Returns

The Key's name.

10.32.3.2 string getName ( ) const [inherited]

Gets name.

Returns

The name.

10.32.3.3 void setName (const string & *name*) [inherited]

Sets name.

Parameters

name : The name.

10.33 DataSet::Graph Struct Reference

Simplified dependency graph.

Public Member Functions

- **Graph** (shared_ptr< Record > first, shared_ptr< Record > last, bool hasChanged)

Public Attributes

- shared_ptr< Record > **first_**
- shared_ptr< Record > **last_**
- bool **hasChanged_**

10.33.1 Detailed Description

Simplified dependency graph. Keys might import their values from other keys. This reference system creates complex dependencies between different keys. This struct is a very simplified version of the dependency tree of a key. (See getGraph().)

10.34 Greater< ElementType > Class Template Reference

Greater-than bound `class`.

`#include <keyvalue/key/generic/bound/Greater.h>`

Public Member Functions

- Greater (const ElementType &bound)

  Constructs and sets bound.

- bool check (const ElementType &x) const

  Performs the check.

- const ElementType & getBound () const

  Gets bound value.

- const char * getName () const

  Gets comparison name.

Private Attributes

- const ElementType **bound_**

10.34.1 Detailed Description

template<typename ElementType> class keyvalue::key::Greater< ElementType >

Greater-than bound `class`. Given a value *x* this `class` checks if *x* > *bound*, where *bound* is fixed at construction time.

Parameters

ElementType : (`template` parameter) Type of *bound*. Accepted types are `double`, `ptime`, `string` and `unsigned int`. (Other types will generate a link error.)

10.34.2 Constructor & Destructor Documentation

10.34.2.1 Greater (const ElementType & *bound*) [explicit]

Constructs and sets bound.

Parameters

bound : The bound value.

10.34.3 Member Function Documentation

10.34.3.1 bool check (const ElementType & *x*) const

Performs the check.

Parameters

x : Value to be checked.

Returns

x > *bound*.

10.34.3.2 const ElementType& getBound ( ) const

Gets bound value.

Returns

The bound.

10.34.3.3 const char* getName ( ) const

Gets comparison name.

Returns

This method returns "greater than" unless *ElementType* is `ptime`, in which case it returns "latter than".

10.35 LexicalToolKit::Helper< From, To > Struct Template Reference

Helper `class` of LexicalToolKit.

`#include <LexicalToolKit.h>`

Public Types

- typedef To(* **Type_**)(const From &)

Static Public Attributes

- static Type_ LexicalToolKit::* **converter_**
- static IO LexicalToolKit::* **io_**

10.35.1 Detailed Description

template<typename From, typename To> struct keyvalue::frontend::LexicalToolKit::Helper< From, To >

Helper `class` of LexicalToolKit. This is a meta-function that returns pointers to LexicalToolkit members corresponding to a specific conversion.

Parameters

From : (`template` parameter) Conversion input type;

To : (`template` parameter) Conversion output type.

Return values

Type_ : Converter's type (signature);

converter_ : Pointer to LexicalToolkit's converter member;

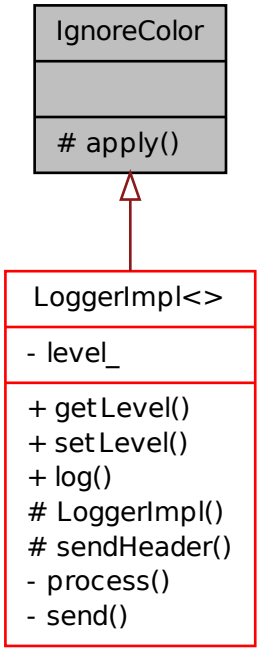
io_ : Pointer to LexicalToolkit's direction member.

10.36 IgnoreColor Class Reference

Defines a policy for Message colors, which is, to ignore them.

`#include <keyvalue/sys/logger/policy/IgnoreColor.h>`

Inheritance diagram for IgnoreColor:



Protected Member Functions

- bool **apply** (const Message &) const

10.36.1 Detailed Description

Defines a policy for Message colors, which is, to ignore them.

10.37 IgnoreFailure Class Reference

Defines a policy to apply when the logger fails, which is, to ignore the failure.

`#include <keyvalue/sys/logger/policy/IgnoreFailure.h>`

Protected Member Functions

- bool apply (const Message &message)

  Applies IgnoreFailure policy.

10.37.1 Detailed Description

Defines a policy to apply when the logger fails, which is, to ignore the failure. WARNING: This is a dangerous policy to apply. Use with caution (or do not use at all).

10.37.2 Member Function Documentation

10.37.2.1 bool apply (const Message & *message*) [protected]

Applies IgnoreFailure policy.

Returns

This method returns `true` to ignore previous failure.

10.38 IgnorePrefix Class Reference

Defines a policy for Message prefixes, which is, to ignore them.

`#include <keyvalue/sys/logger/policy/IgnorePrefix.h>`

Protected Member Functions

- bool **apply** (const Message &message)

10.38.1 Detailed Description

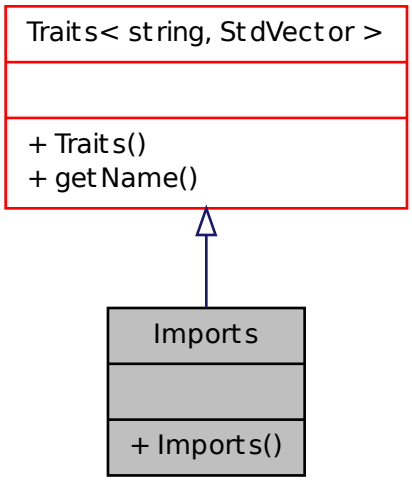
Defines a policy for Message prefixes, which is, to ignore them.

10.39 Imports Class Reference

Imports key.

`#include <keyvalue/key/specific/Imports.h>`

Inheritance diagram for Imports:



Public Types

- enum
- typedef Traits< string, StdVector, Default > **Traits_**
- typedef StdVector< typename Default< string >::OutputType_ > **ConverterType_**
- typedef ConverterType_::InputType_ **InputType_**
- typedef ConverterType_::OutputType_ **OutputType_**

Public Member Functions

- Imports ()
  Constructs key and sets its name.
- string getName () const
  Gets Key's name.
- string getName () const
  Gets name.
- void setName (const string &name)
  Sets name.

10.39.1 Detailed Description

Imports key. Key's name is "Imports".

10.39.2 Constructor & Destructor Documentation

10.39.2.1 Imports ( )

Constructs key and sets its name.

10.39.3 Member Function Documentation

10.39.3.1 string getName ( ) const [inherited]

Gets Key's name.

Returns

The Key's name.

10.39.3.2 string getName ( ) const [inherited]

Gets name.

Returns

The name.

10.39.3.3 void setName (const string & *name*) [inherited]

Sets name.

Parameters

name : The name.

10.40 Increasing Class Reference

Increasing monotone `class`.

`#include <keyvalue/key/generic/monotone/Increasing.h>`

Static Public Member Functions

- template<typename ElementType >
  static bool check (const ElementType &x, const ElementType &y)

  Performs the comparison.

- static const char * getName ()

  Gets type name.

10.40.1 Detailed Description

Increasing monotone `class`. Given two consecutive values of a sequence, *x*[i] and *x*[i+1], this `class` checks if *x*[i] <= *x*[i+1].

10.40.2 Member Function Documentation

10.40.2.1 static bool check (const ElementType & *x*, const ElementType & *y*) [static]

Performs the comparison.

Parameters

x : 1st value to be checked;

y : 2nd value to be checked.

ElementType : (`template` parameter) Type of *x* and *y*. Accepted types are `double`, `ptime`, `string` and `unsigned int`. (Other types will generate a link error.)

Returns

(*x* <= *y*).

10.40.2.2 static const char* getName ( ) [static]

Gets type name.

Returns

"increasing".

10.41 Info Class Reference

MessageImpl instantiation used for info messages.

`#include <keyvalue/sys/message/MessageImpl.h>`

10.41.1 Detailed Description

MessageImpl instantiation used for info messages. `typedef MessageImpl<1> Info;`

10.42 IsBasic< ElementType > Struct Template Reference

This meta-function checks if a type is either `bool`, `double`, `string`, `ptime` or `unsigned int` or not.

`#include <keyvalue/util/IsBasic.h>`

Public Types

- enum { **value** = 0 }

10.42.1 Detailed Description

template<typename ElementType> struct keyvalue::util::IsBasic< ElementType >

This meta-function checks if a type is either `bool`, `double`, `string`, `ptime` or `unsigned int` or not.

Parameters

ElementType : The type to be checked.

Return values

IsBasic<ElementType>::value : 1 if *ElementType* is either `bool`, `double`, `string`, `ptime` or `unsigned int`; 0 otherwise.

10.43 IsBasicOrEnum< ElementType > Struct Template Reference

This meta-function checks if a type is either `bool`, `double`, `string`, `ptime`, `unsigned int` or enum or not.

`#include <keyvalue/util/IsBasic.h>`

Public Types

- enum { **value** }

10.43.1 Detailed Description

template<typename ElementType> struct keyvalue::util::IsBasicOrEnum< ElementType >

This meta-function checks if a type is either `bool`, `double`, `string`, `ptime`, `unsigned int` or enum or not.

Parameters

ElementType : The type to be checked.

Return values

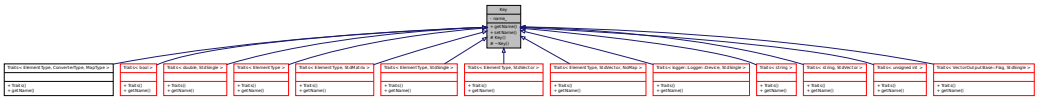
IsBasic<ElementType>::value_ : 1 if *ElementType* is either `bool`, `double`, `string`, `ptime`, `unsigned int` or enum; 0 otherwise.

10.44 Key Class Reference

Base for all real keys.

`#include <keyvalue/key/Key.h>`

Inheritance diagram for Key:



Public Member Functions

- string getName () const

  Gets name.

- void setName (const string &name)

  Sets name.

Protected Member Functions

- Key (const string &name)

  Constructs a Key with a specific name.

Private Attributes

- string **name_**

10.44.1 Detailed Description

Base for all real keys. Every Key has a (`string`) name, which is used by DataSet::getValue() for value look-up. The name must be provided on construction.

Instead of directly derive from this `class`, any `class` encapsulating information on keys must derive from Traits `template` which, in turn, derives from this `class`.

10.44.2 Constructor & Destructor Documentation

10.44.2.1 Key (const string & *name*) [protected]

Constructs a Key with a specific name.

Parameters

name : The name.

10.44.3 Member Function Documentation

10.44.3.1 string getName ( ) const

Gets name.

Returns

The name.

10.44.3.2 void setName (const string & *name*)

Sets name.

Parameters

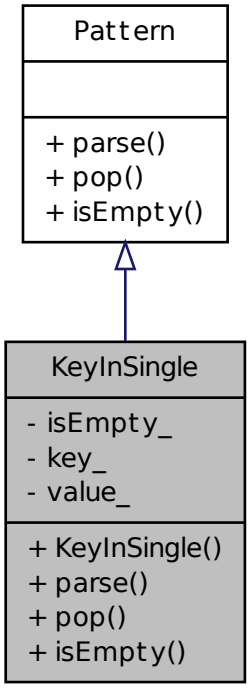
name : The name.

10.45 KeyInSingle Class Reference

Pattern where a value::Single object, recognized as a key, is followed by any value::Value.

`#include <keyvalue/pattern/KeyInSingle.h>`

Inheritance diagram for KeyInSingle:



Public Member Functions

- bool parse (frontend::Queue &queue)

  Parses the beginning of a frontend::Queue.

- std::pair< string, value::Value > pop ()

  Gets and removes the next stored key-value pair.

- bool isEmpty () const

  Checks if the list of recognized key-value pairs is empty.

Private Attributes

- bool **isEmpty_**
- string **key_**
- value::Value **value_**

10.45.1 Detailed Description

Pattern where a value::Single object, recognized as a key, is followed by any value::Value. The content of value::Value object will be the value associated to the key. No further checks are performed.

10.45.2 Member Function Documentation

10.45.2.1 bool parse (frontend::Queue & *queue*) [virtual]

Parses the beginning of a frontend::Queue.

When a pattern is recognized, the value::Value objects which make it are removed from frontend::Queue and all corresponding key-value pairs are stored inside the this `class` for later queries.

Parameters

queue : frontend::Queue to be parsed.

Returns

If the pattern is recognized, then this method returns `true`. Otherwise, it returns `false`.

Implements Pattern.

10.45.2.2 std::pair<string, value::Value> pop ( ) [virtual]

Gets and removes the next stored key-value pair.

The debug version checks if there is any key-value pair be popped (by calling isEmpty()) and throw an exception when the check fails. Release version has undefined behavior when the list is empty.

Returns

The key-value pair.

Exceptions

LogicError : (Debug build only) When the list is empty.

Implements Pattern.

10.45.2.3 bool isEmpty ( ) const [virtual]

Checks if the list of recognized key-value pairs is empty.

Returns

If the list is empty, this method returns `true`. Otherwise, it returns `false`.

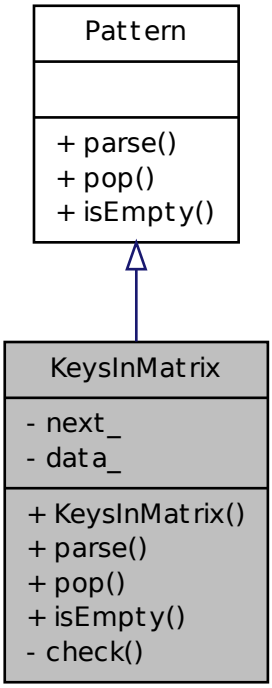
Implements Pattern.

10.46 KeysInMatrix Class Reference

Pattern made by a value::Matrix where all elements either in the first row or in the first column are keys.

`#include <keyvalue/pattern/KeysInMatrix.h>`

Inheritance diagram for KeysInMatrix:



Public Member Functions

- bool parse (frontend::Queue &queue)

  Parses the beginning of a frontend::Queue.

- std::pair< string, value::Value > pop ()

  Gets and removes the next stored key-value pair.

- bool isEmpty () const

  Checks if the list of recognized key-value pairs is empty.

Private Member Functions

- bool check (const value::Matrix &matrix, bool transpose=false)

This method does all the job explained in the `class` description.

Private Attributes

- size_t **next_**
- vector< std::pair< string, value::Value > > **data_**

10.46.1 Detailed Description

Pattern made by a value::Matrix where all elements either in the first row or in the first column are keys. More precisely, it is checked if the value::Matrix, *M=M(i, j)*, has the following properties:

- *M* has at least two rows;
- *M(0, j)* is a key for all *j*;
- *M(1, 0)* is not a key.

If *M* pass this check, then for each column of *M*, its first entry will be a key corresponding to the value given by the value::Vector (or value::Single, when *M* has only two rows) made by the other entries in this column. If *M* does not pass the check above, then the same test and key-value pattern parsing is performed on its transpose.

10.46.2 Member Function Documentation

10.46.2.1 bool parse (frontend::Queue & *queue*) [virtual]

Parses the beginning of a frontend::Queue.

When a pattern is recognized, the value::Value objects which make it are removed from frontend::Queue and all corresponding key-value pairs are stored inside the this `class` for later queries.

Parameters

queue : frontend::Queue to be parsed.

Returns

If the pattern is recognized, then this method returns `true`. Otherwise, it returns `false`.

Implements Pattern.

10.46.2.2 std::pair<string, value::Value> pop ( ) [virtual]

Gets and removes the next stored key-value pair.

The debug version checks if there is any key-value pair be popped (by calling isEmpty()) and throw an exception when the check fails. Release version has undefined behavior when the list is empty.

Returns

The key-value pair.

Exceptions

LogicError : (Debug build only) When the list is empty.

Implements Pattern.

10.46.2.3 bool isEmpty ( ) const [virtual]

Checks if the list of recognized key-value pairs is empty.

Returns

If the list is empty, this method returns `true`. Otherwise, it returns `false`.

Implements Pattern.

10.46.2.4 bool check (const value::Matrix & *matrix*, bool *transpose* = *false*) [private]

This method does all the job explained in the `class` description.

Parameters

matrix : Input matrix;

transpose : If `true`, the the test is performed on *matrix* transpose.

Returns

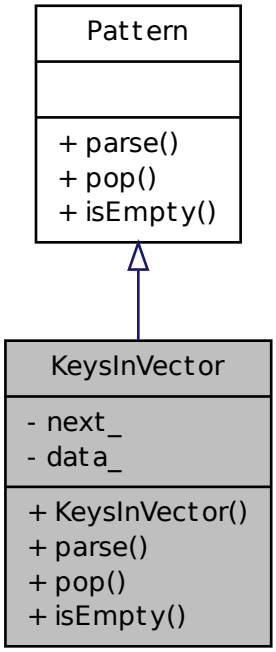
This method returns `true` if the pattern is recognized. Otherwise it returns `false`.

10.47 KeysInVector Class Reference

Pattern made by a value::Vector whose entries are keys followed by a value::Vector or value::Matrix.

`#include <keyvalue/pattern/KeysInVector.h>`

Inheritance diagram for KeysInVector:



Public Member Functions

- bool parse (frontend::Queue &queue)

  Parses the beginning of a frontend::Queue.

- std::pair< string, value::Value > pop ()

  Gets and removes the next stored key-value pair.

- bool isEmpty () const

  Checks if the list of recognized key-value pairs is empty.

Private Attributes

- size_t **next_**
- vector< std::pair< string, value::Value > > **data_**

10.47.1 Detailed Description

Pattern made by a value::Vector whose entries are keys followed by a value::Vector or value::Matrix. The pattern is defined by a value::Vector *K*, where all entries are keys, followed by a value::Value object *V* which is either a value::Vector or value::Matrix such that

- *V* has *n* rows if *K* is a column value::Vector of size *n*;
- *V* has *n* columns if *K* is a row value::Vector of size *n*.

In the first case the *n-th* row of *V* (which is either a value::Single or a row value::Vector) is considered the value associated to the *n-th* key in *K*.

Similarly, in the second case the *n-th* column of *V* is the value associated to the *n-th* key in *K*.

10.47.2 Member Function Documentation

10.47.2.1 bool parse (frontend::Queue & *queue*) [virtual]

Parses the beginning of a frontend::Queue.

When a pattern is recognized, the value::Value objects which make it are removed from frontend::Queue and all corresponding key-value pairs are stored inside the this `class` for later queries.

Parameters

queue : frontend::Queue to be parsed.

Returns

If the pattern is recognized, then this method returns `true`. Otherwise, it returns `false`.

Implements Pattern.

10.47.2.2 std::pair<string, value::Value> pop ( ) [virtual]

Gets and removes the next stored key-value pair.

The debug version checks if there is any key-value pair be popped (by calling isEmpty()) and throw an exception when the check fails. Release version has undefined behavior when the list is empty.

Returns

The key-value pair.

Exceptions

LogicError : (Debug build only) When the list is empty.

Implements Pattern.

10.47.2.3 bool isEmpty ( ) const [virtual]

Checks if the list of recognized key-value pairs is empty.

Returns

If the list is empty, this method returns `true`. Otherwise, it returns `false`.

Implements Pattern.

10.48 Leq< ElementType > Class Template Reference

Less-than-or-equal-to bound `class`.

`#include <keyvalue/key/generic/bound/Leq.h>`

Public Member Functions

- Leq (const ElementType &bound)

  Constructs and sets bound.

- bool check (const ElementType &x) const

  Performs the check.

- const ElementType & getBound () const

  Gets bound value.

- const char * getName () const

  Gets comparison name.

Private Attributes

- const ElementType **bound_**

10.48.1 Detailed Description

template<typename ElementType> class keyvalue::key::Leq< ElementType >

Less-than-or-equal-to bound `class`. Given a value *x* this `class` checks if *x* <= *bound*, where *bound* is fixed at construction time.

Parameters

ElementType : (`template` parameter) Type of *bound*. Accepted types are `double`, `ptime`, `string` and `unsigned int`. (Other types will generate a link error.)

10.48.2 Constructor & Destructor Documentation

10.48.2.1 Leq (const ElementType & *bound*) [explicit]

Constructs and sets bound.

Parameters

bound : The bound value.

10.48.3 Member Function Documentation

10.48.3.1 bool check (const ElementType & *x*) const

Performs the check.

Parameters

x : Value to be checked.

Returns

x <= *bound*.

10.48.3.2 const ElementType& getBound ( ) const

Gets bound value.

Returns

The bound.

10.48.3.3 const char* getName ( ) const

Gets comparison name.

Returns

This method returns "less than or equal to" unless *ElementType* is `ptime`, in which case it returns "earlier than or equal to".

10.49 Less< ElementType > Class Template Reference

Less-than bound `class`.

`#include <keyvalue/key/generic/bound/Less.h>`

Public Member Functions

- Less (const ElementType &bound)

Constructs and sets bound.

- bool check (const ElementType &x) const

  Performs the check.

- const ElementType & getBound () const

  Gets bound value.

- const char * getName () const

  Gets comparison name.

Private Attributes

- const ElementType **bound_**

10.49.1 Detailed Description

template<typename ElementType> class keyvalue::key::Less< ElementType >

Less-than bound `class`. Given a value x this `class` checks if $x < bound$, where *bound* is fixed at construction time.

Parameters

ElementType : (`template` parameter) Type of *bound*. Accepted types are `double`, `ptime`, `string` and `unsigned int`. (Other types will generate a link error.)

10.49.2 Constructor & Destructor Documentation

10.49.2.1 Less (const ElementType & *bound*) [explicit]

Constructs and sets bound.

Parameters

bound : The bound value.

10.49.3 Member Function Documentation

10.49.3.1 bool check (const ElementType & *x*) const

Performs the check.

Parameters

x : Value to be checked.

Returns

x > *bound*.

10.49.3.2 const ElementType& getBound ( ) const

Gets bound value.

Returns

The bound.

10.49.3.3 const char* getName ( ) const

Gets comparison name.

Returns

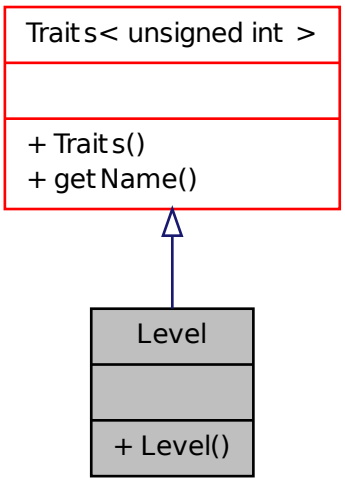
This method returns "less than" unless *ElementType* is `ptime`, in which case it returns "earlier than".

10.50 Level Class Reference

Level key.

`#include <keyvalue/bridge/key/Level.h>`

Inheritance diagram for Level:



Public Types

- enum
- typedef Traits< unsigned int, StdSingle, Default > **Traits_**

- typedef StdSingle< typename Default< unsigned int >::OutputType_ > **ConverterType_**
- typedef ConverterType_::InputType_ **InputType_**
- typedef ConverterType_::OutputType_ **OutputType_**

Public Member Functions

- Level ()

  Constructs key and sets its name.

- string getName () const

  Gets Key's name.

- string getName () const

  Gets name.

- void setName (const string &name)

  Sets name.

10.50.1 Detailed Description

Level key. Key's name is "Level".

10.50.2 Constructor & Destructor Documentation

10.50.2.1 Level ( )

Constructs key and sets its name.

10.50.3 Member Function Documentation

10.50.3.1 string getName ( ) const [inherited]

Gets Key's name.

Returns

The Key's name.

10.50.3.2 string getName ( ) const [inherited]

Gets name.

Returns

The name.

10.50.3.3 void setName (const string & *name*) [inherited]

Sets name.

Parameters

name : The name.

10.51 LexicalToolKit Class Reference

Manager of all lexical converters needed for front-end input/output.

`#include <keyvalue/frontend/LexicalToolKit.h>`

Classes

- class Failure
  This `class` is thrown when LexicalToolKit::convert() fails.
- struct Helper
  Helper `class` of LexicalToolKit.

Public Types

- enum IO { **none** = 0x0, **input** = 0x1, **output** = 0x2, **both** = 0x3 }
  Direction of conversion.

Public Member Functions

- LexicalToolKit ()
  Default constructor.
- template<typename From , typename To >
  void set (IO io, typename Helper< From, To >::Type_ converter=util::Lexical)
  Sets a converter function pointer.
- template<typename From , typename To >
  bool isEnabled (IO io) const
  Checks if a specific conversion is enabled.
- template<typename To , typename From >
  To convert (From input) const
  Performs a conversion.

Private Attributes

- char endInitList

10.51.1 Detailed Description

Manager of all lexical converters needed for front-end input/output. This `class` stores pointers to several lexical conversion functions from/to different types. Possible conversions are:

- `bool` <--> `double`; (*)
- `bool` <--> `string`; (*)
- `double` <--> `ptime`;
- `double` <--> `string`; (*)
- `ptime` <--> `string`.

Those pointers can be changed to customized converters which must have the following signature:

To `function` (*From*)

where *From* and *To* are the input and output types, resp.

Some conversions might fail (*e.g.* from `string` "foo" to any other type) in which case the converter should throw an exception.

Conversions marked with (*) have a quite natural meaning (*e.g* from `string` "True" to `bool true`). For such conversions this `class` provides default converter (see util::Lexical) that can (if the user wishes so), be chosen (through set()) as the official converter.

10.51.2 Member Enumeration Documentation

10.51.2.1 enum IO

Direction of conversion.

See documentation of set();

10.51.3 Constructor & Destructor Documentation

10.51.3.1 LexicalToolKit ( )

Default constructor.

By default, all conversion function pointers are null.

10.51.4 Member Function Documentation

10.51.4.1 void set (IO *io*, typename Helper< From, To >::Type_ *converter* = ***util::Lexical***)

Sets a converter function pointer.

Parameters *From* and *To* specify, resp., the input and output types of the conversion function pointer to be changed.

A lexical conversion may be need for KeyValue's input or output. The parameter *io* defines if the corresponding conversion is set for input, output or both.

This function takes a pointer, *converter*, to a conversion function. If no pointer is provided, then the default converter (see util::Lexical) is used. If a null pointer is passed, then the corresponding conversion is disabled.

Parameters

From : (`template` parameter) The input type;
To : (`template` parameter) The output type;
io : Direction the conversion is set to;
converter : Pointer to the conversion function.

10.51.4.2 bool isEnabled (IO *io*) const

Checks if a specific conversion is enabled.

This `template` function checks if the *From* to *To* conversion is enabled in the *io* direction.

Parameters

From : (`template` parameter) The input type;
To : (`template` parameter) The output type;
io : Conversion direction.

Returns

This method returns `true` if the conversion is enabled. Otherwise it returns `false`.

Exceptions

LogicError : (Debug build only) If *io* is invalid.

10.51.4.3 To convert (From *input*) const

Performs a conversion.

Parameters *From* and *To* specify, resp., the input and output types of the required conversion. Notice that, in general, compilers are able to deduce *From* type and, hence it can be omitted. Therefore a typical use for this method looks like

```
LexicalToolKit lt;
double x = 3.141592654;
string s = lt.convert<string>(x);
```

Parameters

From : (`template` parameter) The input type;
To : (`template` parameter) The output type;
input : Value to be converted.

Returns

The conversion result.

Exceptions

LogicError : (Debug build only) If the required conversion is disabled.
Failure : If required conversion fails.

10.51.5 Member Data Documentation

10.51.5.1 char endInitList [private]

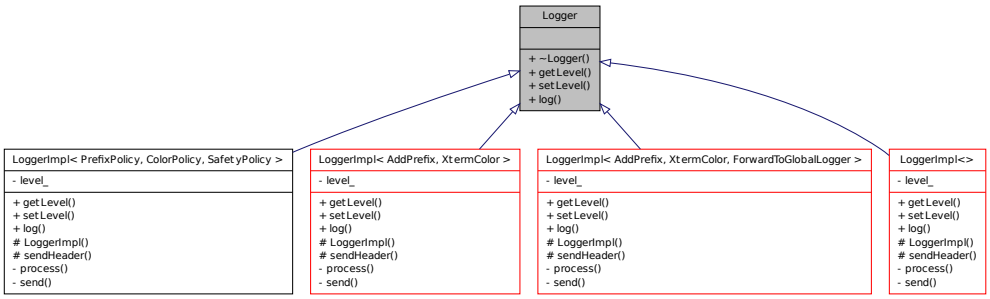
Using the X Macro technique in the constructor initialization list will produce a list of members ending in a comma. It cannot be the end of the list, otherwise, together with the following constructor's body open brace will get a syntax error. We need at least one extra member following by an space (not comma).

10.52 Logger Class Reference

Protocol `class` which defines the interface for all types of logger.

`#include <keyvalue/sys/logger/Logger.h>`

Inheritance diagram for Logger:



Public Types

- enum **Device** { **Console**, **File**, **Standard** }

Public Member Functions

- virtual unsigned int getLevel () const =0

  Gets Logger's current level.

- virtual void setLevel (unsigned int level)=0

  Sets Logger's current level.

- virtual bool log (const Message &message)=0

  Logs a Message.

10.52.1 Detailed Description

Protocol `class` which defines the interface for all types of logger. Each Logger and each Message (except those of type Error) has a verbosity level. If a Logger of level m receives a Message of level n with $m > n$,

then the Message is processed by the Logger. Otherwise it is ignored. Therefore, a 0-level Logger ignores every Message.

On the other hand, a Message of type Error is never ignored.

10.52.2 Member Function Documentation

10.52.2.1 virtual unsigned int getLevel ( ) const [pure virtual]

Gets Logger's current level.

Returns

The level.

Implemented in LoggerImpl< PrefixPolicy, ColorPolicy, SafetyPolicy >.

10.52.2.2 virtual void setLevel (unsigned int *level*) [pure virtual]

Sets Logger's current level.

Parameters

level : New level.

Implemented in LoggerImpl< PrefixPolicy, ColorPolicy, SafetyPolicy >.

10.52.2.3 virtual bool log (const Message & *message*) [pure virtual]

Logs a Message.

This method may fail if the implemented Logger cannot process the message (e.g. a console Logger that has no longer a console window).

Parameters

message : Message to be logged.

Returns

This method returns `true` if it is successful.

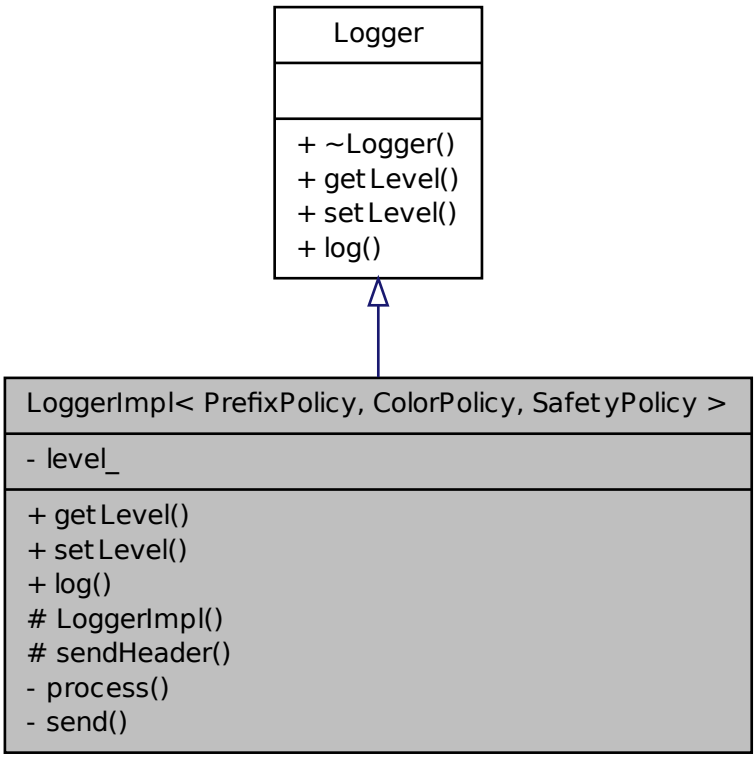
Implemented in LoggerImpl< PrefixPolicy, ColorPolicy, SafetyPolicy >.

10.53 LoggerImpl< PrefixPolicy, ColorPolicy, SafetyPolicy > Class Template Reference

Abstract `class` which implements main methods of concrete Loggers.

`#include <keyvalue/sys/logger/LoggerImpl.h>`

Inheritance diagram for LoggerImpl< PrefixPolicy, ColorPolicy, SafetyPolicy >:



Public Types

- typedef LoggerImpl< PrefixPolicy, ColorPolicy, SafetyPolicy > **LoggerImpl_**
- enum **Device** { **Console**, **File**, **Standard** }

Public Member Functions

- unsigned int getLevel () const

  Gets Logger's current level.

- void setLevel (unsigned int level)

  Sets Logger's current level.

- bool log (const Message &message)

  Logs a Message.

Protected Member Functions

- **LoggerImpl** (unsigned int level)
- bool sendHeader ()

  Sends the header message.

Private Member Functions

- bool process (const Message &message)

  Processes a Message and send its text content to the Logger's underlying device.

- virtual bool send (const string &message)=0

  Sends a raw `string` message to the Logger's underlying device.

Private Attributes

- unsigned int **level_**

10.53.1 Detailed Description

template<typename PrefixPolicy = AddPrefix, typename ColorPolicy = IgnoreColor, typename SafetyPolicy = ForwardToGlobalLogger> class keyvalue::logger::LoggerImpl< PrefixPolicy, ColorPolicy, SafetyPolicy >

Abstract `class` which implements main methods of concrete Loggers. This is a `template class` whose `template` parameters are policy `class`es that define how the Message has to be processed.

Parameters

PrefixPolicy : Defines how to process Message prefixes;

ColorPolicy : Defines how to process Message colors;

SafetyPolicy : Defines what to do when the Logger fails.

10.53.2 Member Function Documentation

10.53.2.1 unsigned int getLevel ( ) const [virtual]

Gets Logger's current level.

Returns

The level.

Implements Logger.

10.53.2.2 void setLevel (unsigned int *level*) [virtual]

Sets Logger's current level.

Parameters

level : New level.

Implements Logger.

10.53.2.3 bool log (const Message & *message*) [virtual]

Logs a Message.

This method may fail if the implemented Logger cannot process the message (e.g. a console Logger that has no longer a console window).

Parameters

message : Message to be logged.

Returns

This method returns `true` if it is successful.

Implements Logger.

10.53.2.4 bool sendHeader ( ) [protected]

Sends the header message.

This method sends (throw send()) an opening header message to the underlying device of the Logger.

Returns

This method returns what print() does.

10.53.2.5 bool process (const Message & *message*) [private]

Processes a Message and send its text content to the Logger's underlying device.

This pure virtual method has to be implemented by each LoggerImpl. After processing the received Message this method should call send() to send the text content of the Message to the device of the Logger.

This method may fail if either any policy or the send() does so.

Parameters

message : Message to be processed.

Returns

This method returns `false` if it fails. Otherwise, it returns `true`.

10.53.2.6 virtual bool send (const string & *message*) [private, pure virtual]

Sends a raw `string` message to the Logger's underlying device.

Parameters

message : Message to be sent.

Returns

This method returns `false` if it fails. Otherwise, it returns `true`.

Implemented in FileLogger, WindowsConsole, and XtermConsole.

10.54 Logic Class Reference

MessageImpl instantiation used for logic error messages.

`#include <keyvalue/sys/message/MessageImpl.h>`

10.54.1 Detailed Description

MessageImpl instantiation used for logic error messages. `typedef MessageImpl<0> Logic;`

10.55 LogicError Class Reference

Base `class` for all *KeyValue* exceptions occurring at development time.

`#include <keyvalue/sys/exception/Exception.h>`

10.55.1 Detailed Description

Base `class` for all *KeyValue* exceptions occurring at development time. When a LogicError (or any of its derived `classes`) is thrown, it means there is a BUG.

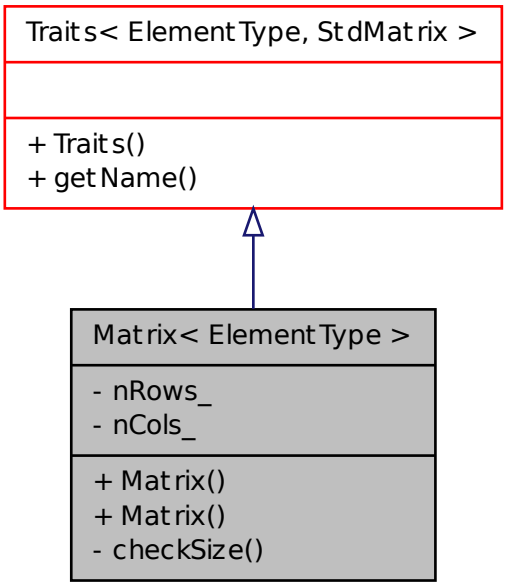
`typedef ExceptionImpl<std::logic_error, Logic> LogicError;`

10.56 Matrix< ElementType > Class Template Reference

Generic key whose converter type is StdMatrix.

`#include <keyvalue/key/generic/Matrix.h>`

Inheritance diagram for Matrix< ElementType >:



Public Types

- enum
- typedef Traits< ElementType, StdMatrix, Default > **Traits_**
- typedef StdMatrix< typename Default< ElementType >::OutputType_ > **ConverterType_**
- typedef ConverterType_::InputType_ **InputType_**
- typedef ConverterType_::OutputType_ **OutputType_**

Public Member Functions

- Matrix (const string &name)

  Constructs the key and sets its name.

- Matrix (const string &name, size_t nRows, size_t nCols_)

  Constructs the key and sets its name and the expected dimension of the StdMatrix content.

- string getName () const

  Gets Key's name.

- string getName () const

  Gets name.

- void setName (const string &name)
  Sets name.

Private Member Functions

- void checkSize (size_t nRows, size_t nCols) const
  Checks if given and expected dimensions match.

Private Attributes

- size_t **nRows_**
- size_t **nCols_**

10.56.1 Detailed Description

template<typename ElementType> class keyvalue::key::Matrix< ElementType >

Generic key whose converter type is StdMatrix.

Parameters

ElementType : (`template` parameter) Matrix element type.

10.56.2 Constructor & Destructor Documentation

10.56.2.1 Matrix (const string & *name*) `[explicit]`

Constructs the key and sets its name.

10.56.2.2 Matrix (const string & *name*, size_t *nRows*, size_t *nCols_*)

Constructs the key and sets its name and the expected dimension of the StdMatrix content.

Parameters

name : The key name;
nRows : Expected number of rows;
nCols : Expected number of columns.

10.56.3 Member Function Documentation

10.56.3.1 void checkSize (size_t *nRows*, size_t *nCols*) const `[private]`

Checks if given and expected dimensions match.

Parameters

nRows : Expected number of rows;

nCols : Expected number of columns.

RuntimeError : If given and expected dimension do not match.

10.56.3.2 string getName ( ) const [inherited]

Gets Key's name.

Returns

The Key's name.

10.56.3.3 string getName ( ) const [inherited]

Gets name.

Returns

The name.

10.56.3.4 void setName (const string & *name*) [inherited]

Sets name.

Parameters

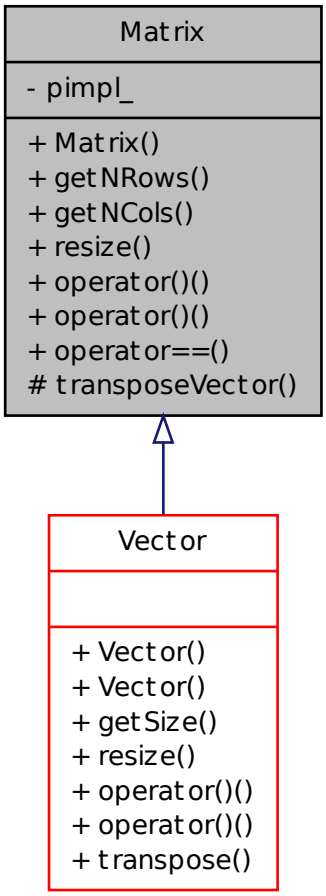
name : The name.

10.57 Matrix Class Reference

Matrix of Variants.

`#include <keyvalue/value/Matrix.h>`

Inheritance diagram for Matrix:



Public Member Functions

- Matrix (size_t nRows=1, size_t nCols=1)

  Constructs a Matrix with a given dimension.

- size_t getNRows () const

  Gets number of rows.

- size_t getNCols () const

  Gets number of columns.

- void resize (size_t nRows, size_t nCols)

  Resizes the matrix.

- const Variant & operator() (size_t i, size_t j) const

  Provides access to the element in the i-th row and j-th column.

- Variant & operator() (size_t i, size_t j)

  Non `const` *operator()().*

- bool operator== (const Matrix &rhs) const

  Comparison operator.

Protected Member Functions

- void transposeVector ()

  Transposes the matrix if one of its dimensions is 1.

Private Attributes

- boost::intrusive_ptr< Impl > **pimpl_**

Friends

- void intrusive_ptr_add_ref (Matrix::Impl *pimpl)
- void intrusive_ptr_release (Matrix::Impl *pimpl)

10.57.1 Detailed Description

Matrix of `Variant`s. Why yet another matrix `class`? The reason is the following. Key-value pattern recognition (see pattern::Pattern) is simplified if vector is publicly derived from matrix (a vector is either a *m* x 1 or a 1 x *n* matrix) and single is publicly derived from vector (a single is a one-dimensional vector).

Furthermore, this `class` and all its derived `class`es implement reference semantics when copying. That is, when a Matrix *M* is copied to a Matrix *N*, then changing *N* also changes *M* and reciprocally.

The reference semantics is implemented through the PImpl idiom which also make this `class` cheap to copy.

10.57.2 Constructor & Destructor Documentation

10.57.2.1 Matrix (size_t *nRows* = *1*, size_t *nCols* = *1*) `[explicit]`

Constructs a Matrix with a given dimension.

If the number of rows or the number of columns is zero, then both are changed to 1. Hence, trying to build a 0 x *n* or *m* x 0 matrix ends up building a 1 x 1 matrix.

Parameters

nRows : number of rows;

nCols : number of columns.

10.57.3 Member Function Documentation

10.57.3.1 size_t getNRows ( ) const

Gets number of rows.

Returns

The number of rows.

10.57.3.2 size_t getNCols ( ) const

Gets number of columns.

Returns

The number of columns.

10.57.3.3 void resize (size_t *nRows*, size_t *nCols*)

Resizes the matrix.

The size of a matrix (*i.e.*, number of rows times number of columns) can not increase. Debug build check for this and an exception is thrown when the test fails. Release build has undefined behavior in this case.

The contents of a matrix after resizing is undefined.

Parameters

nRows : number of rows;
nCols : number of columns.

Exceptions

LogicError : (Debug build only) If the new size is greater than the original one.

10.57.3.4 const Variant& operator() (size_t *i*, size_t *j*) const

Provides access to the element in the *i-th* row and *j-th* column.

Debug build perform bound checks and throws an exception on failure. Release build has undefined behavior in that case.

Parameters

i : Row index;
j : Column index.

Returns

A `const` reference to *(i, j)*-th element.

Exceptions

LogicError : (Debug build only) If *i* is not smaller than the number of rows or *j* is not smaller than the number of columns.

10.57.3.5 Variant& operator() (size_t *i*, size_t *j*)

Non `const` operator()().

10.57.3.6 bool operator== (const Matrix & *rhs*) const

Comparison operator.

Recall that Matrix has reference semantics. Consider the code:

```
Matrix m1(2,2);
m1(0,0) = m1(0,1) = m1(1,0) = m1(1,1) = 1.0;
Matrix m2(2,2);
m2(0,0) = m2(0,1) = m2(1,0) = m2(1,1) = 1.0;
Matrix m3(m1);
```

Then `m1 == m2` is `false` and `m1 == m3` is `true`.

Parameters

rhs : The Value to be compared against.

Returns

This method returns `true` if ∗this and *rhs* refer to the same data. Otherwise it returns `false`.

10.57.3.7 void transposeVector ( ) [protected]

Transposes the matrix if one of its dimensions is 1.

In-place matrix transposition is a complicated stuff. See

http://en.wikipedia.org/wiki/In-place_matrix_transposition

However, since the storage is linear, the memory layouts of n x 1 and 1 x n matrix, are the same. Hence, we only need to swap dimensions. This method is intent to perform only this simple transposition and it is protected to be called by Vector which derives from this class.

Debug build checks if either of the dimensions is 1 and throws an exception if the test fails. Release build has undefined behaviour in that circumstance.

Exceptions

LogicError : (Debug build only) If none of dimensions is 1.

10.57.4 Friends And Related Function Documentation

10.57.4.1 void intrusive_ptr_add_ref (Matrix::Impl ∗ *pimpl*) [friend]

headerfile Matrix.h "keyvalue/value/Matrix.h"

brief Increment Matrix reference counting.

See documentation of intrusive_ptr in the Smart Ptr (Boost) library.

param Pointer to Matrix::Impl storing the counter.

10.57.4.2 void intrusive_ptr_release (Matrix::Impl * *pimpl*) [friend]

headerfile Matrix.h "keyvalue/value/Matrix.h"

brief Decrement Matrix reference counting.

See documentation of intrusive_ptr in the Smart Ptr (Boost) library.

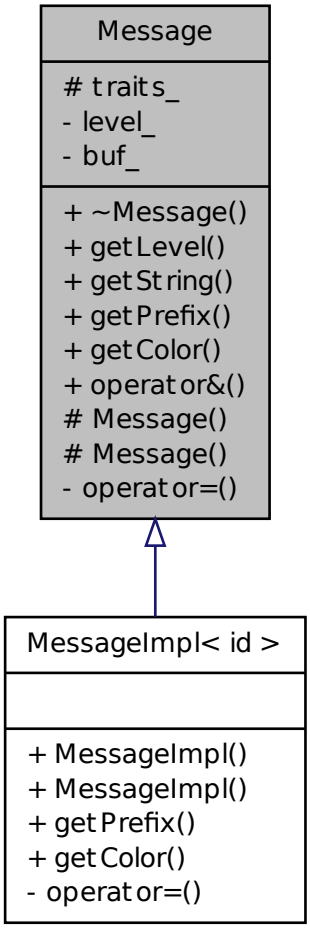
param Pointer to Matrix::Impl storing the counter.

10.58 Message Class Reference

Abstract `class` which defines the interface for all types of message.

`#include <keyvalue/sys/message/Message.h>`

Inheritance diagram for Message:



Public Member Functions

- unsigned int getLevel () const

  Gets Message's level.

- string getString () const

  Gets Message's text.

- virtual const string & getPrefix () const =0

  Gets Message's prefix.

- virtual logger::Color getColor () const =0

  Gets Message's color.

- template<typename DataType >
  Message & operator& (const DataType &data)

  Appends data to the end of message.

Protected Member Functions

- Message (unsigned int level)

  Constructs a Message with a given level.

- **Message** (const Message &orig)

Static Protected Attributes

- static const Traits_ **traits_** [ ]

Private Member Functions

- Message & **operator=** (const Message &orig)

Private Attributes

- unsigned int **level_**
- std::ostringstream **buf_**

10.58.1 Detailed Description

Abstract `class` which defines the interface for all types of message. Each Message has a level that defines if it will be ignored or not by the Logger which receives it. (For more details, see Logger documentation).

10.58.2 Constructor & Destructor Documentation

10.58.2.1 Message (unsigned int *level*) [explicit, protected]

Constructs a Message with a given level.

Parameters

level : Message's level.

10.58.3 Member Function Documentation

10.58.3.1 unsigned int getLevel ( ) const

Gets Message's level.

Returns

The level.

10.58.3.2 string getString ( ) const

Gets Message's text.

Returns

The text.

10.58.3.3 virtual const string& getPrefix ( ) const [pure virtual]

Gets Message's prefix.

Returns

The prefix.

Implemented in MessageImpl< id >.

10.58.3.4 virtual logger::Color getColor ( ) const [pure virtual]

Gets Message's color.

Returns

The color.

Implemented in MessageImpl< id >.

10.58.3.5 Message & operator& (const DataType & *data*)

Appends data to the end of message.

This `template` method makes use of `std::ostream::operator<<()` to append data to this Message. Hence, the type of data *DataType* must be such that `std::ostream::operator<<(ostream&, const DataType&)` is well defined.

A typical use of this method is

```
Info info(1);  // Info is derived from Message
size_t i;
std::vector<double> x;
// ...
info & "x[" & i & "] = " & x[i];
```

Warning: For some reason, one cannot append `std::endl` to Message, despite the fact that a standard `ostream` can accept it!

Remark: It would be more natural using operator<< rather than operator&. However, the main use of operator& is related to exceptions (see exception::ExceptionImpl documentation). Indeed, there exist ExceptionImpl::operator&() which just redirects its call to this operator& here. In the past, *KeyValue* exceptions were derived from boost::exception which defines operator<< for other purposes. Therefore, operator& was prefered rather than operator<< to avoid conflict with boost. Be aware that, in the near future, if it is proven convenient, one might change operator& to operator<< again.

Parameters

DataType : (`template` parameter) Type of data;

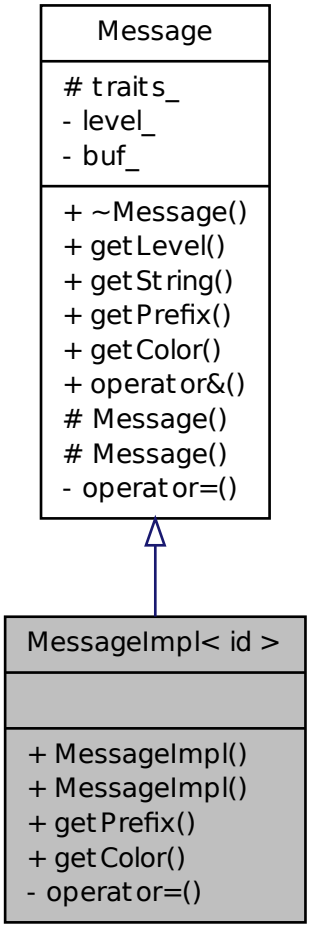
data : the data to be appended.

10.59 MessageImpl< id > Class Template Reference

Implementation of `class` Message.

`#include <keyvalue/sys/message/MessageImpl.h>`

Inheritance diagram for MessageImpl< id >:



Public Member Functions

- MessageImpl ()
- **MessageImpl** (unsigned int level)
- const string & getPrefix () const

  Gets prefix associated this type of Message.

- logger::Color getColor () const

  Gets the color associated this type of Message.

- unsigned int getLevel () const

  Gets Message's level.

- string getString () const

  Gets Message's text.

- template<typename DataType >
  Message & operator& (const DataType &data)

  Appends data to the end of message.

Static Protected Attributes

- static const Traits_ **traits_** [ ]

Private Member Functions

- Message & **operator=** (const MessageImpl &orig)

10.59.1 Detailed Description

template<unsigned int id> class keyvalue::MessageImpl< id >

Implementation of `class` Message. This `template class` provides default implementations for Message methods and extends its interface.

In this `template class`, parameter `unsigned int` *id* serves to distinguish specializations. Clients, rather than directly instantiate this `template` by providing *id*, will use provided `typedefs` for the following types of Messages:

- Error;
- Info;
- Warning;
- Report;
- Debug.

Parameters

id : (`template` parameter) Used to distinguish different specializations.

10.59.2 Constructor & Destructor Documentation

10.59.2.1 MessageImpl ( ) `[inline]`

ATTENTION: If you get a compiler error here about

'boostSTATIC_ASSERTION_FAILURE<x>'

this means that you have illegally construct an Info, Warning, Report, or Debug without providing its level.

10.59.3 Member Function Documentation

10.59.3.1 const string& getPrefix ( ) const [virtual]

Gets prefix associated this type of Message.

Returns

The prefix.

Implements Message.

10.59.3.2 logger::Color getColor ( ) const [virtual]

Gets the color associated this type of Message.

Returns

The prefix.

Implements Message.

10.59.3.3 unsigned int getLevel ( ) const [inherited]

Gets Message's level.

Returns

The level.

10.59.3.4 string getString ( ) const [inherited]

Gets Message's text.

Returns

The text.

10.59.3.5 Message & operator& (const DataType & *data*) [inherited]

Appends data to the end of message.

This `template` method makes use of `std::ostream::operator<<()` to append data to this Message. Hence, the type of data *DataType* must be such that `std::ostream::operator<<(ostream&, const DataType&)` is well defined.

A typical use of this method is

```
Info info(1);  // Info is derived from Message
size_t i;
std::vector<double> x;
// ...
info & "x[" & i & "] = " & x[i];
```

Warning: For some reason, one cannot append `std::endl` to Message, despite the fact that a standard `ostream` can accept it!

Remark: It would be more natural using operator<< rather than operator&. However, the main use of operator& is related to exceptions (see exception::ExceptionImpl documentation). Indeed, there exist ExceptionImpl::operator&() which just redirects its call to this operator& here. In the past, *KeyValue* exceptions were derived from boost::exception which defines operator<< for other purposes. Therefore, operator& was prefered rather than operator<< to avoid conflict with boost. Be aware that, in the near future, if it is proven convenient, one might change operator& to operator<< again.

Parameters

DataType : (`template` parameter) Type of data;

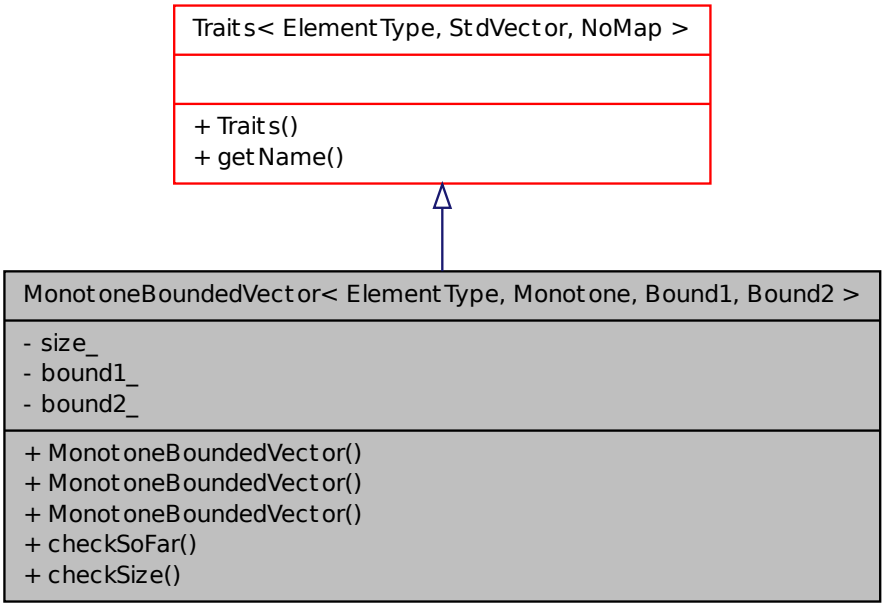
data : the data to be appended.

10.60 MonotoneBoundedVector< ElementType, Monotone, Bound1, Bound2 > Class Template Reference

Key for monotone bounded vectors.

`#include <MonotoneBoundedVector.h>`

Inheritance diagram for MonotoneBoundedVector< ElementType, Monotone, Bound1, Bound2 >:



Public Types

- enum
- typedef Traits< ElementType, StdVector, NoMap > **Traits_**
- typedef StdVector< typename NoMap< ElementType >::OutputType_ > **ConverterType_**
- typedef ConverterType_::InputType_ **InputType_**
- typedef ConverterType_::OutputType_ **OutputType_**
- typedef ElementType **OutputType_**

Public Member Functions

- MonotoneBoundedVector (const string &name, size_t size=0)

  Constructs a MonotoneBoundedVector given its name and expected size.

- MonotoneBoundedVector (const string &name, const ElementType &bound1, size_t size=0)

  Constructs a MonotoneBoundedVector given its name, a bound and expected size.

- MonotoneBoundedVector (const string &name, const ElementType &bound1, const ElementType &bound2, size_t size=0)

  Constructs a MonotoneBoundedVector given its name, two bounds and expected size.

- void checkSoFar (const StdVector< ElementType > &container) const

  Checks for monotonicity and boundness of the values.

- void checkSize (size_t size) const

  Checks for the size of the vector of values.

- string getName () const

  Gets Key's name.

- string getName () const

  Gets name.

- void setName (const string &name)

  Sets name.

- ElementType map (const value::Variant &variant) const

  Performs the mapping.

Private Attributes

- const size_t **size_**
- const Bound1< ElementType > **bound1_**
- const Bound2< ElementType > **bound2_**

10.60.1 Detailed Description

template<typename ElementType, typename Monotone, template< typename > class Bound1 = NoBound, template< typename > class Bound2 = NoBound> class keyvalue::key::MonotoneBoundedVector< ElementType, Monotone, Bound1, Bound2 >

Key for monotone bounded vectors. This `template` is used for keys whose converter type is a StdVector holding a `std::vector<`*ElementType*`>` *V* (of size *n*) with bounded and ordered elements.

There are five types of monotonicity modeled by monotone `class`es:

- NonMonotone (which means the no monotonicity);
- Increasing;
- StrictlyIncreasing;
- Decreasing; and
- StrictlyDecreasing.

Given a monotone `class` *Monotone* and two bound `template class`es *Bound1* and *Bound2* (See Bounded for more information on bound `template class`es), *V* will be accepted only if

- *n* = 1 or *V*[i-1] and *V*[i] are in right order as defined by *Monotone*, for all *i* = 1, ..., *n* - 1;
- *bound1_*::check(*V*[i]) = `true` for all *i* = 0, ..., *n* - 1; and
- *bound2_*::check(*V*[i]) = `true` for all *i* = 0 ..., *n* - 1;

where *bound1_* and *bound2_* are instantiations (for *ElementType*) of bound `template class`es *Bound1* and *Bound2*, resp.

Additionally, at construction time, the key name and the expected size of the *V* are fixed. At the time the value for this key is requested, *V*'s size and its expected size are compared. If they do not match, then *V* is rejected.

Example 1: A key for a vector of dates in strictly increasing order:

```
MonotoneBoundedVector<ptime, StrictlyIncreasing> dates("Dates");
```

Example 2: A key for a vector of prices which accepts positive numbers.

```
MonotoneBoundedVector<double, NoBound, Geq> prices("Prices", 0.0);
```

Example 3: Same as in Example 2 but accepting only 10 elements:

```
MonotoneBoundedVector<double, NoBound, Geq> prices("Prices", 0.0, 10);
```

Example 4: A key for a vector of 5 decreasing numbers in the interval (-2,2):

```
MonotoneBoundedVector<double, Decreasing, Greater, Less> a("A", -2.0, 2.0, 5);
```

Parameters

ElementType : (`template` parameter) See description above;
Monotone : (`template` parameter) Monotone `class`;
Bound1 : (`template` parameter) 1st bound `template class`;
Bound2 : (`template` parameter) 2nd bound `template class`.

10.60.2 Constructor & Destructor Documentation

10.60.2.1 MonotoneBoundedVector (const string & *name*, size_t *size* = *0*) [explicit]

Constructs a MonotoneBoundedVector given its name and expected size.

Parameters

name : The key name;

size : Expected size. By default, *size* = 0 which means there is no expected size, i.e., any strictly positive size for the StdVector content will be accepted.

10.60.2.2 MonotoneBoundedVector (const string & *name*, const ElementType & *bound1*, size_t *size* = *0*)

Constructs a MonotoneBoundedVector given its name, a bound and expected size.

Parameters

name : The key name;

bound1 : The value to be wrapped by *Bound1*;

size : Expected size. By default, *size* = 0 which means there is no expected size, i.e., any stricktly positive size for the StdVector content will be accepted.

10.60.2.3 MonotoneBoundedVector (const string & *name*, const ElementType & *bound1*, const ElementType & *bound2*, size_t *size* = *0*)

Constructs a MonotoneBoundedVector given its name, two bounds and expected size.

Parameters

name : The key name;

bound1 : The value to be wrapped by *Bound1*;

bound2 : The value to be wrapped by *Bound2*;

size : Expected size. By default, *size* = 0 which means there is no expected size, i.e., any strictly positive size for the StdVector content will be accepted.

10.60.3 Member Function Documentation

10.60.3.1 void checkSoFar (const StdVector< ElementType > & *container*) const

Checks for monotonicity and boundness of the values.

Parameters

container : The StdVector containing the `std::vector` of to be checked.

Exceptions

RuntimeError : If test fails.

10.60.3.2 void checkSize (size_t *size*) const

Checks for the size of the vector of values.

Parameters

container : The StdVector containing the `std::vector` whose size should be checked.

Exceptions

RuntimeError : If there is no expected size or if it does not agree with real size.

10.60.3.3 string getName ( ) const [inherited]

Gets Key's name.

Returns

The Key's name.

10.60.3.4 string getName ( ) const [inherited]

Gets name.

Returns

The name.

10.60.3.5 void setName (const string & *name*) [inherited]

Sets name.

Parameters

name : The name.

10.60.3.6 ElementType map (const value::Variant & *variant*) const [inherited]

Performs the mapping.

This method retrieves the content of value::Variant and, by considering enabled lexical convertions, converts it to *Output*. (See LexicalToolKit documentation.)

Parameters

variant : A value::Variant containing the input to be mapped.

Returns

The mapping result.

10.61 NoBound< ElementType > Class Template Reference

NoBound bound `class`.

`#include <keyvalue/key/generic/bound/NoBound.h>`

Public Member Functions

- bool check (const ElementType &x) const
  Returns `true`.
- const ElementType & getBound () const
  Does not do anything.
- const char * getName () const
  Does not do anything.

10.61.1 Detailed Description

template<typename ElementType> class keyvalue::key::NoBound< ElementType >

NoBound bound `class`.

Parameters

ElementType : (`template` parameter) Ignored. Accepted types are `double`, `ptime`, `string` and `unsigned int`. (Other types will generate a link error.)

10.61.2 Member Function Documentation

10.61.2.1 bool check (const ElementType & *x*) const

Returns `true`.

Parameters

x : Ignored;

Returns

`true`.

10.61.2.2 const ElementType& getBound ( ) const

Does not do anything.

This method must not be called. Otherwise we a LogicError will be thrown. It is here just to comply with bound `class`es specifications.

Returns

N/A.

Exceptions

LogicError.

10.61.2.3 const char* getName ( ) const

Does not do anything.

This method must not be called. Otherwise we a LogicError will be thrown. It is here just to comply with bound `classes` specifications.

Returns

N/A.

Exceptions

LogicError.

10.62 NoMap< OutputType > Class Template Reference

Identity map (aka, NoMap).

`#include <keyvalue/key/map/NoMap.h>`

Public Types

- typedef OutputType **OutputType_**

Public Member Functions

- OutputType map (const value::Variant &variant) const

  Performs the mapping.

Private Types

- typedef boost::is_same< OutputType, bool >::type **IsBool_**
- typedef boost::is_same< OutputType, double >::type **IsDouble_**
- typedef boost::is_same< OutputType, ptime >::type **IsPtime_**
- typedef boost::is_same< OutputType, string >::type **IsString_**
- typedef boost::is_same< OutputType, unsigned int >::type **IsUInt_**

Private Member Functions

- virtual string getName () const =0

  Gets Key's name.

- **BOOST_STATIC_ASSERT** (IsBool_::value||IsDouble_::value||IsString_::value||IsPtime_::value||IsUInt_::value)

10.62.1 Detailed Description

template<typename OutputType> class keyvalue::key::NoMap< OutputType >

Identity map (aka, NoMap). The input value is mapped to itself. For example, the value associated to key "Price" is supposed to be a `double` which must be passed on, without any kind of mapping or, in other terms, mapped by the identity function.

Remark: Only instantiations for *Output* equal to `bool`, `double`, `ptime`, `string` or `unsigned int` are provided.

Parameters

OutputType : (`template` parameter) Output type.

Return values

OutputType_ : Same as *OutputType*.

10.62.2 Member Function Documentation

10.62.2.1 OutputType map (const value::Variant & *variant*) const

Performs the mapping.

This method retrieves the content of value::Variant and, by considering enabled lexical convertions, converts it to *Output*. (See LexicalToolKit documentation.)

Parameters

variant : A value::Variant containing the input to be mapped.

Returns

The mapping result.

10.62.2.2 virtual string getName ( ) const [private, pure virtual]

Gets Key's name.

Returns

The Key's name.

10.63 NonMonotone Class Reference

NonMonotone `class`.

`#include <keyvalue/key/generic/monotone/NonMonotone.h>`

Static Public Member Functions

- template<typename ElementType >
  static bool check (const ElementType &x, const ElementType &y)

  Returns `true`.

- static const char * getName ()

  Does not do anything.

10.63.1 Detailed Description

NonMonotone `class`. Used for keys with non monotone values.

10.63.2 Member Function Documentation

10.63.2.1 static bool check (const ElementType & *x*, const ElementType & *y*) [static]

Returns `true`.

Parameters

x : Ignored;

y : Ignored.

ElementType : (`template` parameter) Type of *x*. Accepted types are `double`, `ptime`, `string` and `unsigned int`. (Other types will generate a link error.)

Returns

`true`.

10.63.2.2 static const char* getName ( ) [static]

Does not do anything.

This method must not be called. Otherwise we a LogicError will be thrown. It is here just to comply with comparison `class`es specifications.

Returns

N/A.

Exceptions

LogicError.

10.64 Repository::NotFound Class Reference

Exception thrown by get() when a DataSet is not found.

`#include <Repository.h>`

Inherits exception::RuntimeError.

Public Member Functions

- **NotFound** (string name)
- const string & **getName** () const

Private Attributes

- const string **name_**

10.64.1 Detailed Description

Exception thrown by get() when a DataSet is not found.

10.65 Nothing Class Reference

Empty `class` to represent empty data.

`#include <keyvalue/value/Nothing.h>`

10.65.1 Detailed Description

Empty `class` to represent empty data.

10.66 NullDeleter Class Reference

A `shared_ptr` deleter that does not do anything.

`#include <keyvalue/util/NullDeleter.h>`

Public Member Functions

- void **operator()** (void const ∗) const

Private Member Functions

- NullDeleter & **operator=** (const NullDeleter &original)

10.66.1 Detailed Description

A `shared_ptr` deleter that does not do anything.

10.67 ObjectMap< ObjectType > Class Template Reference

Maps names to objects.

`#include <keyvalue/key/map/ObjectMap.h>`

Public Types

- typedef shared_ptr< ObjectType > **OutputType_**

Public Member Functions

- shared_ptr< ObjectType > map (const value::Variant &variant) const

  Performs the mapping.

Private Member Functions

- virtual string getName () const =0

  Gets Key's name.

10.67.1 Detailed Description

template<typename ObjectType> class keyvalue::key::ObjectMap< ObjectType >

Maps names to objects. A name (`string`) is mapped to (a pointer to) an object with that name. The mapping is performed by the Repository.

When the object's name is not found in the Repository, an attempt to build one from the given input (which, in fact, is a value::Variant) is done.

This map policy is automatically chosen by Traits when *OutputType* is not considered a basic type (see Default documentation). Hence, most users can ignore this map.

Parameters

ObjectType : (`template` parameter) The type of output object.

Return values

OutputType_ : Same as `shared_ptr<ObjectType>`.

10.67.2 Member Function Documentation

10.67.2.1 shared_ptr< ObjectType > map (const value::Variant & *variant*) const

Performs the mapping.

Parameters

variant : A value::Variant containing the object's name.

Returns

The mapping result.

10.67.2.2 virtual string getName ( ) const [private, pure virtual]

Gets Key's name.

Returns

The Key's name.

10.68 ObjectPtr Class Reference

Pointer to objects.

`#include <keyvalue/value/Result.h>`

10.68.1 Detailed Description

Pointer to objects. Builders return pointers to objects which are converted to pointer to `void`.

```
typedef shared_ptr<void> ObjectPtr;
```

10.69 Parent< T > Struct Template Reference

Primary Parent `template` meta-function.

`#include <Parent.h>`

10.69.1 Detailed Description

template<typename T> struct keyvalue::value::Parent< T >

Primary Parent `template` meta-function. The types in `namespace` value follow an hierarchy NOT built through inheritance. Before explaining how this hierarchy is implemented, we will consider the reason for its existence by an explanatory example.

Consider a Calculator whose result is a `double` x. Since all Calculators are expected to return Values, x must be converted before being returned. Recall that Value is a container for Matrix, Vector or Single. Obviously, Single is the right choice to carry x. However, Single is a container for Variants. Hence, firstly x must be converted to Variant. We conclude that x must to pass trough a series of conversions provided by copy-constructors. Therefore, the Calculator might return x as follows:

```
return Value(Single(Variant(x)));
```

It would be more convenient if these conversions were automatic. That is the reason for the hierarchy of types. Any type might be copy-constructed or assigned from any other type in a lower level of the hierarchy. In the example, returning x directly from the Calculator would imply in all the conversions up to Value.

Let T be a type (a basic one or one created by a Builder) and N be of type `size_t`. The hierarchy has Result on the top level and is as follows:

```
Result
`- Result::DataType_
```

```
+- ObjectPtr
|   `- shared_ptr<T>
|      `- T*
`- Value
   `- Value::DataType_
      +- Single
      |   `- Variant
      |      `- Variant::DataType_
      |         +- Nothing
      |         +- bool
      |         +- double
      |         |   +- int
      |         |   +- unsigned int
      |         |   `- long unsigned int
      |         +- string
      |         |   +- char[]
      |         |   +- char[N]
      |         |   +- char*
      |         |   +- const char[]
      |         |   +- const char[N]
      |         |   `- const char*
      |         `- ptime
      +- Vector
      |   `- vector<T>
      `- Matrix
```

Remarks:

- Do not bother with types Result::DataType_, Value::DataType_ and Variant::DataType_. They act as intermediary steps before conversions to their respective owner `classes`. You are not intent to directly create objects of those types.
- The presence of *T*∗ just below `shared_ptr<`*T*`>` provides an extra level of safety. Indeed, if a raw pointer to *T* is assigned or copy-constructed to a Result or any other type between them, then it will be immediately wrapped by a `shared_ptr<`*T*`>`.

Finally, Parent a `template` meta-function which returns the parent of a type according to the value hierarchy. Each for each type in the hierarchy (excluding Result) there should be a specialization.

Parameters

T : The type whose parent should be returned.

Return values

Parent*<*T*>*::Type_ : *T*'s parent type.

10.70 PartialMap< OutputType > Class Template Reference

Partially maps names to constants.

`#include <keyvalue/key/map/PartialMap.h>`

Public Types

- typedef OutputType **OutputType_**

Public Member Functions

- OutputType map (const value::Variant &variant) const
  Performs the mapping.
- virtual OutputType get (const string &name) const =0
  Performs the mapping from `string` to OutputType.

Private Types

- typedef boost::is_same< OutputType, double >::type **IsDouble_**
- typedef boost::is_same< OutputType, ptime >::type **IsPtime_**
- typedef boost::is_same< OutputType, unsigned int >::type **IsUInt_**

Private Member Functions

- virtual string getName () const =0
  Gets Key's name.
- **BOOST_STATIC_ASSERT** (IsDouble_::value||IsPtime_::value||IsUInt_::value)

10.70.1 Detailed Description

template<typename OutputType> class keyvalue::key::PartialMap< OutputType >

Partially maps names to constants. Use this map, when a special `double`, `ptime` or `unsigned int` value has a name.

For instance, when asked for the number of edges of a regular polygon, the answer must be an `unsigned int` greater than 2. For some special values (not all) there correspond a polygon name (*e.g* 'Triangle' for 3 or 'Square' for 4). There is no special name for a regular polygon with 1111 edges.

Traits derived `class`es which uses this map must implement a method to perform the mapping. This method has the following signature

```
Output get(const string& name) const;
```

Parameters

OutputType : (`template` parameter) Output type.

Return values

OutputType_ : Same as *OutputType*.

10.70.2 Member Function Documentation

10.70.2.1 OutputType map (const value::Variant & *variant*) const

Performs the mapping.

This method converts the input from value::Variant to `string` and calls get().

Parameters

variant : A value::Variant containing the input `string` to be mapped.

Returns

The mapping result.

10.70.2.2 virtual OutputType get (const string & *name*) const [pure virtual]

Performs the mapping from `string` to *OutputType*.

Must be implemented by real keys which use this map.

Parameters

name : The `string` to be mapped.

Returns

The mapping result.

10.70.2.3 virtual string getName ( ) const [private, pure virtual]

Gets Key's name.

Returns

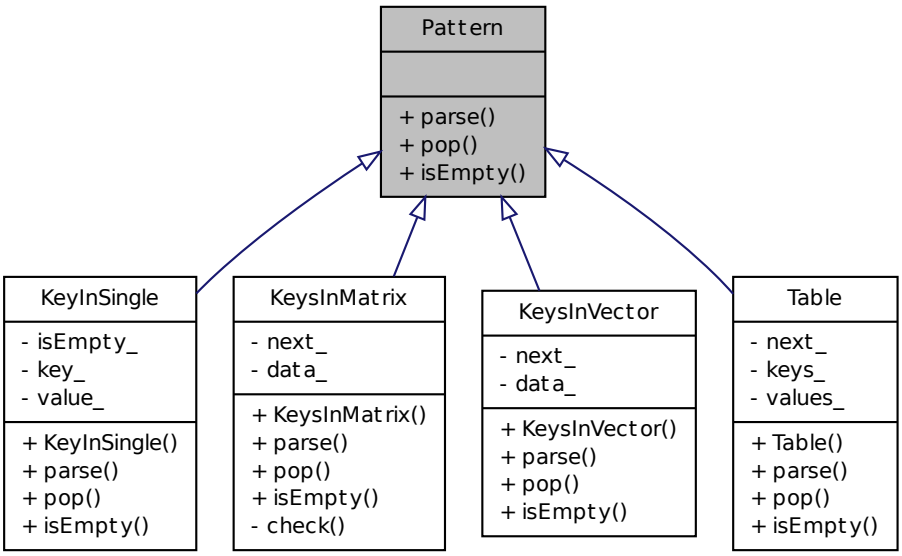
The Key's name.

10.71 Pattern Class Reference

Protocol `class` which defines the interface for pattern recognition `class`es.

`#include <keyvalue/pattern/Pattern.h>`

Inheritance diagram for Pattern:



Classes

- class QueueRaii

  This `class` implements RAII for a frontend::Queue.

Public Member Functions

- virtual bool parse (frontend::Queue &queue)=0

  Parses the beginning of a frontend::Queue.

- virtual std::pair< string, value::Value > pop ()=0

  Gets and removes the next stored key-value pair.

- virtual bool isEmpty () const =0

  Checks if the list of recognized key-value pairs is empty.

10.71.1 Detailed Description

Protocol `class` which defines the interface for pattern recognition `class`es. Each derived `class` is able to parse the beginning of a queue in an attempt to recognize a specific pattern of key-value pairs.

When a pattern is recognized (by parse()), the corresponding key-value pairs are stored in a list to be extracted later (by pop()).

10.71.2 Member Function Documentation

10.71.2.1 virtual bool parse (frontend::Queue & *queue*) [pure virtual]

Parses the beginning of a frontend::Queue.

When a pattern is recognized, the value::Value objects which make it are removed from frontend::Queue and all corresponding key-value pairs are stored inside the this `class` for later queries.

Parameters

queue : frontend::Queue to be parsed.

Returns

If the pattern is recognized, then this method returns `true`. Otherwise, it returns `false`.

Implemented in KeyInSingle, KeysInMatrix, KeysInVector, and Table.

10.71.2.2 virtual std::pair<string, value::Value> pop ( ) [pure virtual]

Gets and removes the next stored key-value pair.

The debug version checks if there is any key-value pair be popped (by calling isEmpty()) and throw an exception when the check fails. Release version has undefined behavior when the list is empty.

Returns

The key-value pair.

Exceptions

LogicError : (Debug build only) When the list is empty.

Implemented in KeyInSingle, KeysInMatrix, KeysInVector, and Table.

10.71.2.3 virtual bool isEmpty ( ) const [pure virtual]

Checks if the list of recognized key-value pairs is empty.

Returns

If the list is empty, this method returns `true`. Otherwise, it returns `false`.

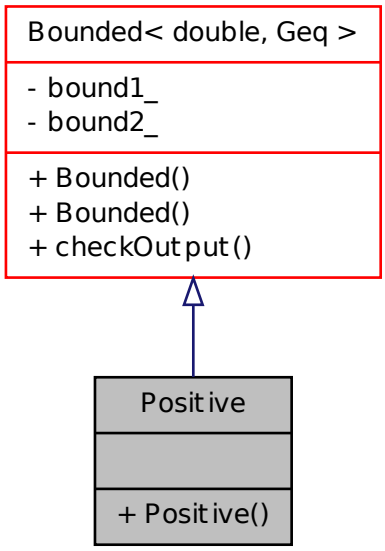
Implemented in KeyInSingle, KeysInMatrix, KeysInVector, and Table.

10.72 Positive Class Reference

A general key for positive values.

`#include <keyvalue/key/generic/Positive.h>`

Inheritance diagram for Positive:



Public Types

- enum
- typedef Traits< double, StdSingle, Default > **Traits_**
- typedef StdSingle< typename Default< double >::OutputType_ > **ConverterType_**
- typedef ConverterType_::InputType_ **InputType_**
- typedef ConverterType_::OutputType_ **OutputType_**

Public Member Functions

- Positive (const string &name)

  Constructs key and sets its name.

- void checkOutput (doubleelement) const

  Compares element against the bound(s).

- string getName () const

  Gets Key's name.

- string getName () const

  Gets name.

- void setName (const string &name)

Sets name.

10.72.1 Detailed Description

A general key for positive values.

10.72.2 Constructor & Destructor Documentation

10.72.2.1 Positive (const string & *name*) [explicit]

Constructs key and sets its name.

This class is derived from Bounded<double, Geq> and set its lower bound to 0.0 at construction time.

10.72.3 Member Function Documentation

10.72.3.1 void checkOutput (double *element*) const [inherited]

Compares *element* against the bound(s).

Parameters

element : Element to be checked.

Exceptions

RuntimeError : If *element* is out of bound(s).

10.72.3.2 string getName ( ) const [inherited]

Gets Key's name.

Returns

The Key's name.

10.72.3.3 string getName ( ) const [inherited]

Gets name.

Returns

The name.

10.72.3.4 void setName (const string & *name*) [inherited]

Sets name.

Parameters

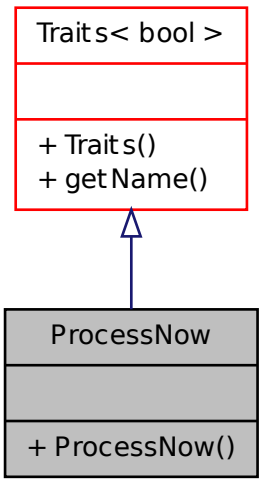
name : The name.

10.73 ProcessNow Class Reference

ProcessNow key.

`#include <keyvalue/key/specific/ProcessNow.h>`

Inheritance diagram for ProcessNow:



Public Types

- enum
- typedef Traits< bool, StdSingle, Default > **Traits_**
- typedef StdSingle< typename Default< bool >::OutputType_ > **ConverterType_**
- typedef ConverterType_::InputType_ **InputType_**
- typedef ConverterType_::OutputType_ **OutputType_**

Public Member Functions

- ProcessNow ()

  Constructs key and set its name.

- string getName () const

  Gets Key's name.

- string getName () const

  Gets name.

- void setName (const string &name)

Sets name.

10.73.1 Detailed Description

ProcessNow key. Key's name is "ProcessNow".

10.73.2 Constructor & Destructor Documentation

10.73.2.1 ProcessNow ( )

Constructs key and set its name.

10.73.3 Member Function Documentation

10.73.3.1 string getName ( ) const [inherited]

Gets Key's name.

Returns

The Key's name.

10.73.3.2 string getName ( ) const [inherited]

Gets name.

Returns

The name.

10.73.3.3 void setName (const string & *name*) [inherited]

Sets name.

Parameters

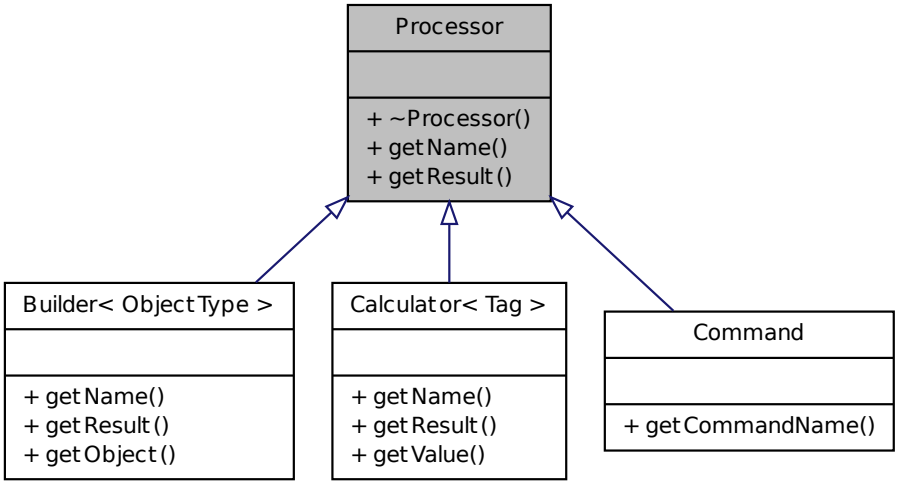
name : The name.

10.74 Processor Class Reference

Protocol `class` which defines Processor interface.

`#include <keyvalue/mngt/Processor.h>`

Inheritance diagram for Processor:



Public Member Functions

- virtual const char * getName () const =0

  Gets Processor's name.

- virtual value::Result getResult (const DataSet &data) const =0

  Gets result of processing a DataSet.

10.74.1 Detailed Description

Protocol `class` which defines Processor interface. This is the abstract base `class` for all processors. Since this `class` is abstract, it cannot be instantiated to build or calculate a particular type of object or value. When one wants to build or calculate a known type of result, then he/she must use ProcessorInstantiator.

10.74.2 Member Function Documentation

10.74.2.1 virtual const char* getName ( ) const [pure virtual]

Gets Processor's name.

Returns

The name.

Implemented in Builder< ObjectType >, and Calculator< Tag >.

10.74.2.2 virtual value::Result getResult (const DataSet & *data*) const [pure virtual]

Gets result of processing a DataSet.

This is a low level method which bypasses the memoization technique implemented by DataSet (see DataSet::mustUpdate()). Therefore, if *processor* is a Processor and *data* is a DataSet, then rather than

```
processor.getResult(data);
```

one should use

```
data.process(processor);
```

Actually, the latter calls the former adding memoization support.

Parameters

data : DataSet to be processed.

Returns

The result.

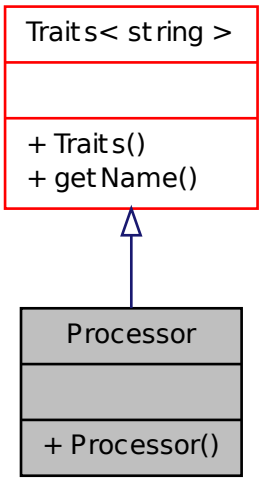
Implemented in Builder< ObjectType >, and Calculator< Tag >.

10.75 Processor Class Reference

Processor key.

`#include <keyvalue/key/specific/Processor.h>`

Inheritance diagram for Processor:



Public Types

- enum
- typedef Traits< string, StdSingle, Default > **Traits_**
- typedef StdSingle< typename Default< string >::OutputType_ > **ConverterType_**
- typedef ConverterType_::InputType_ **InputType_**
- typedef ConverterType_::OutputType_ **OutputType_**

Public Member Functions

- Processor ()
  Constructs key and sets its name.
- string getName () const
  Gets Key's name.
- string getName () const
  Gets name.
- void setName (const string &name)
  Sets name.

10.75.1 Detailed Description

Processor key. Key's name is "Processor".

10.75.2 Constructor & Destructor Documentation

10.75.2.1 Processor ( )

Constructs key and sets its name.

10.75.3 Member Function Documentation

10.75.3.1 string getName ( ) const [inherited]

Gets Key's name.

Returns

The Key's name.

10.75.3.2 string getName ( ) const [inherited]

Gets name.

Returns

The name.

10.75.3.3 void setName (const string & *name*) [inherited]

Sets name.

Parameters

name : The name.

10.76 ProcessorInstantiator< Tag > Class Template Reference

Instantiate a processor given its *Tag*.

`#include <keyvalue/mngt/ProcessorInstantiator.h>`

Static Public Member Functions

- static Processor & getInstance ()

  Gets the unique instance of the Processor assigned to Tag.

10.76.1 Detailed Description

template<typename Tag> class keyvalue::ProcessorInstantiator< Tag >

Instantiate a processor given its *Tag*. To each Processor there should be assigned a *Tag* which is a type that uniquely identifies the Builder or Calculator. Certain parts of *KeyValue* assume that a *Tag* is a (declared but no defined) `class` member of `namespace` tag. Additionally, Builders can also be instantiated from the type of object they build.

10.76.2 Member Function Documentation

10.76.2.1 static Processor& getInstance ( ) [static]

Gets the unique instance of the Processor assigned to *Tag*.

Returns

The unique instance of the Processor assigned to *Tag*.

10.77 ProcessorMngr Class Reference

The processor manager.

`#include <keyvalue/mngt/ProcessorMngr.h>`

Public Member Functions

- void add (const Processor *processor)

  Registers a Processor.

- const Processor & getProcessor (const string &name) const

  Gets Processor of a given name.

- const Command & getCommand (const string &name) const

  Gets Command of a given name.

- value::Vector getCmdList () const

  Gets the list of all registered Commands.

Private Types

- typedef std::map< string, const Processor * > **PrcMapType_**
- typedef std::map< string, const Command * > **CmdMapType_**

Private Attributes

- PrcMapType_ **prcMap_**
- CmdMapType_ **cmdMap_**
- bool dummy_

Static Private Attributes

- static const Processor * **kvProcessors_**
- static const Processor * **processors_**

10.77.1 Detailed Description

The processor manager. This `class` centralizes the processing of all types of results. It maps Processor's names to a corresponding pointer to Processor.

Given the Processor's name and an input sufficient for the processing, this `class` redirects the input to the correct Processor.

Use util::Global to access the global ProcessorMngr.

10.77.2 Member Function Documentation

10.77.2.1 void add (const Processor * *processor*)

Registers a Processor.

Parameters

processor : A pointer to the Processor to be registered.

10.77.2.2 const Processor& getProcessor (const string & *name*) const

Gets Processor of a given name.

Parameters

name : The name of the Processor to be retrieved.

Returns

The Processor.

Exceptions

RuntimeError() : If there is no registered Processor called *name*.

10.77.2.3 const Command& getCommand (const string & *name*) const

Gets Command of a given name.

Parameters

name : The name of the Command to be retrieved.

Returns

The Command.

Exceptions

RuntimeError() : If there is no registered Command called *name*.

10.77.2.4 value::Vector getCmdList ( ) const

Gets the list of all registered Commands.

Returns

A vector with names of registered Commands.

10.77.3 Member Data Documentation

10.77.3.1 bool dummy_ [private]

The following three data members are used to refer to ProcessorInstantiators implemented by bridge libraries. Otherwise, the processor pointers won't be included by the linker and, consequently, registration won't occur.

10.78 Queue Class Reference

Protocol `class` which defines the Queue interface.

`#include <keyvalue/frontend/Queue.h>`

Public Member Functions

- virtual string **getName** () const =0
- virtual bool isEmpty () const =0

  Checks if this Queue is empty.

- virtual value::Value pop ()=0

  Gets the next element and mark it for latter removal.

- virtual void remove ()=0

  Removes popped elements definitely from the Queue.

- virtual void putBack ()=0

  Puts popped elements back on the Queue.

10.78.1 Detailed Description

Protocol `class` which defines the Queue interface. Each front-end (spreadsheet add-in, XML parser, ...) has to implement a Queue which is responsible for converting from its own types to value::Value objects which become elements of the Queue.

Elements can be popped (throw pop()) one by one. When one element is popped, it is marked for later removal. Marked elements can be put back on the Queue (throw putBack()) or they can be definitely removed from it (throw remove()).

10.78.2 Member Function Documentation

10.78.2.1 virtual bool isEmpty ( ) const [pure virtual]

Checks if this Queue is empty.

Returns

If this Queue is empty or all its elements are marked for removal, then this methods returns `true`. Otherwise, it returns `false`.

10.78.2.2 virtual value::Value pop ( ) [pure virtual]

Gets the next element and mark it for latter removal.

The debug version should check if there is any element to be popped (by calling isEmpty()) and throw an exception when the check fails. The release version has undefined behavior when this Queue is empty.

Returns

Next value which is not marked for removal.

Exceptions

LogicError : (Debug build only) When there is no element to be popped.

10.78.2.3 virtual void remove ( ) [pure virtual]

Removes popped elements definitely from the Queue.

The implementation should be such that it is safe to call this method even when there is no element to be removed.

10.78.2.4 virtual void putBack ( ) [pure virtual]

Puts popped elements back on the Queue.

The implementation should be such that it is safe to call this method even when there is no element to be put back.

10.79 Pattern::QueueRaii Class Reference

This `class` implements RAII for a frontend::Queue.

`#include <keyvalue/pattern/Pattern.h>`

Public Member Functions

- QueueRaii (frontend::Queue &queue)
  Acquires resource (frontend::Queue).

Private Attributes

- frontend::Queue & **queue_**

10.79.1 Detailed Description

This `class` implements RAII for a frontend::Queue. By constructing an object type at the beginning of parse() we assure a commit-or-roolback semantics for the Queue and, thus, a strong exception safety guarantee for the parse() method.

10.79.2 Constructor & Destructor Documentation

10.79.2.1 QueueRaii (frontend::Queue & *queue*)

Acquires resource (frontend::Queue).

Parameters

queue : frontend::Queue which will have the commit-or-rollback semantics.

10.80 DataSet::Record Struct Reference

A wrapper around a value::Value.

Public Types

- typedef boost::weak_ptr< Record > **WeakPtr**

Public Member Functions

- **Record** (const value::Value &rawValue)
- **Record** (WeakPtr next, WeakPtr driver)

Public Attributes

- value::Value **rawValue_**
- bool **isBusy_**
- WeakPtr **next_**
- WeakPtr **driver_**
- boost::any **cache_**
- std::type_info * **keyType_**

10.80.1 Detailed Description

A wrapper around a value::Value. Beside each key-value pair the DataSet must stores data needed to process the value (*e.g.*, cached results). This `struct` wrapps a value::Value and all those extra information.

10.81 Report Class Reference

MessageImpl instantiation used for reporting results.

`#include <keyvalue/sys/message/MessageImpl.h>`

10.81.1 Detailed Description

MessageImpl instantiation used for reporting results. `typedef MessageImpl<3> Report;`

10.82 Repository Class Reference

The Repository of DataSets.

`#include <keyvalue/mngt/Repository.h>`

Classes

- class NotFound

  Exception thrown by get() when a DataSet is not found.

Public Member Functions

- void add (shared_ptr< DataSet > dataSet)

  Inserts a DataSet to Repository.

- bool erase (const string &name)

  Erases a DataSet from Repository.

- size_t clear ()

  Erases all DataSets from Repository.

- shared_ptr< DataSet > find (const string &name) const

  Gets a DataSet stored in the Repository.

- shared_ptr< DataSet > getDataSet (const string &name) const

  Gets a DataSet stored in the Repository.

- size_t getSize () const

  Gets the current number of stored DataSets.

- value::Vector getList () const

  Gets the list of all stored DataSets.

Private Types

- typedef std::map< string, shared_ptr< DataSet > > **Map_**
- typedef Map_::value_type **Pair_**

Private Attributes

- Map_ **map_**

10.82.1 Detailed Description

The Repository of DataSets. Use util::Global to access the global Repository.

10.82.2 Member Function Documentation

10.82.2.1 void add (shared_ptr< DataSet > *dataSet*)

Inserts a DataSet to Repository.

If a DataSet with the same name already exists in the repository, then it will be destroyed and replaced by the new one.

If the DataSet name starts with '#', then nothing will be done.

Debug build asserts that name is not empty. Relase build has undefined behavior in this circumstance.

Parameters

dataSet : Pointer to the DataSet to be inserted.

Exceptions

LogicError : (Debug build only) If DataSet name is empty.

10.82.2.2 bool erase (const string & *name*)

Erases a DataSet from Repository.

Parameters

name : The name of DataSet to be erased.

Returns

This method returns `true` if the DataSet was found and erased. Otherwise it returns `false`.

10.82.2.3 size_t clear ( )

Erases all DataSets from Repository.

Returns

The number of DataSets erased.

10.82.2.4 shared_ptr<DataSet> find (const string & *name*) const

Gets a DataSet stored in the Repository.

Parameters

name : The name of the required DataSet.

Returns

If DataSet is fond, then this method returns a pointer to it. Otherwise, it returns a null pointer.

10.82.2.5 shared_ptr<DataSet> getDataSet (const string & *name*) const

Gets a DataSet stored in the Repository.

Parameters

name : The name of the required DataSet.

Exceptions

NotFound : If DataSet *name* is not found.

Returns

A pointer to the DataSet.

10.82.2.6 size_t getSize ( ) const

Gets the current number of stored DataSets.

Returns

The number of stored DataSets.

10.82.2.7 value::Vector getList ( ) const

Gets the list of all stored DataSets.

Returns

A vector with names of stored DataSets.

10.83 Result Class Reference

A single-valued container for ObjectPtr or Value.

`#include <keyvalue/value/Result.h>`

Public Types

- typedef boost::variant< ObjectPtr, Value > **DataType_**

Public Member Functions

- Result ()
  Default constructor sets content to null ObjectPtr.
- Result (const DataType_ &data)
  Builds and sets content.
- template<typename Rhs >
  Result (const Rhs &rhs)
  Builds and sets content.
- template<typename Rhs >
  Result & operator= (const Rhs &rhs)
  Assignment operator.
- ObjectPtr getObjectPtr () const
  Gets content.
- ObjectPtr getObjectPtr ()
  Non `const` getObjectPtr().
- const Value * getValue () const

Gets content.

- Value * getValue ()

  Non `const` *getValue().*

Private Attributes

- DataType_ **data_**

10.83.1 Detailed Description

A single-valued container for ObjectPtr or Value.

10.83.2 Constructor & Destructor Documentation

10.83.2.1 Result ( )

Default constructor sets content to null ObjectPtr.

10.83.2.2 Result (const DataType_ & *data*)

Builds and sets content.

Takes type on 1-level-down on the value hierarchy.

Parameters

data : The content.

10.83.2.3 Result (const Rhs & *rhs*)

Builds and sets content.

Takes type on 2-or-more-levels-down on the value hierarchy.

Parameters

rhs : The content.

If you get an error here about 'keyvaluevalue::Parent<Rhs>', this means that Rhs isn't in a lower level in the hierarchy.

10.83.3 Member Function Documentation

10.83.3.1 Result & operator= (const Rhs & *rhs*)

Assignment operator.

Takes type on 2-or-more-levels-down on the value hierarchy.

Parameters

rhs : The data to be assigned.

Returns

A reference to `this`.

If you get an error here about 'keyvaluevalue::Parent<P>', this means that Rhs isn't in a lower level in the hierarchy.

10.83.3.2 ObjectPtr getObjectPtr ( ) const

Gets content.

Returns

A `const` pointer to the content, if it is an ObjectPtr. Otherwise, a null pointer.

10.83.3.3 ObjectPtr getObjectPtr ( )

Non `const` getObjectPtr().

10.83.3.4 const Value* getValue ( ) const

Gets content.

Returns

A `const` pointer to the content, if it is a Value. Otherwise, a null pointer.

10.83.3.5 Value* getValue ( )

Non `const` getValue().

10.84 RuntimeError Class Reference

Base `class` for all *KeyValue* exceptions occurring at runtime.

`#include <keyvalue/sys/exception/Exception.h>`

10.84.1 Detailed Description

Base `class` for all *KeyValue* exceptions occurring at runtime. `typedef ExceptionImpl<std::runtime_error, Error> RuntimeError;`

10.85 Saver< SavedType > Class Template Reference

Saves a variable reference at construction time to restore/set its value at destruction time.

Public Member Functions

- Saver (SavedType &data)

  Constructs a Saver for a variable.

- Saver (SavedType &data, const SavedType &onExit)

  Constructs a Saver for a variable and sets the value it will get at time the of Saver's destruction.

- SavedType getSaved () const

  Gets saved value.

Private Attributes

- SavedType & **data_**
- SavedType **onExit_**

10.85.1 Detailed Description

template<typename SavedType> class keyvalue::util::Saver< SavedType >

Saves a variable reference at construction time to restore/set its value at destruction time. This `class` applies Stroustrup's RAII technique to reset a (probably, static) variable when Saver go out of scope. The variable can be reset to the value it had when Saver was constructed or set to another given value.

Parameters

SavedType : (`template` parameter) The type of saved value.

10.85.2 Constructor & Destructor Documentation

10.85.2.1 Saver (SavedType & *data*) `[explicit]`

Constructs a Saver for a variable.

Parameters

data : the variable to be saved.

10.85.2.2 Saver (SavedType & *data*, const SavedType & *onExit*)

Constructs a Saver for a variable and sets the value it will get at time the of Saver's destruction.

Parameters

data : the variable to be saved.

onExit : the value that *data* is set to when Saver is destroyed.

10.85.3 Member Function Documentation

10.85.3.1 SavedType getSaved ( ) const

Gets saved value.

Returns

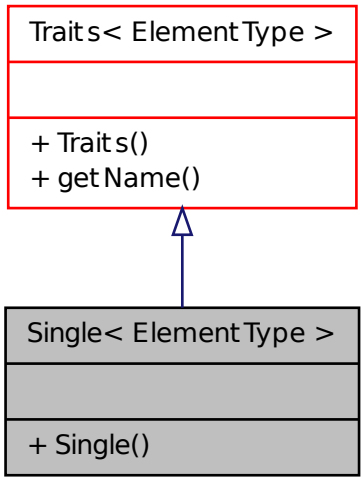
The value.

10.86 Single< ElementType > Class Template Reference

Generic key whose converter type is StdSingle.

`#include <keyvalue/key/generic/Single.h>`

Inheritance diagram for Single< ElementType >:



Public Types

- enum
- typedef Traits< ElementType, StdSingle, Default > **Traits_**
- typedef StdSingle< typename Default< ElementType >::OutputType_ > **ConverterType_**
- typedef ConverterType_::InputType_ **InputType_**
- typedef ConverterType_::OutputType_ **OutputType_**

Public Member Functions

- Single (const string &name)

Constructs the key and sets its name.

- string getName () const

  Gets Key's name.

- string getName () const

  Gets name.

- void setName (const string &name)

  Sets name.

10.86.1 Detailed Description

template<typename ElementType> class keyvalue::key::Single< ElementType >

Generic key whose converter type is StdSingle.

Parameters

ElementType : (`template` parameter) The output type.

10.86.2 Constructor & Destructor Documentation

10.86.2.1 Single (const string & *name*) [explicit]

Constructs the key and sets its name.

10.86.3 Member Function Documentation

10.86.3.1 string getName ( ) const [inherited]

Gets Key's name.

Returns

The Key's name.

10.86.3.2 string getName ( ) const [inherited]

Gets name.

Returns

The name.

10.86.3.3 void setName (const string & *name*) [inherited]

Sets name.

Parameters

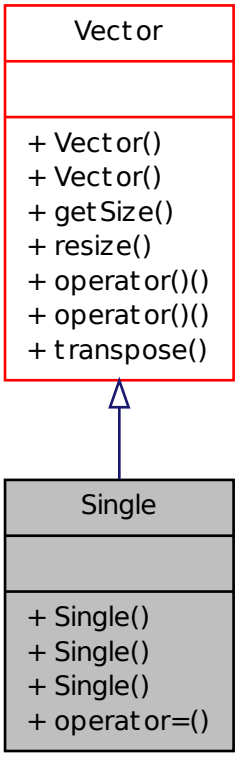
name : The name.

10.87 Single Class Reference

A 1 x 1 Matrix.

`#include <keyvalue/value/Single.h>`

Inheritance diagram for Single:



Public Member Functions

- Single ()

  Default constructor sets content to Variant's default.

- Single (const Variant &data)

  Builds and sets content.

- template<typename Rhs >
  Single (const Rhs &rhs)

  Builds and sets content.

- template<typename Rhs >
  Single & operator= (const Rhs &rhs)

  Assignment operator.

- size_t getSize () const

  Gets Vector size.

- void resize (size_t size)

  Resizes the vector.

- void resize (size_t nRows, size_t nCols)

  Resizes the matrix.

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

  Provides access to the i-th element.

- Variant & operator() (size_t i)

  Non `const` operator()().

- const Variant & operator() (size_t i, size_t j) const

  Provides access to the element in the i-th row and j-th column.

- Variant & operator() (size_t i, size_t j)

  Non `const` operator()().

- void transpose ()

  Transposes the vector.

- size_t getNRows () const

  Gets number of rows.

- size_t getNCols () const

  Gets number of columns.

- bool operator== (const Matrix &rhs) const

  Comparison operator.

Protected Member Functions

- void transposeVector ()

  Transposes the matrix if one of its dimensions is 1.

10.87.1 Detailed Description

A 1 x 1 Matrix. See Matrix documentation for more details.

10.87.2 Constructor & Destructor Documentation

10.87.2.1 Single ( )

Default constructor sets content to Variant's default.

10.87.2.2 Single (const Variant & *data*)

Builds and sets content.

Takes type on 1-level-down on the value hierarchy.

Parameters

data : The content.

10.87.2.3 Single (const Rhs & *rhs*)

Builds and sets content.

Takes type on 2-or-more-levels-down on the value hierarchy.

Parameters

rhs : The content.

If you get an error here about 'keyvaluevalue::Parent<Rhs>', this means that Rhs isn't in a lower level in the hierarchy.

10.87.3 Member Function Documentation

10.87.3.1 Single & operator= (const Rhs & *rhs*)

Assignment operator.

Takes type on 2-or-more-levels-down on the value hierarchy.

Parameters

rhs : The data to be assigned.

Returns

A reference to `this`.

If you get an error here about 'keyvaluevalue::Parent<Rhs>', this means that Rhs isn't in a lower level in the hierarchy.

10.87.3.2 size_t getSize ( ) const [inherited]

Gets Vector size.

Returns

The size.

10.87.3.3 void resize (size_t *size*) [inherited]

Resizes the vector.

The size of a vector can not increase. Debug build check for this and an exception is thrown when the test fails. Release build has undefined behavior in this case.

Parameters

size : New size.

Exceptions

LogicError : (Debug build only) If the new dimension is greater than the original one.

10.87.3.4 void resize (size_t *nRows*, size_t *nCols*) [inherited]

Resizes the matrix.

The size of a matrix (*i.e.*, number of rows times number of columns) can not increase. Debug build check for this and an exception is thrown when the test fails. Release build has undefined behavior in this case.

The contents of a matrix after resizing is undefined.

Parameters

nRows : number of rows;

nCols : number of columns.

Exceptions

LogicError : (Debug build only) If the new size is greater than the original one.

10.87.3.5 const Variant& operator() (size_t *i*) const [inherited]

Provides access to the *i-th* element.

Debug build performs bound check and throws an exception when the check fail. Release build has undefined behavior in that circumstance.

Parameters

i : Element's index.

Returns

A `const` reference to the *i-th* element.

Exceptions

LogicError : (Debug build only) When *i* is not smaller than vector's dimension.

10.87.3.6 Variant& operator() (size_t *i*) [inherited]

Non `const` operator()().

10.87.3.7 const Variant& operator() (size_t *i*, size_t *j*) const [inherited]

Provides access to the element in the *i-th* row and *j-th* column.

Debug build perform bound checks and throws an exception on failure. Release build has undefined behavior in that case.

Parameters

i : Row index;

j : Column index.

Returns

A `const` reference to *(i, j)*-th element.

Exceptions

LogicError : (Debug build only) If *i* is not smaller than the number of rows or *j* is not smaller than the number of columns.

10.87.3.8 Variant& operator() (size_t *i*, size_t *j*) [inherited]

Non `const` operator()().

10.87.3.9 void transpose ( ) [inherited]

Transposes the vector.

10.87.3.10 size_t getNRows ( ) const [inherited]

Gets number of rows.

Returns

The number of rows.

10.87.3.11 size_t getNCols ( ) const [inherited]

Gets number of columns.

Returns

The number of columns.

10.87.3.12 bool operator== (const Matrix & *rhs*) const [inherited]

Comparison operator.

Recall that Matrix has reference semantics. Consider the code:

```
Matrix m1(2,2);
m1(0,0) = m1(0,1) = m1(1,0) = m1(1,1) = 1.0;
Matrix m2(2,2);
m2(0,0) = m2(0,1) = m2(1,0) = m2(1,1) = 1.0;
Matrix m3(m1);
```

Then `m1 == m2` is `false` and `m1 == m3` is `true`.

Parameters

rhs : The Value to be compared against.

Returns

This method returns `true` if *this and *rhs* refer to the same data. Otherwise it returns `false`.

10.87.3.13 void transposeVector ( ) [protected, inherited]

Transposes the matrix if one of its dimensions is 1.

In-place matrix transposition is a complicated stuff. See

http://en.wikipedia.org/wiki/In-place_matrix_transposition

However, since the storage is linear, the memory layouts of n x 1 and 1 x n matrix, are the same. Hence, we only need to swap dimensions. This method is intent to perform only this simple transposition and it is protected to be called by Vector which derives from this class.

Debug build checks if either of the dimensions is 1 and throws an exception if the test fails. Release build has undefined behaviour in that circumstance.

Exceptions

LogicError : (Debug build only) If none of dimensions is 1.

10.88 StdMatrix< ElementType > Class Template Reference

Converter from value::Matrix to `shared_ptr<std::vector<std::vector<ElementType>>>`.

`#include <keyvalue/key/converter/StdMatrix.h>`

Public Types

- typedef value::Matrix InputType_

  Compulsory member.

- typedef shared_ptr< std::vector< std::vector< ElementType > > > OutputType_

  Compulsory member.

Public Member Functions

- StdMatrix (const InputType_ &input, OutputType_ *cache)

  Compulsory constructor.

- bool isEmpty () const

  Compulsory member: Checks if conversion has finished.

- value::Variant pop ()

  Compulsory member: Gets next input element.

- void insert (const ElementType &element)

  Compulsory member: Inserts a new element into the output container.

- OutputType_ getOutput () const

  Compulsory member: Gets output container.

- bool mustUpdate () const

  Compulsory member: Informs if cached value is up to date.

Private Attributes

- const InputType_ & **input_**
- OutputType_ **output_**
- OutputType_ * **cache_**
- size_t **nRows_**
- size_t **nCols_**
- size_t **i_**
- size_t **j_**
- bool **mustUpdate_**

10.88.1 Detailed Description

template<typename ElementType> class keyvalue::key::StdMatrix< ElementType >

Converter from value::Matrix to `shared_ptr<std::vector<std::vector<ElementType>>>`. StdSingle together with StdMatrix and StdMatrix might serve as samples for bridge developers who want to implement their own converters. A few constraints have to be verified. Reading documentation for these three `class`es before implement any converter is strongly advisable.

Parameters

ElementType : The output container's element type.

Return values

InputType_ : value::Matrix;

OutputType_ : `shared_ptr<std::vector<std::vector<Element>>>`.

10.88.2 Member Typedef Documentation

10.88.2.1 typedef value::Matrix InputType_

Compulsory member.

10.88.2.2 typedef shared_ptr<std::vector<std::vector<ElementType> > > OutputType_

Compulsory member.

Notice that getOutput() returns an OutputType_ object by value. Hence, it is advisable for this type to be cheap-to-copy.

10.88.3 Constructor & Destructor Documentation

10.88.3.1 StdMatrix (const InputType_ & *input*, OutputType_ * *cache*)

Compulsory constructor.

Internally, the converter must store a copy of the input and a copy of previously cached output (if any). It have also to compare the new computed output with the cached one to see if anything dependent on the cached output must bu updated. (See mustUpdate().)

Parameters

input : Input data.

cache : A pointer to the previoulsy cached value if there exist one. Otherwise, a null pointer must be given.

10.88.4 Member Function Documentation

10.88.4.1 bool isEmpty ( ) const

Compulsory member: Checks if conversion has finished.

Returns

This methods must return `true` if all elements of input data have been processed. Otherwise, it must returns `false`.

10.88.4.2 value::Variant pop ( )

Compulsory member: Gets next input element.

Returns

A copy of next input element to be processed.

10.88.4.3 void insert (const ElementType & *element*)

Compulsory member: Inserts a new element into the output container.

Parameters

element : The element to be inserted into the output container.

10.88.4.4 StdMatrix< ElementType >::OutputType_ getOutput ( ) const

Compulsory member: Gets output container.

Returns

A copy of output container.

10.88.4.5 bool mustUpdate ( ) const

Compulsory member: Informs if cached value is up to date.

Returns

This method returns `true` if cached value is out of date. Otherwise it returns `false`.

10.89 StdSingle< ElementType > Class Template Reference

Converter from value::Single to *ElementType*.

`#include <keyvalue/key/converter/StdSingle.h>`

Public Types

- typedef value::Single InputType_

  Compulsory member.

- typedef ElementType OutputType_

  Compulsory member.

Public Member Functions

- StdSingle (const InputType_ &input, OutputType_ *cache)

  Compulsory constructor.

- bool isEmpty () const

  Compulsory member: Checks if conversion has finished.

- value::Variant pop ()

Compulsory member: Gets next input element.

- void insert (const ElementType &element)

  Compulsory member: Inserts a new element into the output container.

- OutputType_ getOutput () const

  Compulsory member: Gets output container.

- bool mustUpdate () const

  Compulsory member: Informs if cached value is up to date.

Private Attributes

- const InputType_ & **input_**
- OutputType_ *const **cache_**
- OutputType_ **output_**
- bool **empty_**
- bool **mustUpdate_**

10.89.1 Detailed Description

template<typename ElementType> class keyvalue::key::StdSingle< ElementType >

Converter from value::Single to *ElementType*. Values provided by the end users must be converted to other types before being forwarded to the core library. This converter is used when the input value is a value::Single.

StdSingle together with StdVector and StdMatrix might serve as samples for bridge developers who want to implement their own converters. A few constraints have to be verified. Reading documentation for these three `classes` before implement any converter is strongly advisable.

These constraints are such that for converters taking value::Single as input there is little (if any) room for other converters. However, converters taking value::Vector or value::Matrix can be tailored to specific core library needs.

The constraints are composed by certain `typedefs` and methods that must be member of any converter. They will be explained below.

Parameters

ElementType : The output type.

Return values

InputType_ : value::Single;

OutputType_ : *ElementType*.

10.89.2 Member Typedef Documentation

10.89.2.1 typedef value::Single InputType_

Compulsory member.

10.89.2.2 typedef ElementType OutputType_

Compulsory member.

Notice that getOutput() returns an OutputType_ object by value. Hence, it is advisable for this type to be cheap-to-copy.

10.89.3 Constructor & Destructor Documentation

10.89.3.1 StdSingle (const InputType_ & *input*, OutputType_ * *cache*)

Compulsory constructor.

Internally, the converter must store a copy of the input and a copy of previously cached output (if any). Also, it must compare the new computed output with the cached one to see if anything dependent on the cached output must be updated. (See mustUpdate().)

Parameters

input : Input data.

cache : A pointer to the previously cached value if there exist any. Otherwise, a null pointer must be given.

10.89.4 Member Function Documentation

10.89.4.1 bool isEmpty ( ) const

Compulsory member: Checks if conversion has finished.

Returns

This methods must return `true` if all elements of input data have been processed. Otherwise, it must returns `false`.

10.89.4.2 value::Variant pop ( )

Compulsory member: Gets next input element.

There should be at least one element to be popped, that is, isEmpty() must be `false`. The debug build checks this condition and throws an exception in case of failure. The release build has undefined behavior in this case.

Returns

A copy of next input element to be processed.

Exceptions

LogicError : (Debug build only) If isEmpty() is `true`.

10.89.4.3 void insert (const ElementType & *element*)

Compulsory member: Inserts a new element into the output container.

Parameters

element : The element to be inserted into the output container.

10.89.4.4 StdSingle< ElementType >::OutputType_ getOutput ( ) const

Compulsory member: Gets output container.

Returns

A copy of output container.

10.89.4.5 bool mustUpdate ( ) const

Compulsory member: Informs if cached value is up to date.

Returns

This method returns `true` if cached value is out of date. Otherwise it returns `false`.

10.90 StdVector< ElementType > Class Template Reference

Converter from value::Vector to `shared_ptr<std::vector<ElementType>>`.

`#include <keyvalue/key/converter/StdVector.h>`

Public Types

- typedef value::Vector InputType_

  Compulsory member.

- typedef shared_ptr< std::vector< ElementType > > OutputType_

  Compulsory member.

Public Member Functions

- StdVector (const InputType_ &input, OutputType_ *cache)

  Compulsory constructor.

- bool isEmpty () const

  Compulsory member: Check if conversion has finished.

- value::Variant pop ()

Compulsory member: Gets next input element.

- void insert (const ElementType &element)

  Compulsory member: Inserts a new element into the output container.

- OutputType_ getOutput () const

  Compulsory member: Gets output container.

- bool mustUpdate () const

  Compulsory member: Informs if cached value is up to date.

Private Attributes

- const InputType_ & **input_**
- OutputType_ **output_**
- OutputType_ *const **cache_**
- const size_t **size_**
- size_t **i_**
- bool **mustUpdate_**

10.90.1 Detailed Description

template<typename ElementType> class keyvalue::key::StdVector< ElementType >

Converter from value::Vector to `shared_ptr<std::vector<ElementType>>`. StdSingle together with StdVector and StdMatrix might serve as samples for bridge developers who want to implement their own converters. A few constraints have to be verified. Reading documentation for these three `class`es before implement any converter is strongly advisable.

Parameters

ElementType : The output container's element type.

Return values

InputType_ : value::Vector;

OutputType_ : `shared_ptr<std::vector<Element>>`.

10.90.2 Member Typedef Documentation

10.90.2.1 typedef value::Vector InputType_

Compulsory member.

10.90.2.2 typedef shared_ptr<std::vector<ElementType> > OutputType_

Compulsory member.

Notice that getOutput() returns an OutputType_ object by value. Hence, it is advisable for this type to be cheap-to-copy.

10.90.3 Constructor & Destructor Documentation

10.90.3.1 StdVector (const InputType_ & *input,* OutputType_ * *cache*)

Compulsory constructor.

Internally, the converter must store a copy of the input and a copy of previously cached output (if any). It have also to compare the new computed output with the cached one to see if anything dependent on the cached output must bu updated. (See mustUpdate().)

Parameters

input : Input data.

cache : A pointer to the previoulsy cached value if there exist one. Otherwise, a null pointer must be given.

10.90.4 Member Function Documentation

10.90.4.1 bool isEmpty ( ) const

Compulsory member: Check if conversion has finished.

Returns

This methods must return `true` if all elements of input data have been processed. Otherwise, it must returns `false`.

10.90.4.2 value::Variant pop ( )

Compulsory member: Gets next input element.

Returns

A copy of next input element to be processed.

10.90.4.3 void insert (const ElementType & *element*)

Compulsory member: Inserts a new element into the output container.

Parameters

element : The element to be inserted into the output container.

10.90.4.4 StdVector< ElementType >::OutputType_ getOutput ( ) const

Compulsory member: Gets output container.

Returns

A copy of output container.

10.90.4.5 bool mustUpdate ( ) const

Compulsory member: Informs if cached value is up to date.

Returns

This method returns `true` if cached value is out of date. Otherwise it returns `false`.

10.91 StrictlyDecreasing Class Reference

StrictlyDecreasing monotone `class`.

`#include <keyvalue/key/generic/monotone/StrictlyDecreasing.h>`

Static Public Member Functions

- template<typename ElementType >
  static bool check (const ElementType &x, const ElementType &y)

  Performs the comparison.

- static const char * getName ()

  Gets type name.

10.91.1 Detailed Description

StrictlyDecreasing monotone `class`. Given two consecutive values of a sequence, x[i] and x[i+1], this `class` checks if x[i] > x[i+1].

10.91.2 Member Function Documentation

10.91.2.1 static bool check (const ElementType & *x*, const ElementType & *y*) [static]

Performs the comparison.

Parameters

x : 1st value to be checked;

y : 2nd value to be checked.

ElementType : (`template` parameter) Type of x and y. Accepted types are `double`, `ptime`, `string` and `unsigned int`. (Other types will generate a link error.)

Returns

($x > y$).

10.91.2.2 static const char* getName ( ) [static]

Gets type name.

Returns

"strictly decreasing".

10.92 StrictlyIncreasing Class Reference

StrictlyIncreasing monotone `class`.

`#include <keyvalue/key/generic/monotone/StrictlyIncreasing.h>`

Static Public Member Functions

- template<typename ElementType >
  static bool check (const ElementType &x, const ElementType &y)

  Performs the comparison.

- static const char * getName ()

  Gets type name.

10.92.1 Detailed Description

StrictlyIncreasing monotone `class`. Given two consecutive values of a sequence, x[i] and x[i+1], this `class` checks if x[i] < x[i+1].

10.92.2 Member Function Documentation

10.92.2.1 static bool check (const ElementType & *x*, const ElementType & *y*) [static]

Performs the comparison.

Parameters

x : 1st value to be checked;

y : 2nd value to be checked.

ElementType : (`template` parameter) Type of x and y. Accepted types are `double`, `ptime`, `string` and `unsigned int`. (Other types will generate a link error.)

Returns

($x < y$).

10.92.2.2 static const char* getName ( ) [static]

Gets type name.

Returns

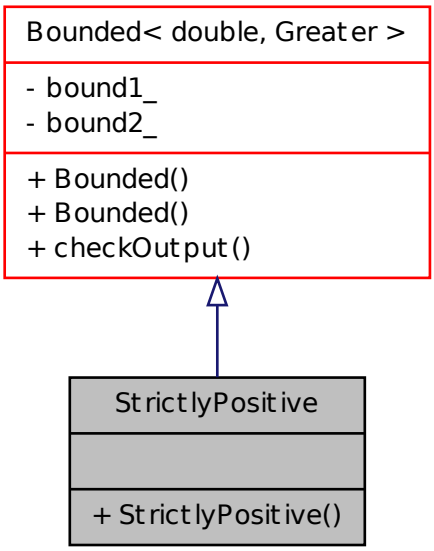
"strictly increasing".

10.93 StrictlyPositive Class Reference

A general key whose value is a strictly positive number.

`#include <keyvalue/key/generic/StrictlyPositive.h>`

Inheritance diagram for StrictlyPositive:



Public Types

- enum
- typedef Traits< double, StdSingle, Default > **Traits_**
- typedef StdSingle< typename Default< double >::OutputType_ > **ConverterType_**
- typedef ConverterType_::InputType_ **InputType_**
- typedef ConverterType_::OutputType_ **OutputType_**

Public Member Functions

- StrictlyPositive (const string &name)

Constructs key and sets its name.

- void checkOutput (doubleelement) const

  Compares element against the bound(s).

- string getName () const

  Gets Key's name.

- string getName () const

  Gets name.

- void setName (const string &name)

  Sets name.

10.93.1 Detailed Description

A general key whose value is a strictly positive number.

10.93.2 Constructor & Destructor Documentation

10.93.2.1 StrictlyPositive (const string & *name*) [explicit]

Constructs key and sets its name.

10.93.3 Member Function Documentation

10.93.3.1 void checkOutput (double *element*) const [inherited]

Compares *element* against the bound(s).

Parameters

element : Element to be checked.

Exceptions

RuntimeError : If *element* is out of bound(s).

10.93.3.2 string getName ( ) const [inherited]

Gets Key's name.

Returns

The Key's name.

10.93.3.3 string getName ( ) const [inherited]

Gets name.

Returns

The name.

10.93.3.4 void setName (const string & *name*) [inherited]

Sets name.

Parameters

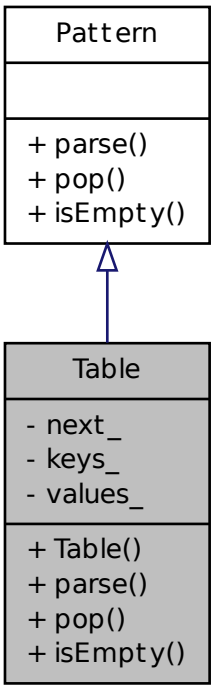
name : The name.

10.94 Table Class Reference

Pattern where a value::Matrix contains the row, column and table keys.

`#include <keyvalue/pattern/Pattern.h>`

Inheritance diagram for Table:



Public Member Functions

- bool parse (frontend::Queue &queue)

  Parses the beginning of a frontend::Queue.

- std::pair< string, value::Value > pop ()

  Gets and removes the next stored key-value pair.

- bool isEmpty () const

  Checks if the list of recognized key-value pairs is empty.

Private Attributes

- size_t **next_**
- string **keys_** [3]
- value::Value **values_** [3]

10.94.1 Detailed Description

Pattern where a value::Matrix contains the row, column and table keys. The value::Matrix *M=M(i, j)* is a *m* x *n* matrix with $m > 2$ and $n > 2$. It defines three key-value pairs (row, column and table) which can be given in two different formats.

- Format 1: Row key is in *M(1, 0)* and its value is the vector *M(i, 0)* for $i = 2, ..., m$. Column key is in *M(0, 1)* and its value is the vector *M(0, j)* for $j = 2, ..., n$.

- Format 2: Row key is in *M(2, 0)* and its value is the vector *M(i, 1)* for $i = 2, ..., m$. Column key is in *M(0, 2)* and its value is the vector *M(1, j)* for $j = 2, ..., n$.

For both formats, key table is in *M(0, 0)* and its value is the matrix *M(i, j)* for $i = 2, ..., m$ and $j = 2, ..., n$.

10.94.2 Member Function Documentation

10.94.2.1 bool parse (frontend::Queue & *queue*) [virtual]

Parses the beginning of a frontend::Queue.

When a pattern is recognized, the value::Value objects which make it are removed from frontend::Queue and all corresponding key-value pairs are stored inside the this `class` for later queries.

Parameters

queue : frontend::Queue to be parsed.

Returns

If the pattern is recognized, then this method returns `true`. Otherwise, it returns `false`.

Implements Pattern.

10.94.2.2 std::pair<string, value::Value> pop ( ) [virtual]

Gets and removes the next stored key-value pair.

The debug version checks if there is any key-value pair be popped (by calling isEmpty()) and throw an exception when the check fails. Release version has undefined behavior when the list is empty.

Returns

The key-value pair.

Exceptions

LogicError : (Debug build only) When the list is empty.

Implements Pattern.

10.94.2.3 bool isEmpty ( ) const [virtual]

Checks if the list of recognized key-value pairs is empty.

Returns

If the list is empty, this method returns `true`. Otherwise, it returns `false`.

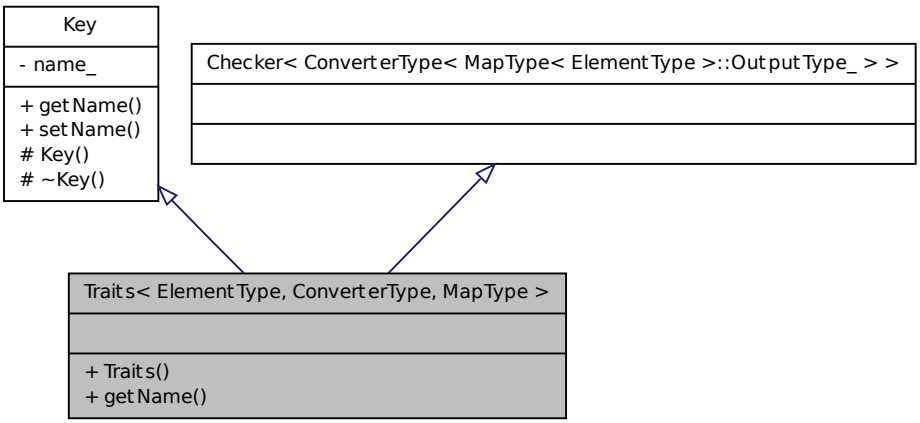
Implements Pattern.

10.95 Traits< ElementType, ConverterType, MapType > Class Template Reference

Real keys must derive from adequate instantiations of this `template class`.

`#include <keyvalue/key/Traits.h>`

Inheritance diagram for Traits< ElementType, ConverterType, MapType >:



Public Types

- enum { **isBasicOrEnum** = util::IsBasicOrEnum<ElementType>::value }
- typedef Traits< ElementType, ConverterType, MapType > **Traits_**
- typedef ConverterType< typename MapType< ElementType >::OutputType_ > **ConverterType_**
- typedef ConverterType_::InputType_ **InputType_**
- typedef ConverterType_::OutputType_ **OutputType_**

Public Member Functions

- Traits (const string &name)

  Initializes Key's name.

- string getName () const

  Gets Key's name.

- string getName () const

  Gets name.

- void setName (const string &name)

  Sets name.

10.95.1 Detailed Description

template<typename ElementType, template< typename > class ConverterType = StdSingle, template< typename > class MapType = Default> class keyvalue::key::Traits< ElementType, ConverterType, MapType >

Real keys must derive from adequate instantiations of this `template class`. Keys have more than just a name: Each of them is associated to a value which need to be processed. The processing depends on many types which are passed as `template` parameters stored inside this `class`.

The input value stored in the DataSet, can be either a value::Single, value::Vector or value::Matrix. The content must be converted to another kind of container suitable for the core library. For instance, a key named "Length" might be associated to a value::Single type of input which must be converted to `double`. On the same way, a key named "Lengths" might be associated to an input of type value::Vector which should be converted to `std::vector<double>`.

To allow more flexibility, conversions are responsibility of converter `classes`. Traits takes a converter *ConverterType* as a parameter. KeyValue library provides three standard implementations: StdSingle, StdVector and StdMatrix.

Without get into details, notice that *ConverterType* is a `template` depending on a type *T*. Essentially, StdSingle<*T*> converts from value::Single to *T*; StdVector<*T*> converts from value::Vector to `std::vector<`*T*`>`; StdMatrix<*T*> converts from value::Matrix to `std::vector<std::vector<`*T*`>>`. Bridge developers might implement converters whose output is more suitable for the core library (*e.g.* a converter from value::Matrix to `boost::matrix<`*T*`>`). (For more details, in particular, the requirements on converters, see StdSingle, StdVector and StdMatrix documentation.)

In some cases, the value contains `strings` (names) which must be mapped to corresponding values of type *ElementType*. For instance, the value associated to the key 'Logger' is supposed to be a logger::Logger.

The user refers to the correct logger::Logger object through its name (*e.g.* 'GlobalLogger'). In this case, there must be a map that, provided the name, retrieves a pointer to the corresponding object. Indeed, this mapping is performed with Repository's help. Other kinds of maps are available.

Traits takes a map *MapType* which implies constraints on Traits' derived `classes`. For more details see the relevant map documentation.

If *MapType* is not specified, then Default will be selected.

Finally, values associated to some keys are expected to fulfill certain criteria (*e.g.* a price cannot be negative.) Therefore, some sanity checks on the value might be needed. Those checks are performed by Checker which is a `template class` that Traits derives from.

Parameters

ElementType : (`template` parameter) Type of elements in converter's output;

ConverterType : (`template template` parameter) Type of container converter.

MapType : (`template parameter`) Type of map. (See description above.)

Return values

Traits_ : Complete instantiation of Traits;

ConverterType_ : Complete instantiation of *ConverterType*;

InputType_ : Converter's input type;

OutputType_ : Converter's output type.

10.95.2 Constructor & Destructor Documentation

10.95.2.1 Traits (const string & *name*)

Initializes Key's name.

Parameters

name : The Key's name.

10.95.3 Member Function Documentation

10.95.3.1 string getName ( ) const

Gets Key's name.

Returns

The Key's name.

10.95.3.2 string getName ( ) const `[inherited]`

Gets name.

Returns

The name.

10.95.3.3 void setName (const string & *name*) [inherited]

Sets name.

Parameters

name : The name.

10.96 TypeName< Type > Struct Template Reference

Meta function which returns the name of type.

`#include <keyvalue/value/TypeName.h>`

Static Public Attributes

- static const char * **value_**

10.96.1 Detailed Description

template<typename Type> struct keyvalue::value::TypeName< Type >

Meta function which returns the name of type.

Parameters

Type : (`template` parameter) The type.

10.97 Value Class Reference

A single-valued container for Single, Vector or Matrix.

`#include <keyvalue/value/Value.h>`

Public Types

- typedef boost::variant< Single, Vector, Matrix > **DataType_**

Public Member Functions

- Value ()

  Default constructor sets content to Singles's default.

- Value (const DataType_ &data)

  Builds and sets content.

- template<typename Rhs >
  Value (const Rhs &rhs)

  Builds and sets content.

- template<typename Rhs >
  Value & operator= (const Rhs &rhs)

  Assignment operator.

- bool operator== (const Value &rhs) const

  Comparison operator.

- template<typename Type >
  const Type * get () const

  Gets content if it has a certain type.

- template<typename Type >
  Type * get ()

  Non `const` *get().*

Private Attributes

- DataType_ **data_**

10.97.1 Detailed Description

A single-valued container for Single, Vector or Matrix.

10.97.2 Constructor & Destructor Documentation

10.97.2.1 Value ( )

Default constructor sets content to Singles's default.

10.97.2.2 Value (const DataType_ & *data*)

Builds and sets content.

Takes type on 1-level-down on the value hierarchy.

Parameters

data : The content.

10.97.2.3 Value (const Rhs & *rhs*)

Builds and sets content.

Takes type on 2-or-more-levels-down on the value hierarchy.

Parameters

rhs : The content.

If you get an error here about 'keyvaluevalue::Parent<Rhs>', this means that Rhs isn't in a lower level in the hierarchy.

10.97.3 Member Function Documentation

10.97.3.1 Value & operator= (const Rhs & *rhs*)

Assignment operator.

Takes type on 2-or-more-levels-down on the value hierarchy.

Parameters

rhs : The data to be assigned.

Returns

A reference to `this`.

If you get an error here about 'keyvaluevalue::Parent<P>', this means that Rhs isn't in a lower level in the hierarchy.

10.97.3.2 bool operator== (const Value & *rhs*) const

Comparison operator.

10.97.3.3 const Type* get ( ) const

Gets content if it has a certain type.

Parameters

Type : (`template` parameter) The expected content's type.

Returns

A `const` pointer to the content, if it is of type *Type*. Otherwise, a null pointer.

10.97.3.4 Type* get ( )

Non `const` get().

10.98 Variant Class Reference

Single-value multi-type container.

`#include <keyvalue/value/Variant.h>`

Public Types

- typedef boost::variant< Nothing, bool, double, ptime, string > **DataType_**

Public Member Functions

- Variant ()

  Default constructor sets content to Nothing.

- Variant (const DataType_ &data)

  Builds and sets content.

- template<typename Rhs >
  Variant (const Rhs &rhs)

  Builds and sets content.

- Variant (const Single &data)

  Builds and sets content.

- template<typename Rhs >
  Variant & operator= (const Rhs &rhs)

  Assignment operator.

- Variant & operator= (const Single &rhs)

  Assignment operator.

- template<typename ElementType >
  bool is (frontend::LexicalToolKit::IO io) const

  Checks if content is convertible to ElementType.

- template<typename ElementType >
  ElementType get (frontend::LexicalToolKit::IO io) const

  Gets content converted ElementType.

Private Member Functions

- template<typename ElementType >
  const ElementType * get () const

  Gets content.

Private Attributes

- DataType_ **data_**

Friends

- std::ostream & operator<< (std::ostream &os, const Variant &variant)

  ostream operator<<() for Variant.

10.98.1 Detailed Description

Single-value multi-type container. A Variant is a single-value container for one of the following types:

- Nothing;
- `bool`;
- `double`;
- `ptime`;
- `string`.

10.98.2 Constructor & Destructor Documentation

10.98.2.1 Variant ( )

Default constructor sets content to Nothing.

10.98.2.2 Variant (const DataType_ & *data*)

Builds and sets content.

Takes type on 1-level-down on the value hierarchy.

Parameters

data : The content.

10.98.2.3 Variant (const Rhs & *rhs*)

Builds and sets content.

Takes type on 2-or-more-levels-down on the value hierarchy.

Parameters

rhs : The content.

If you get an error here about 'keyvaluevalue::Parent<Rhs>', this means that Rhs isn't in a lower level in the hierarchy.

10.98.2.4 Variant (const Single & *data*)

Builds and sets content.

Takes type on the 1-level-up on the value hierarchy.

Parameters

data : The content.

10.98.3 Member Function Documentation

10.98.3.1 Variant & operator= (const Rhs & *rhs*)

Assignment operator.

Takes type on a 2-or-more-levels-below of the value hierarchy.

Parameters

rhs : The data to be assigned.

Returns

A reference to `this`.

If you get an error here about 'keyvaluevalue::Parent<P>', this means that Rhs isn't in a lower level in the hierarchy.

10.98.3.2 Variant& operator= (const Single & *rhs*)

Assignment operator.

Assigns to the content of a Single object.

Parameters

data : The content.

10.98.3.3 bool is (frontend::LexicalToolKit::IO *io*) const

Checks if content is convertible to *ElementType*.

If *ElementType* and content's type are the same, then this method returns `true`. If not, this method also considers lexical conversions (see documentation of get()) but only partially: Indeed, it returns `true` if the current frontend::FrontEnd enables lexical conversion from content's type to *ElementType*. However, it does not guarantee that the conversion will succeed when performed. For instance, suppose `*this` stores the `string` "foo". Assume also *ElementType* is `double` and conversion from `string` to `double` is enabled by frontend::FrontEnd. Then this method will return `true` but the method get<double>() will (probably) fail because "foo" is not convertible to a `double` (nevertheless, remind that what happens depends on the lexical converter which can do whatever it wants!)

ElementType can be any Variant allowed type. Additionally, this method allows for *ElementType* to be `unsigned int`. In this case, it checks if content type is a `double` whose value is a positive integer.

See get() for more details on lexical conversions.

Parameters

ElementType : (`template` parameter) The type to be checked;

io : Direction for required conversion (see frontend::LexicalToolKit).

Returns

If content's type is convertible to *ElementType*, then this method returns `true`. Otherwise, it returns `false`.

10.98.3.4 ElementType get (frontend::LexicalToolKit::IO *io*) const

Gets content converted *ElementType*.

Debug build firstly checks whether content is convertible to *ElementType* (through is<ElementType>()). If not, an exception is thrown. Release build does not check and has undefined behavior if content is not convertible to *ElementType*.

Conversion follows these steps: First it checks if content's type and *ElementType* coincide, in which case, it returns a copy of content. Otherwise, it calls enabled and meaningful frontend::FrontEnd lexical converter functions (see frontend::FrontEnd and frontend::LexicalToolKit) in the following order:

- from `double` to *ElementType*;
- from `string` to *ElementType*;
- from `bool` to *ElementType*;
- from `ptime` to *ElementType*.

ElementType can be any Variant allowed type. Additionally, this method allows for the special *ElementType* `unsigned int`. In this case, it checks if content type is a `double` whose value is a positive integer.

Parameters

ElementType : (`template` parameter) The target type;
io : Direction of required conversion (see frontend::LexicalToolKit).

Returns

The converted content if conversion succeeds.

Exceptions

LogicError : (Debug built only) if content is not convertible to *ElementType*.
frontend::LexicalToolKit::Failure& : If conversion fails.

10.98.3.5 const ElementType* get ( ) const [private]

Gets content.

This `private` low-level method which is a helper to `public` methods is() and get(). No lexical conversions are considered.

Parameters

ElementType : (`template` parameter) Expected content's type.

Returns

A `const` pointer to the content, if it is of type *ElementType*. Otherwise, a null pointer.

10.98.4 Friends And Related Function Documentation

10.98.4.1 std::ostream& operator<< (std::ostream & *os*, const Variant & *variant*) [friend]

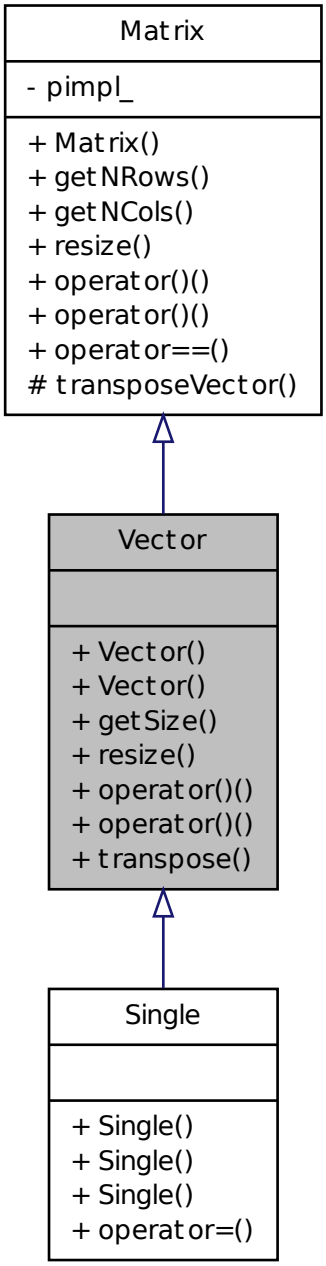
ostream operator<<() for Variant.

10.99 Vector Class Reference

A m x 1 or 1 x n Matrix.

`#include <keyvalue/value/Vector.h>`

Inheritance diagram for Vector:



Public Member Functions

- Vector (size_t size=1, bool isRow=false)

Constructs a Vector with a given size.

- template<typename ElementType >
  Vector (const vector< ElementType > &data, bool isRow=false)

  Builds a Vector from a `std::vector<ElementType>`

- size_t getSize () const

  Gets Vector size.

- void resize (size_t size)

  Resizes the vector.

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

  Provides access to the i-th element.

- Variant & operator() (size_t i)

  Non `const` operator()().

- void transpose ()

  Transposes the vector.

- size_t getNRows () const

  Gets number of rows.

- size_t getNCols () const

  Gets number of columns.

- void resize (size_t nRows, size_t nCols)

  Resizes the matrix.

- const Variant & operator() (size_t i, size_t j) const

  Provides access to the element in the i-th row and j-th column.

- Variant & operator() (size_t i, size_t j)

  Non `const` operator()().

- bool operator== (const Matrix &rhs) const

  Comparison operator.

Protected Member Functions

- void transposeVector ()

  Transposes the matrix if one of its dimensions is 1.

10.99.1 Detailed Description

A m x 1 or 1 x n Matrix. See Matrix documentation for more details.

10.99.2 Constructor & Destructor Documentation

10.99.2.1 Vector (size_t *size* = *1*, bool *isRow* = *false*) [explicit]

Constructs a Vector with a given size.

Debug build throws an exception when size is zero. Release build has undefined behavior in that circumstance.

Parameters

size : Vector size;

isRow : When this parameter is `true` (default), a 1 x *dim* Matrix is built. Otherwise a *dim* x 1 Matrix is built.

Exceptions

LogicError : (Debug build only) When *dim* = 0.

10.99.2.2 Vector (const vector< ElementType > & *data*, bool *isRow* = *false*)

Builds a Vector from a `std::vector<`*ElementType*`>`

ElementType must be one of the following types,

- `bool`;
- `double`;
- `ptime`;
- `string`.

Parameters

ElementType : (`template` parameter) The input vector element type;

data : A vector defining the content.

isRow : When this parameter is `true` (default), a 1 x *dim* Matrix is built. Otherwise a *dim* x 1 Matrix is built.

10.99.3 Member Function Documentation

10.99.3.1 size_t getSize ( ) const

Gets Vector size.

Returns

The size.

10.99.3.2 void resize (size_t *size*)

Resizes the vector.

The size of a vector can not increase. Debug build check for this and an exception is thrown when the test fails. Release build has undefined behavior in this case.

Parameters

size : New size.

Exceptions

LogicError : (Debug build only) If the new dimension is greater than the original one.

10.99.3.3 const Variant& operator() (size_t *i*) const

Provides access to the *i-th* element.

Debug build performs bound check and throws an exception when the check fail. Release build has undefined behavior in that circumstance.

Parameters

i : Element's index.

Returns

A `const` reference to the *i-th* element.

Exceptions

LogicError : (Debug build only) When *i* is not smaller than vector's dimension.

10.99.3.4 Variant& operator() (size_t *i*)

Non `const` operator()().

10.99.3.5 void transpose ( )

Transposes the vector.

10.99.3.6 size_t getNRows ( ) const [inherited]

Gets number of rows.

Returns

The number of rows.

10.99.3.7 size_t getNCols ( ) const [inherited]

Gets number of columns.

Returns

The number of columns.

10.99.3.8 void resize (size_t *nRows*, size_t *nCols*) [inherited]

Resizes the matrix.

The size of a matrix (*i.e.*, number of rows times number of columns) can not increase. Debug build check for this and an exception is thrown when the test fails. Release build has undefined behavior in this case.

The contents of a matrix after resizing is undefined.

Parameters

nRows : number of rows;

nCols : number of columns.

Exceptions

LogicError : (Debug build only) If the new size is greater than the original one.

10.99.3.9 const Variant& operator() (size_t *i*, size_t *j*) const [inherited]

Provides access to the element in the *i-th* row and *j-th* column.

Debug build perform bound checks and throws an exception on failure. Release build has undefined behavior in that case.

Parameters

i : Row index;

j : Column index.

Returns

A `const` reference to *(i, j)*-th element.

Exceptions

LogicError : (Debug build only) If *i* is not smaller than the number of rows or *j* is not smaller than the number of columns.

10.99.3.10 Variant& operator() (size_t *i*, size_t *j*) [inherited]

Non `const` operator()().

10.99.3.11 bool operator== (const Matrix & *rhs*) const [inherited]

Comparison operator.

Recall that Matrix has reference semantics. Consider the code:

```
Matrix m1(2,2);
m1(0,0) = m1(0,1) = m1(1,0) = m1(1,1) = 1.0;
Matrix m2(2,2);
m2(0,0) = m2(0,1) = m2(1,0) = m2(1,1) = 1.0;
Matrix m3(m1);
```

Then `m1 == m2` is `false` and `m1 == m3` is `true`.

Parameters

rhs : The Value to be compared against.

Returns

This method returns `true` if *this and *rhs* refer to the same data. Otherwise it returns `false`.

10.99.3.12 void transposeVector ( ) [protected, inherited]

Transposes the matrix if one of its dimensions is 1.

In-place matrix transposition is a complicated stuff. See

http://en.wikipedia.org/wiki/In-place_matrix_transposition

However, since the storage is linear, the memory layouts of n x 1 and 1 x n matrix, are the same. Hence, we only need to swap dimensions. This method is intent to perform only this simple transposition and it is protected to be called by Vector which derives from this class.

Debug build checks if either of the dimensions is 1 and throws an exception if the test fails. Release build has undefined behaviour in that circumstance.

Exceptions

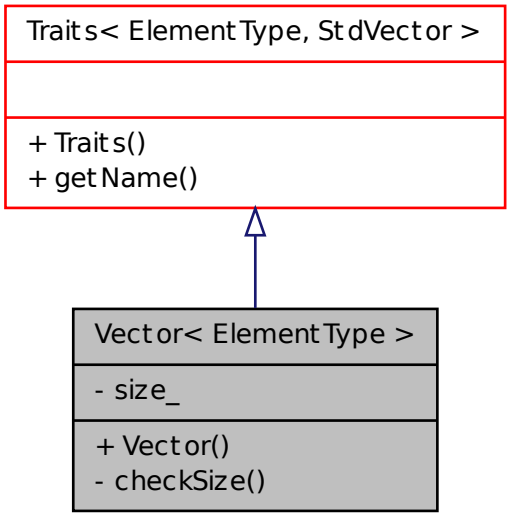
LogicError : (Debug build only) If none of dimensions is 1.

10.100 Vector< ElementType > Class Template Reference

Generic key whose converter type is StdVector.

`#include <keyvalue/key/generic/Vector.h>`

Inheritance diagram for Vector< ElementType >:



Public Types

- enum
- typedef Traits< ElementType, StdVector, Default > **Traits_**
- typedef StdVector< typename Default< ElementType >::OutputType_ > **ConverterType_**
- typedef ConverterType_::InputType_ **InputType_**
- typedef ConverterType_::OutputType_ **OutputType_**

Public Member Functions

- Vector (const string &name, size_t size=0)

  Constructs the key and sets its name and the expected size of the StdVector content.

- string getName () const

  Gets Key's name.

- string getName () const

  Gets name.

- void setName (const string &name)

  Sets name.

Private Member Functions

- void checkSize (size_t size) const

  Checks if size matches expected size.

Private Attributes

- size_t **size_**

10.100.1 Detailed Description

template<typename ElementType> class keyvalue::key::Vector< ElementType >

Generic key whose converter type is StdVector. On construction, the key name and the expected size of the StdVector content are fixed. (Recall that StdVector holds a std::vector.) At the time the value for this key is requested, the size of StdVector content and its expected size will be compared. If they do not match, then an exception::RuntimeError will be thrown.

Parameters

ElementType : (`template` parameter) Vector element type.

10.100.2 Constructor & Destructor Documentation

10.100.2.1 Vector (const string & *name*, size_t *size* = *0*) [explicit]

Constructs the key and sets its name and the expected size of the StdVector content.

Parameters

name : The key name;

size : Expected size. By default, *size* = 0 which means there is no expected size, i.e., any stricktly positive size for the StdVector content will be accepted.

10.100.3 Member Function Documentation

10.100.3.1 void checkSize (size_t *size*) const [private]

Checks if size matches expected size.

Parameters

size : The size.

RuntimeError : If *size* does not match expected size.

10.100.3.2 string getName ( ) const [inherited]

Gets Key's name.

Returns

The Key's name.

10.100.3.3 string getName ( ) const [inherited]

Gets name.

Returns

The name.

10.100.3.4 void setName (const string & *name*) [inherited]

Sets name.

Parameters

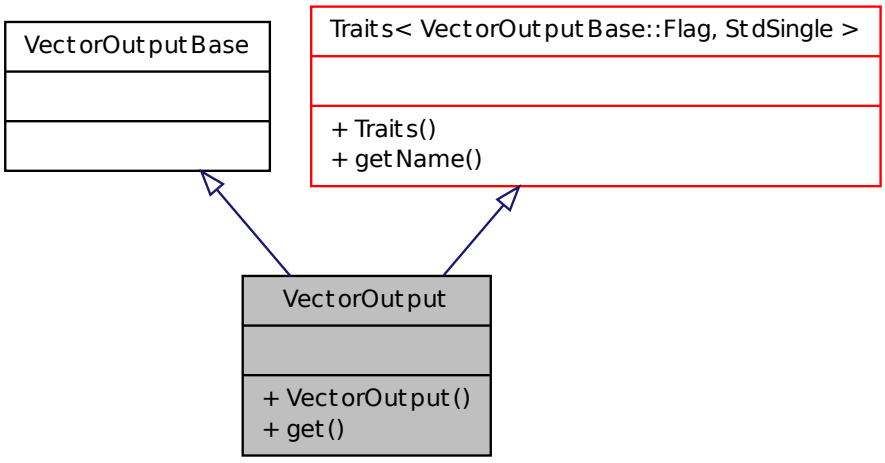
name : The name.

10.101 VectorOutput Class Reference

VectorOutput key.

`#include <keyvalue/key/specific/VectorOutput.h>`

Inheritance diagram for VectorOutput:



Public Types

- enum Flag { **AsIs**, **Row**, **Column** }

  Defines the possible values for VectorOutput.

- enum
- typedef Traits< VectorOutputBase::Flag, StdSingle, Default > **Traits_**
- typedef StdSingle< typename Default< VectorOutputBase::Flag >::OutputType_ > **ConverterType_**
- typedef ConverterType_::InputType_ **InputType_**
- typedef ConverterType_::OutputType_ **OutputType_**

Public Member Functions

- Flag get (const string &name) const

  Partially maps names into flags.

- string getName () const

  Gets Key's name.

- string getName () const

  Gets name.

- void setName (const string &name)

  Sets name.

10.101.1 Detailed Description

VectorOutput key. Key's name is "VectorOutput".

10.101.2 Member Enumeration Documentation

10.101.2.1 enum Flag `[inherited]`

Defines the possible values for VectorOutput.

10.101.3 Member Function Documentation

10.101.3.1 Flag get (const string & *name*) const

Partially maps names into flags.

Maps "AsIs", "Row", and "Column", resp., to *AsIs*, *Row* and *Column*.

Parameters

name : Name to be mapped;

Returns

Flag corresponding to name.

Exceptions

keyvalue::RuntimeError : If *name* is unknown.

10.101.3.2 string getName ( ) const [inherited]

Gets Key's name.

Returns

The Key's name.

10.101.3.3 string getName ( ) const [inherited]

Gets name.

Returns

The name.

10.101.3.4 void setName (const string & *name*) [inherited]

Sets name.

Parameters

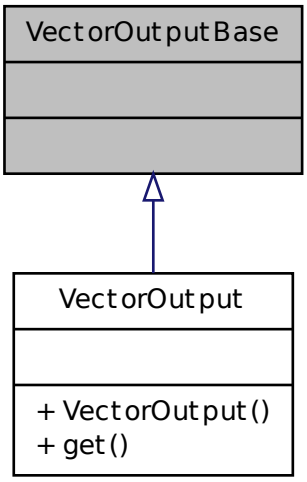
name : The name.

10.102 VectorOutputBase Struct Reference

VectorOutput's base `class`.

`#include <keyvalue/key/specific/VectorOutput.h>`

Inheritance diagram for VectorOutputBase:



Public Types

- enum Flag { **AsIs**, **Row**, **Column** }

  Defines the possible values for VectorOutput.

10.102.1 Detailed Description

VectorOutput's base `class`. The sole purpose of this `class` is to define `enum Flag` which is inherited by VectorOutput.

10.102.2 Member Enumeration Documentation

10.102.2.1 enum Flag

Defines the possible values for VectorOutput.

10.103 Warning Class Reference

MessageImpl instantiation used for warning messages.

`#include <keyvalue/sys/message/MessageImpl.h>`

10.103.1 Detailed Description

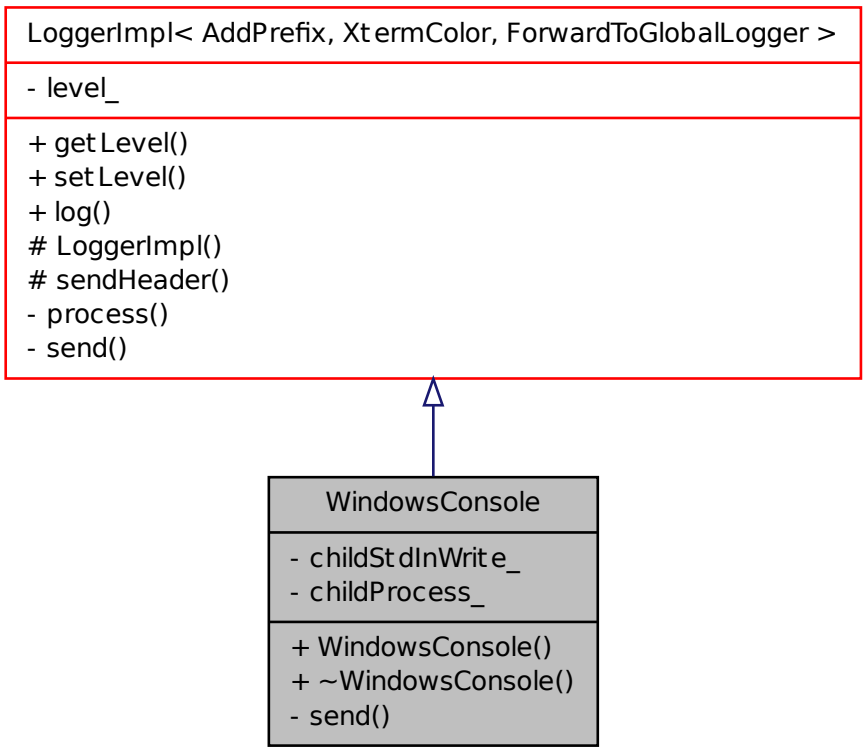
MessageImpl instantiation used for warning messages. `typedef MessageImpl<2> Warning;`

10.104 WindowsConsole Class Reference

This `class` implements a Windows console Logger.

`#include <keyvalue/sys/logger/WindowsConsole.h>`

Inheritance diagram for WindowsConsole:



Public Types

- typedef LoggerImpl< keyvalue::logger::AddPrefix, keyvalue::logger::XtermColor, keyvalue::logger::ForwardToGlobalLogger > **MyBase_**
- typedef LoggerImpl< AddPrefix, XtermColor, ForwardToGlobalLogger > **LoggerImpl_**
- enum **Device** { **Console**, **File**, **Standard** }

Public Member Functions

- **WindowsConsole** (unsigned int level, const string &title, const string &path)
- unsigned int **getLevel** () const
- virtual unsigned int getLevel () const =0

  Gets Logger's current level.

- void **setLevel** (unsigned int level)
- virtual void setLevel (unsigned int level)=0

  Sets Logger's current level.

- bool **log** (const Message &message)
- virtual bool log (const Message &message)=0

  Logs a Message.

Protected Member Functions

- bool sendHeader ()

  Sends the header message.

- virtual bool **send** (const string &message)=0
- virtual bool **send** (const string &message)=0

Private Member Functions

- bool send (const string &message)

  Sends a raw `string` message to the Logger's underlying device.

Private Attributes

- HANDLE **childStdInWrite_**
- PROCESS_INFORMATION **childProcess_**

10.104.1 Detailed Description

This `class` implements a Windows console Logger.

10.104.2 Member Function Documentation

10.104.2.1 bool send (const string & *message*) [private, virtual]

Sends a raw `string` message to the Logger's underlying device.

Parameters

message : Message to be sent.

Returns

This method returns `false` if it fails. Otherwise, it returns `true`.

Implements LoggerImpl< AddPrefix, XtermColor, ForwardToGlobalLogger >.

10.104.2.2 virtual unsigned int getLevel ( ) const [pure virtual, inherited]

Gets Logger's current level.

Returns

The level.

Implemented in LoggerImpl< PrefixPolicy, ColorPolicy, SafetyPolicy >.

10.104.2.3 virtual void setLevel (unsigned int *level*) [pure virtual, inherited]

Sets Logger's current level.

Parameters

level : New level.

Implemented in LoggerImpl< PrefixPolicy, ColorPolicy, SafetyPolicy >.

10.104.2.4 virtual bool log (const Message & *message*) [pure virtual, inherited]

Logs a Message.

This method may fail if the implemented Logger cannot process the message (e.g. a console Logger that has no longer a console window).

Parameters

message : Message to be logged.

Returns

This method returns `true` if it is successful.

Implemented in LoggerImpl< PrefixPolicy, ColorPolicy, SafetyPolicy >.

10.104.2.5 bool sendHeader ( ) [protected, inherited]

Sends the header message.

This method sends (throw send()) an opening header message to the underlying device of the Logger.

Returns

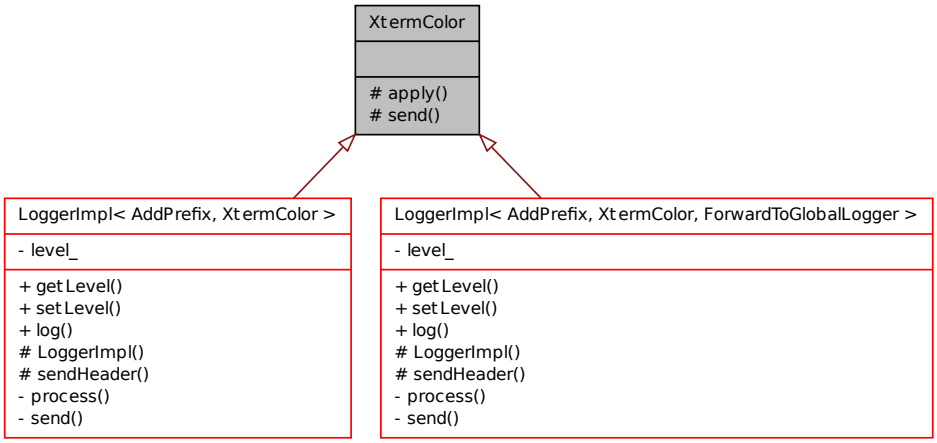
This method returns what print() does.

10.105 XtermColor Class Reference

Defines a policy for Message color, which is, to set the underlying device of the Logger to print in Message's color.

`#include <keyvalue/sys/logger/policy/XtermColor.h>`

Inheritance diagram for XtermColor:



Protected Member Functions

- bool **apply** (const Message &message)
- virtual bool **send** (const string &message)=0

10.105.1 Detailed Description

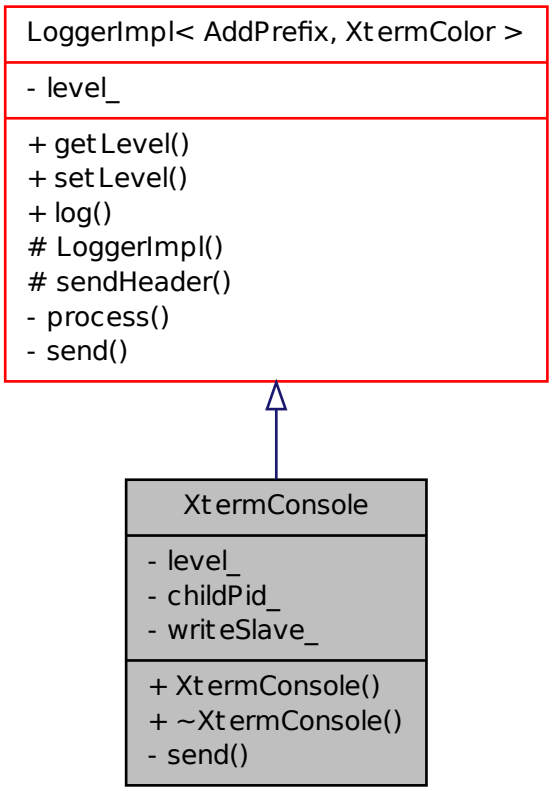
Defines a policy for Message color, which is, to set the underlying device of the Logger to print in Message's color.

10.106 XtermConsole Class Reference

Implements an xterm console Logger.

`#include <keyvalue/sys/logger/XtermConsole.h>`

Inheritance diagram for XtermConsole:



Classes

- class FileRaii

  XtermConsole's helper `class` for file management.

Public Types

- typedef LoggerImpl< AddPrefix, XtermColor, ForwardToGlobalLogger > **LoggerImpl_**
- enum **Device** { **Console**, **File**, **Standard** }

Public Member Functions

- **XtermConsole** (unsigned int level, const string &title)
- unsigned int **getLevel** () const
- virtual unsigned int getLevel () const =0

Gets Logger's current level.

- void **setLevel** (unsigned int level)
- virtual void setLevel (unsigned int level)=0

  Sets Logger's current level.

- bool **log** (const Message &message)
- virtual bool log (const Message &message)=0

  Logs a Message.

Protected Member Functions

- bool sendHeader ()

  Sends the header message.

- virtual bool **send** (const string &message)=0
- virtual bool **send** (const string &message)=0

Private Member Functions

- bool send (const string &message)

  Sends a raw `string` message to the Logger's underlying device.

Private Attributes

- unsigned int **level_**
- pid_t **childPid_**
- FileRaii **writeSlave_**

10.106.1 Detailed Description

Implements an xterm console Logger.

10.106.2 Member Function Documentation

10.106.2.1 bool send (const string & *message*) [private, virtual]

Sends a raw `string` message to the Logger's underlying device.

Parameters

message : Message to be sent.

Returns

This method returns `false` if it fails. Otherwise, it returns `true`.

Implements LoggerImpl< AddPrefix, XtermColor >.

10.106.2.2 virtual unsigned int getLevel ( ) const [pure virtual, inherited]

Gets Logger's current level.

Returns

The level.

Implemented in LoggerImpl< PrefixPolicy, ColorPolicy, SafetyPolicy >.

10.106.2.3 virtual void setLevel (unsigned int *level*) [pure virtual, inherited]

Sets Logger's current level.

Parameters

level : New level.

Implemented in LoggerImpl< PrefixPolicy, ColorPolicy, SafetyPolicy >.

10.106.2.4 virtual bool log (const Message & *message*) [pure virtual, inherited]

Logs a Message.

This method may fail if the implemented Logger cannot process the message (e.g. a console Logger that has no longer a console window).

Parameters

message : Message to be logged.

Returns

This method returns `true` if it is successful.

Implemented in LoggerImpl< PrefixPolicy, ColorPolicy, SafetyPolicy >.

10.106.2.5 bool sendHeader ( ) [protected, inherited]

Sends the header message.

This method sends (throw send()) an opening header message to the underlying device of the Logger.

Returns

This method returns what print() does.

Chapter 11

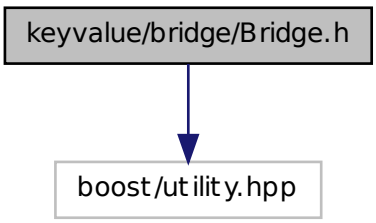
File Documentation

11.1 keyvalue/bridge/Bridge.h File Reference

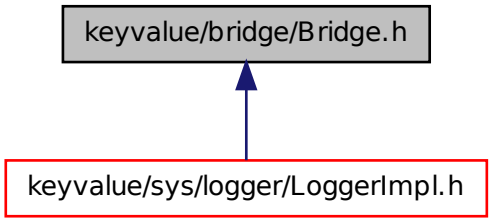
Declaration of `class` Bridge.

```
#include <boost/utility.hpp>
```

Include dependency graph for Bridge.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Bridge

  Bridge's specific data.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

11.1.1 Detailed Description

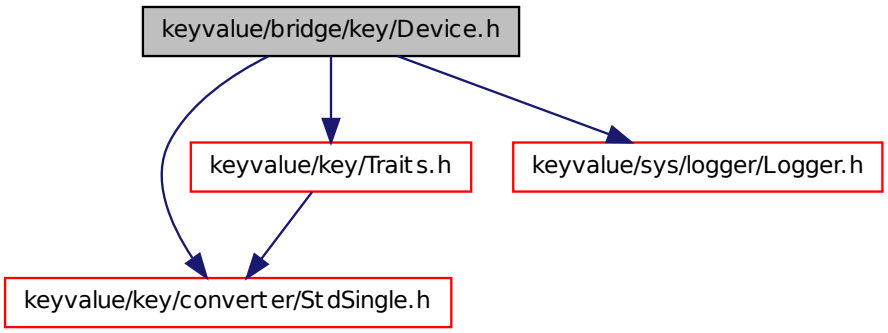
Declaration of `class` Bridge.

11.2 keyvalue/bridge/key/Device.h File Reference

Declaration of `class` Device.

```
#include "keyvalue/key/converter/StdSingle.h"
#include "keyvalue/key/Traits.h"
#include "keyvalue/sys/logger/Logger.h"
```

Include dependency graph for Device.h:



Classes

- class Device

  Device key.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::key

  All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

11.2.1 Detailed Description

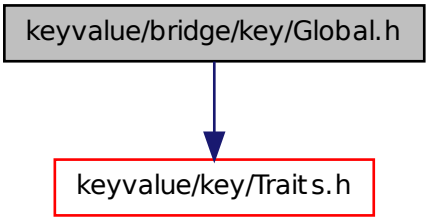
Declaration of `class` Device.

11.3 keyvalue/bridge/key/Global.h File Reference

Declaration of `class` Global.

```
#include "keyvalue/key/Traits.h"
```

Include dependency graph for Global.h:



Classes

- class Global

  Global key.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::key

  All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

11.3.1 Detailed Description

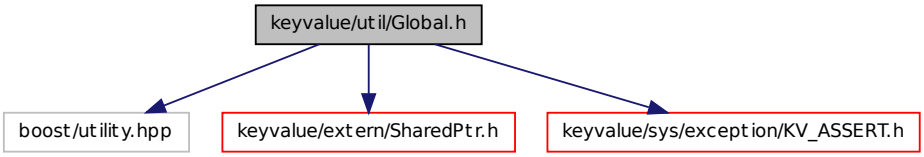
Declaration of `class` Global.

11.4 keyvalue/util/Global.h File Reference

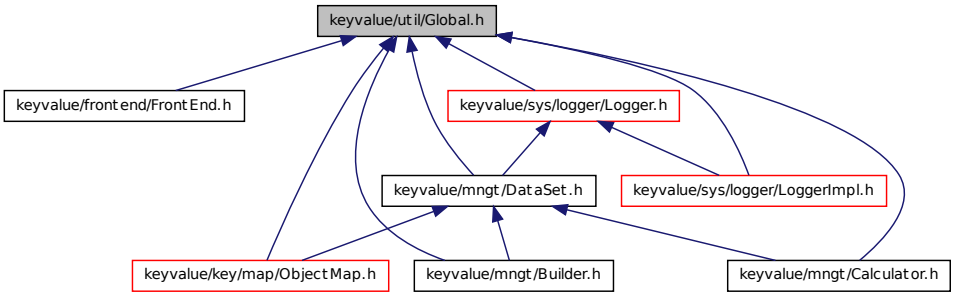
Declaration of `template class` Global.

```
#include <boost/utility.hpp>
#include "keyvalue/extern/SharedPtr.h"
#include "keyvalue/sys/exception/KV_ASSERT.h"
```

Include dependency graph for Global.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Global< ObjectType >

  Manager of global objects.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::util

  Utility `classes` *and functions are member of this* `namespace`.

11.4.1 Detailed Description

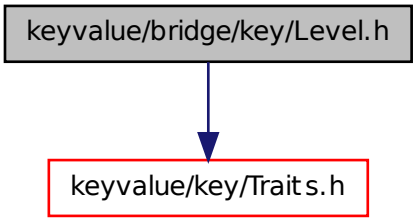
Declaration of `template class` Global.

11.5 keyvalue/bridge/key/Level.h File Reference

Declaration of `class` Level.

`#include "keyvalue/key/Traits.h"`

Include dependency graph for Level.h:



Classes

- class Level

  Level key.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::key

  All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

11.5.1 Detailed Description

Declaration of `class` Level.

11.6 keyvalue/bridge/Register.h File Reference

List of reserved processors.

11.6.1 Detailed Description

List of reserved processors. This file lists all reserved processors that will be registered into keyvalue::ProcessorMngr.

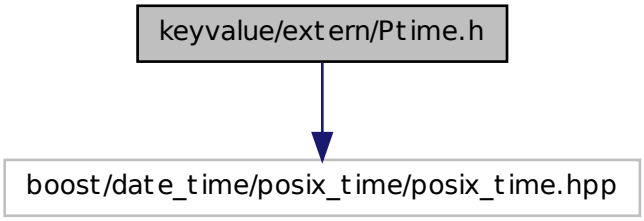
The X Macro technique is used. For an interesting article on X Macros see http://www.drdobbs.com/cpp/184401387;jsessionid=JEHRBGWSWYURNQE1GHPCKHWATMY32JVN

11.7 keyvalue/extern/Ptime.h File Reference

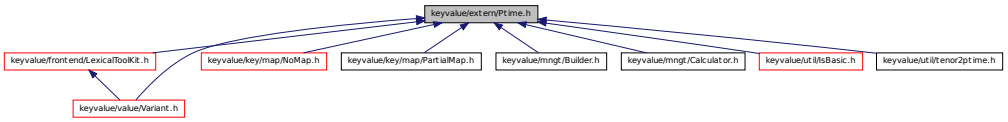
Export `boost::ptime` to `namespace` keyvalue.

```
#include <boost/date_time/posix_time/posix_time.hpp>
```

Include dependency graph for Ptime.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

11.7.1 Detailed Description

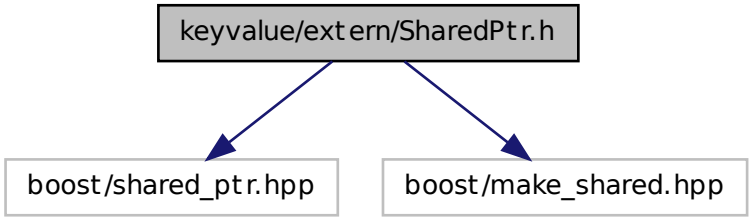
Export `boost::ptime` to `namespace` keyvalue.

11.8 keyvalue/extern/SharedPtr.h File Reference

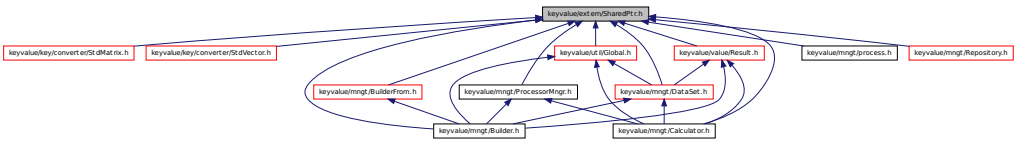
Export `boost::shared_ptr` to `namespace` keyvalue.

```
#include <boost/shared_ptr.hpp>
#include <boost/make_shared.hpp>
```

Include dependency graph for SharedPtr.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

11.8.1 Detailed Description

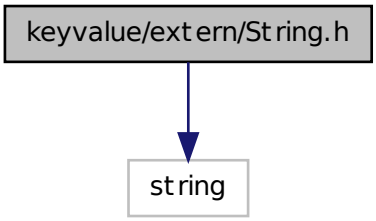
Export `boost::shared_ptr` to `namespace` keyvalue.

11.9 keyvalue/extern/String.h File Reference

Export `std::string` to `namespace` keyvalue.

```
#include <string>
```

Include dependency graph for String.h:



This graph shows which files directly or indirectly include this file:

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

11.9.1 Detailed Description

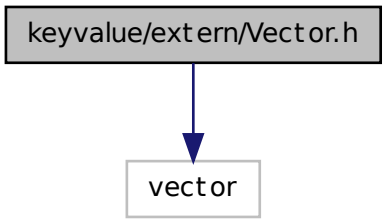
Export `std::string` to `namespace` keyvalue.

11.10 keyvalue/extern/Vector.h File Reference

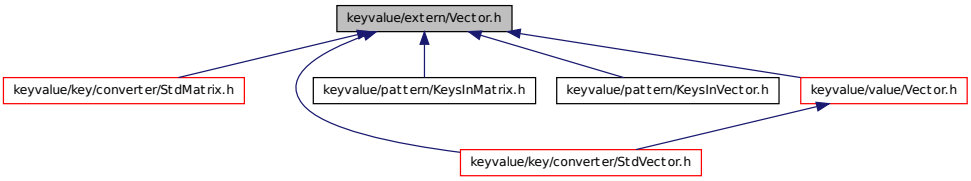
Export `std::vector` to `namespace` keyvalue.

```
#include <vector>
```

Include dependency graph for Vector.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

11.10.1 Detailed Description

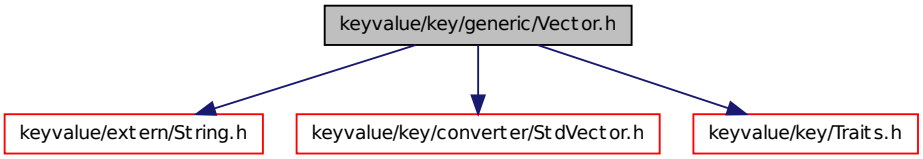
Export `std::vector` to `namespace` keyvalue.

11.11 keyvalue/key/generic/Vector.h File Reference

Declaration and implementation of `template class` Vector (generic key).

```
#include "keyvalue/extern/String.h"
#include "keyvalue/key/converter/StdVector.h"
#include "keyvalue/key/Traits.h"
```

Include dependency graph for Vector.h:



Classes

- class Vector< ElementType >

  Generic key whose converter type is StdVector.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::key

  All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

11.11.1 Detailed Description

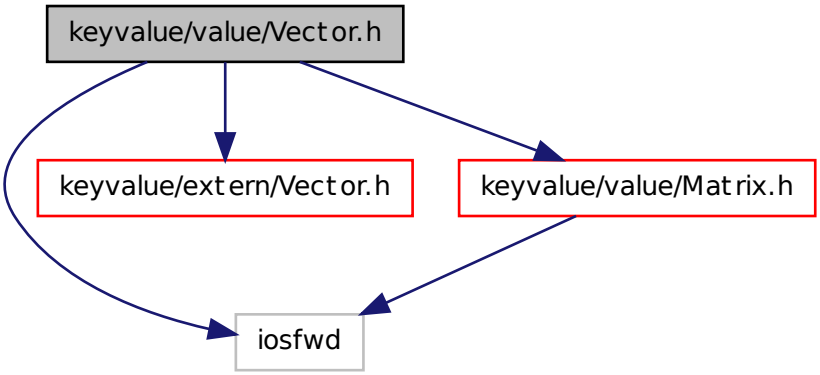
Declaration and implementation of `template class` Vector (generic key).

11.12 keyvalue/value/Vector.h File Reference

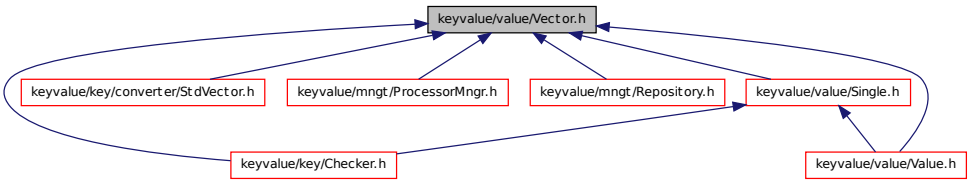
Declaration of `class` Vector (container).

```
#include <iosfwd>
#include "keyvalue/extern/Vector.h"
#include "keyvalue/value/Matrix.h"
```

Include dependency graph for Vector.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Vector

  A m x 1 or 1 x n Matrix.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::value

  All KeyValue containers belong to this namespace.

Functions

- std::ostream & operator<< (std::ostream &os, const Vector &vector)

  ostream operator<<() for Vector.

11.12.1 Detailed Description

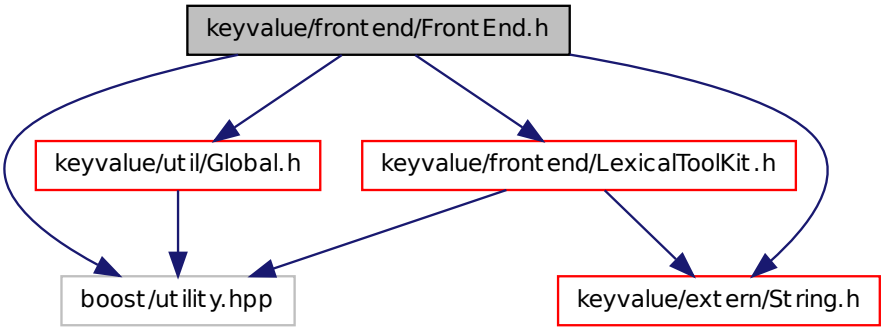
Declaration of `class` Vector (container).

11.13 keyvalue/frontend/FrontEnd.h File Reference

Declaration of `class` FrontEnd.

```
#include <boost/utility.hpp>
#include "keyvalue/extern/String.h"
#include "keyvalue/frontend/LexicalToolKit.h"
#include "keyvalue/util/Global.h"
```

Include dependency graph for FrontEnd.h:



Classes

- class FrontEnd

  Front-end's specific data.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::frontend

  All `classes` and functions needed by frontends belong to this namespace.

- namespace keyvalue::util

  Utility `classes` and functions are member of this `namespace`.

11.13.1 Detailed Description

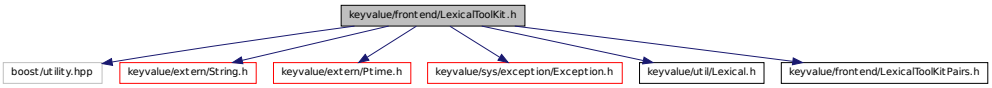
Declaration of `class` FrontEnd.

11.14 keyvalue/frontend/LexicalToolKit.h File Reference

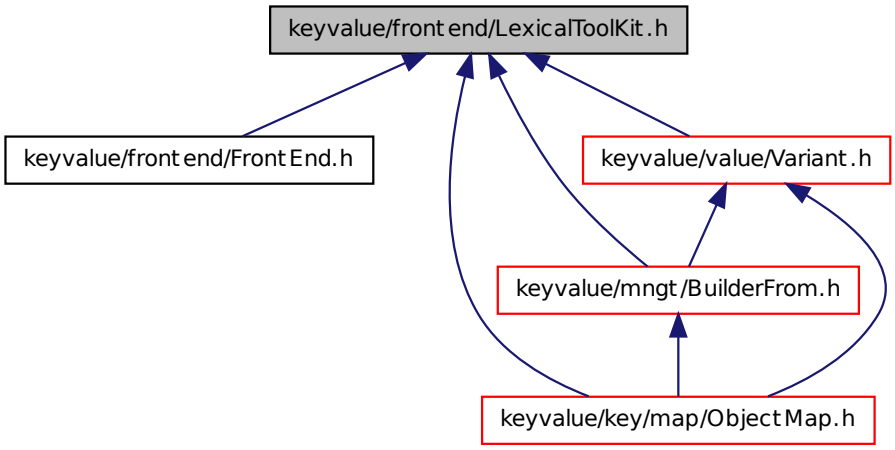
Definition of `class` LexicalToolKit.

```
#include <boost/utility.hpp>
#include "keyvalue/extern/String.h"
#include "keyvalue/extern/Ptime.h"
#include "keyvalue/sys/exception/Exception.h"
#include "keyvalue/util/Lexical.h"
#include "keyvalue/frontend/LexicalToolKitPairs.h"
```

Include dependency graph for LexicalToolKit.h:



This graph shows which files directly or indirectly include this file:



Classes

- class LexicalToolKit

  Manager of all lexical converters needed for front-end input/output.

- struct LexicalToolKit::Helper< From, To >

  Helper `class` of LexicalToolKit.

- class LexicalToolKit::Failure

  This `class` is thrown when LexicalToolKit::convert() fails.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::frontend

  All `classes` and functions needed by frontends belong to this namespace.

Defines

- #define KV_LEXICAL_TOOL_KIT_PAIR(From, To)

11.14.1 Detailed Description

Definition of `class` LexicalToolKit.

11.14.2 Define Documentation

11.14.2.1 #define KV_LEXICAL_TOOL_KIT_PAIR(*From, To*)

Value:

```
Helper<From, To>::Type_ From##2##To##_; \
  IO                    From##2##To##_io_;
```

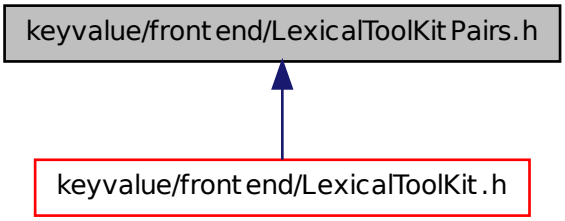
For each pair of types in LexicalToolKitPairs.h, this code declares two LexicalToolKit's members. For instance, for (double, bool) they are

Helper<From, To>::Type_ double2bool_; IO double2bool_io_.

11.15 keyvalue/frontend/LexicalToolKitPairs.h File Reference

X Macro helper for `class` LexicalToolkit.

This graph shows which files directly or indirectly include this file:



11.15.1 Detailed Description

X Macro helper for `class` LexicalToolkit.

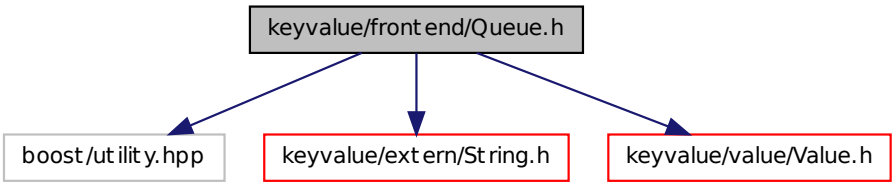
11.16 keyvalue/frontend/Queue.h File Reference

Declaration of `class` Queue.

```
#include <boost/utility.hpp>
#include "keyvalue/extern/String.h"
```

`#include "keyvalue/value/Value.h"`

Include dependency graph for Queue.h:



Classes

- class Queue

  Protocol `class` which defines the Queue interface.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::frontend

  All `classes` and functions needed by frontends belong to this namespace.

11.16.1 Detailed Description

Declaration of `class` Queue.

11.17 keyvalue/key/Checker.h File Reference

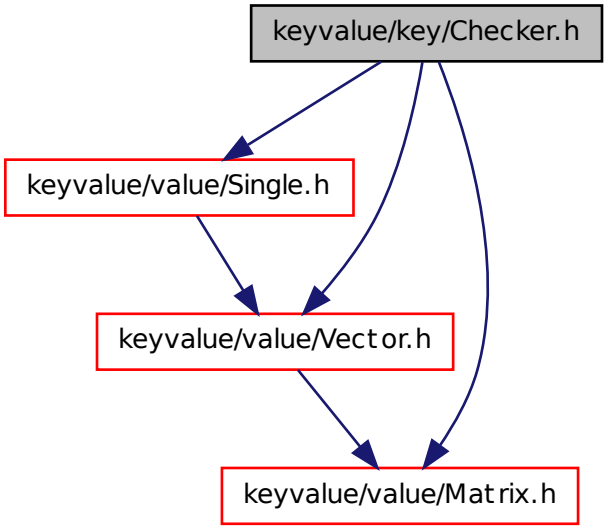
Declaration of `template class` Checker.

`#include "keyvalue/value/Single.h"`

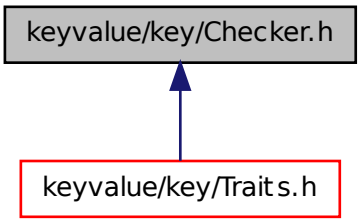
`#include "keyvalue/value/Vector.h"`

`#include "keyvalue/value/Matrix.h"`

Include dependency graph for Checker.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Checker< ConverterType, InputType >

  Primary key checker `template class`.

- class Checker< ConverterType, value::Single >

  Specialization of Checker for InputType equal to value::Single.

- class Checker< ConverterType, value::Vector >

  Specialization of Checker for InputType equal to value::Vector.

- class Checker< ConverterType, value::Matrix >

  Specialization of Checker for InputType equal to value::Matrix.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::key

  All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

11.17.1 Detailed Description

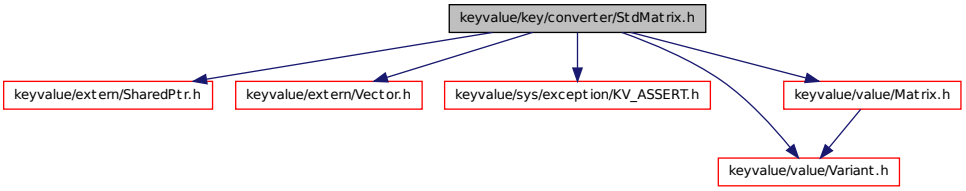
Declaration of `template class` Checker.

11.18 keyvalue/key/converter/StdMatrix.h File Reference

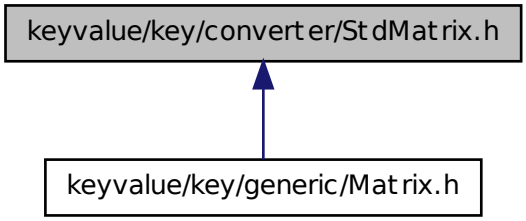
Declaration and implementation of `template class` StdMatrix.

```
#include "keyvalue/extern/SharedPtr.h"
#include "keyvalue/extern/Vector.h"
#include "keyvalue/sys/exception/KV_ASSERT.h"
#include "keyvalue/value/Variant.h"
#include "keyvalue/value/Matrix.h"
```

Include dependency graph for StdMatrix.h:



This graph shows which files directly or indirectly include this file:



Classes

- class StdMatrix< ElementType >

  Converter from value::Matrix to `shared_ptr<std::vector<std::vector<ElementType>>>`.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::key

  All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

11.18.1 Detailed Description

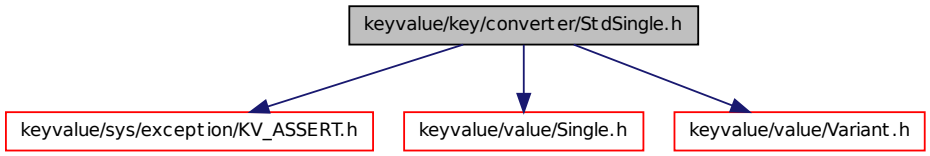
Declaration and implementation of `template class` StdMatrix.

11.19 keyvalue/key/converter/StdSingle.h File Reference

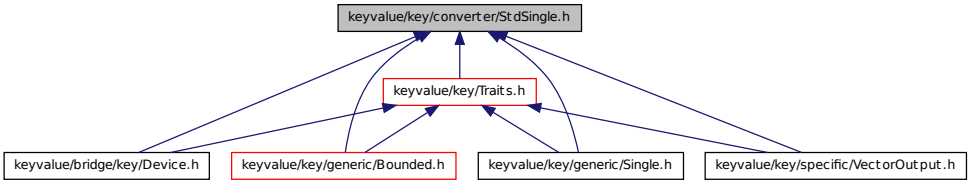
Declaration and implementation of `template class` StdSingle.

```
#include "keyvalue/sys/exception/KV_ASSERT.h"
#include "keyvalue/value/Single.h"
#include "keyvalue/value/Variant.h"
```

Include dependency graph for StdSingle.h:



This graph shows which files directly or indirectly include this file:



Classes

- class StdSingle< ElementType >

  Converter from value::Single to ElementType.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::key

  All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

11.19.1 Detailed Description

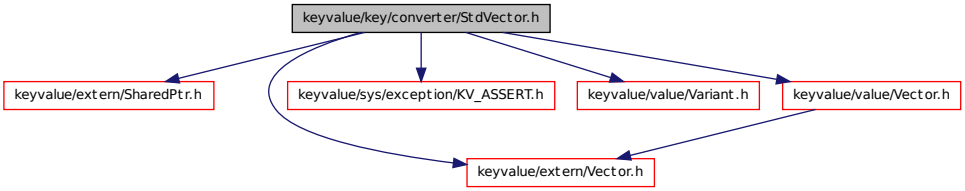
Declaration and implementation of `template class` StdSingle.

11.20 keyvalue/key/converter/StdVector.h File Reference

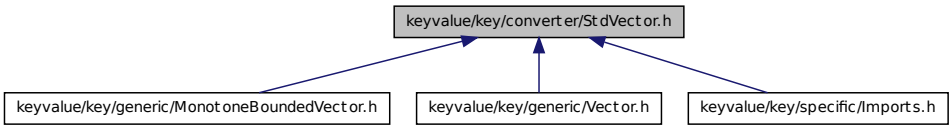
Declaration and implementation of `template class` StdVector.

```
#include "keyvalue/extern/SharedPtr.h"
#include "keyvalue/extern/Vector.h"
#include "keyvalue/sys/exception/KV_ASSERT.h"
#include "keyvalue/value/Variant.h"
#include "keyvalue/value/Vector.h"
```

Include dependency graph for StdVector.h:



This graph shows which files directly or indirectly include this file:



Classes

- class StdVector< ElementType >

  Converter from value::Vector to `shared_ptr<std::vector<ElementType>>`.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::key

  All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

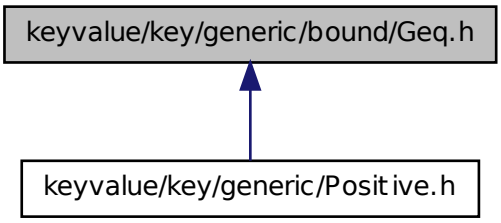
11.20.1 Detailed Description

Declaration and implementation of `template class` StdVector.

11.21 keyvalue/key/generic/bound/Geq.h File Reference

Declaration of `template class` Geq.

This graph shows which files directly or indirectly include this file:



Classes

- class Geq< ElementType >

  Greater-than-or-equal-to bound `class`.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::key

  All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

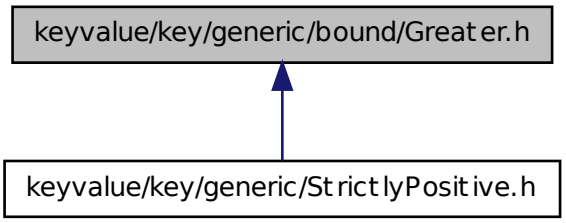
11.21.1 Detailed Description

Declaration of `template class` Geq.

11.22 keyvalue/key/generic/bound/Greater.h File Reference

Declaration of `template class` Greater.

This graph shows which files directly or indirectly include this file:



Classes

- class Greater< ElementType >

  Greater-than bound `class`.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::key

  All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

11.22.1 Detailed Description

Declaration of `template class` Greater.

11.23 keyvalue/key/generic/bound/Leq.h File Reference

Declaration of `template class` Leq.

Classes

- class Leq< ElementType >

  Less-than-or-equal-to bound `class`.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::key

  All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

11.23.1 Detailed Description

Declaration of `template class` Leq.

11.24 keyvalue/key/generic/bound/Less.h File Reference

Declaration of `template class` Less.

Classes

- class Less< ElementType >

  Less-than bound `class`.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::key

  All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

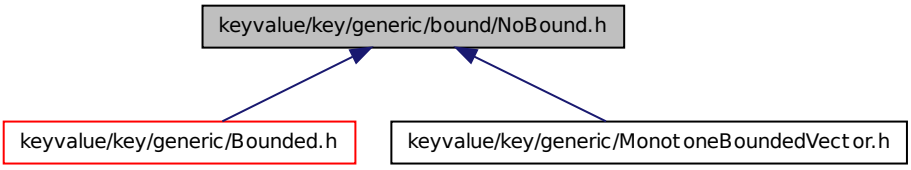
11.24.1 Detailed Description

Declaration of `template class` Less.

11.25 keyvalue/key/generic/bound/NoBound.h File Reference

Declaration of `template class` NoBound.

This graph shows which files directly or indirectly include this file:



Classes

- class NoBound< ElementType >

  NoBound bound `class`.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::key

  All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

11.25.1 Detailed Description

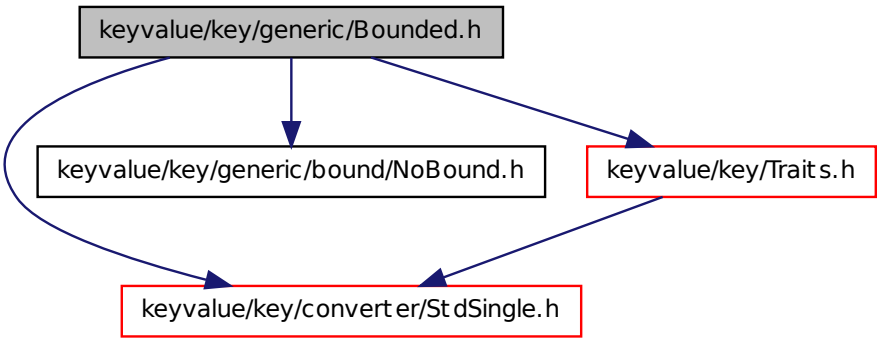
Declaration of `template class` NoBound.

11.26 keyvalue/key/generic/Bounded.h File Reference

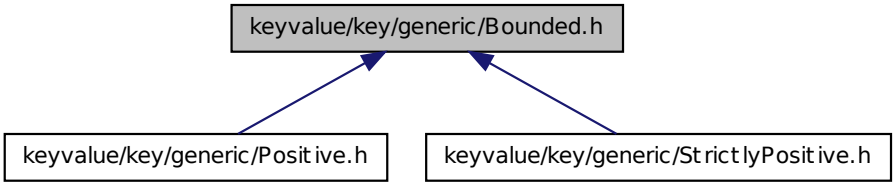
Declaration and implementation of `template class` Bounded.

```
#include "keyvalue/key/converter/StdSingle.h"
#include "keyvalue/key/generic/bound/NoBound.h"
#include "keyvalue/key/Traits.h"
```

Include dependency graph for Bounded.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Bounded< ElementType, Bound1, Bound2 >

  Key for bounded values.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::key

  All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

11.26.1 Detailed Description

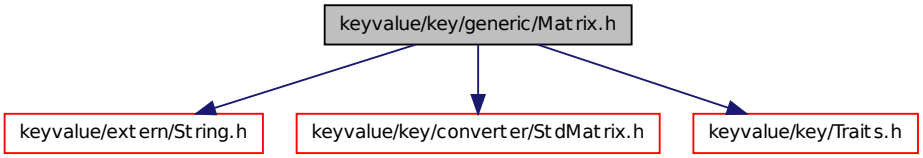
Declaration and implementation of `template class` Bounded.

11.27 keyvalue/key/generic/Matrix.h File Reference

Declaration and implementation of `template class` Matrix (generic key).

```
#include "keyvalue/extern/String.h"
#include "keyvalue/key/converter/StdMatrix.h"
#include "keyvalue/key/Traits.h"
```

Include dependency graph for Matrix.h:



Classes

- class Matrix< ElementType >

  Generic key whose converter type is StdMatrix.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::key

  All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

11.27.1 Detailed Description

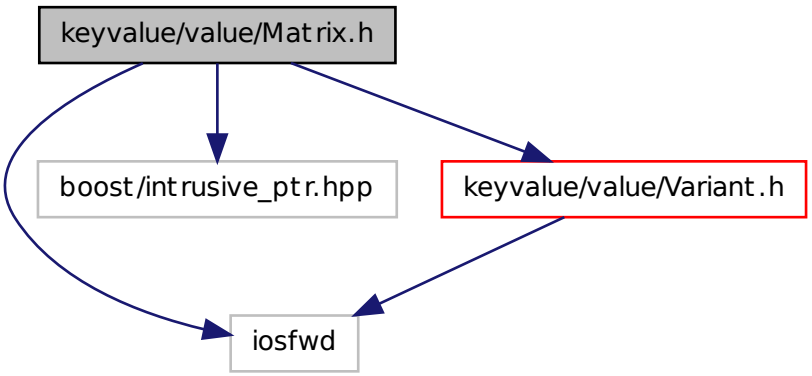
Declaration and implementation of `template class` Matrix (generic key).

11.28 keyvalue/value/Matrix.h File Reference

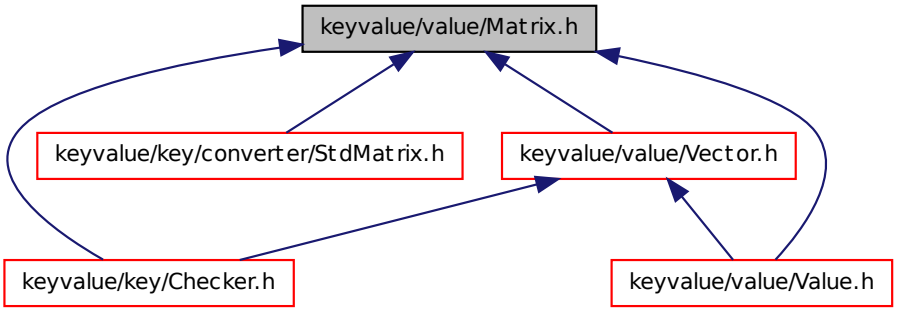
Declaration of `class` Matrix (container).

```
#include <iosfwd>
#include <boost/intrusive_ptr.hpp>
#include "keyvalue/value/Variant.h"
```

Include dependency graph for Matrix.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Matrix

  Matrix of `Variant`s.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::value

  All KeyValue containers belong to this namespace.

Functions

- void intrusive_ptr_add_ref (Matrix::Impl *pimpl)
- void intrusive_ptr_release (Matrix::Impl *pimpl)
- std::ostream & operator<< (std::ostream &os, const Matrix &matrix)

  ostream operator<<() for Matrix.

11.28.1 Detailed Description

Declaration of `class` Matrix (container).

11.29 keyvalue/key/generic/monotone/Decreasing.h File Reference

Declaration of `class` Decreasing.

Classes

- class Decreasing

  Decreasing monotone `class`.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::key

  All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

11.29.1 Detailed Description

Declaration of `class` Decreasing.

11.30 keyvalue/key/generic/monotone/Increasing.h File Reference

Declaration of `class` Increasing.

Classes

- class Increasing

  Increasing monotone `class`.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::key

  All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

11.30.1 Detailed Description

Declaration of `class` Increasing.

11.31 keyvalue/key/generic/monotone/NonMonotone.h File Reference

Declaration of `class` NonMonotone.

Classes

- class NonMonotone

  NonMonotone `class`.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::key

  All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

11.31.1 Detailed Description

Declaration of `class` NonMonotone.

11.32 keyvalue/key/generic/monotone/StrictlyDecreasing.h File Reference

Declaration of `class` StrictlyDecreasing.

Classes

- class StrictlyDecreasing

  StrictlyDecreasing monotone `class`.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::key

  All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

11.32.1 Detailed Description

Declaration of `class` StrictlyDecreasing.

11.33 keyvalue/key/generic/monotone/StrictlyIncreasing.h File Reference

Declaration of `class` StrictlyIncreasing.

Classes

- class StrictlyIncreasing

  StrictlyIncreasing monotone `class`.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::key

  All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

11.33.1 Detailed Description

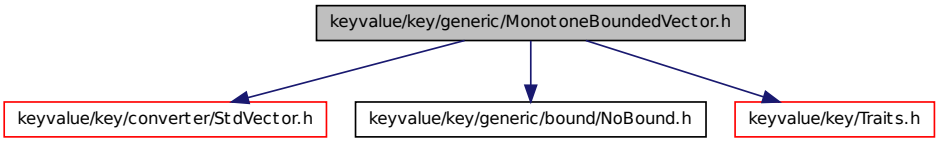
Declaration of `class` StrictlyIncreasing.

11.34 keyvalue/key/generic/MonotoneBoundedVector.h File Reference

Declaration and implementation of `template class` MonotoneBoundedVector.

```
#include "keyvalue/key/converter/StdVector.h"
#include "keyvalue/key/generic/bound/NoBound.h"
#include "keyvalue/key/Traits.h"
```

Include dependency graph for MonotoneBoundedVector.h:



Classes

- class MonotoneBoundedVector< ElementType, Monotone, Bound1, Bound2 >

  Key for monotone bounded vectors.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::key

  All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

11.34.1 Detailed Description

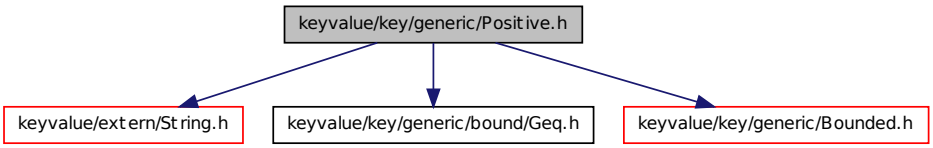
Declaration and implementation of `template class` MonotoneBoundedVector.

11.35 keyvalue/key/generic/Positive.h File Reference

Declaration of `class` Positive.

```
#include "keyvalue/extern/String.h"
#include "keyvalue/key/generic/bound/Geq.h"
#include "keyvalue/key/generic/Bounded.h"
```

Include dependency graph for Positive.h:



Classes

- class Positive

  A general key for positive values.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::key

  All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

11.35.1 Detailed Description

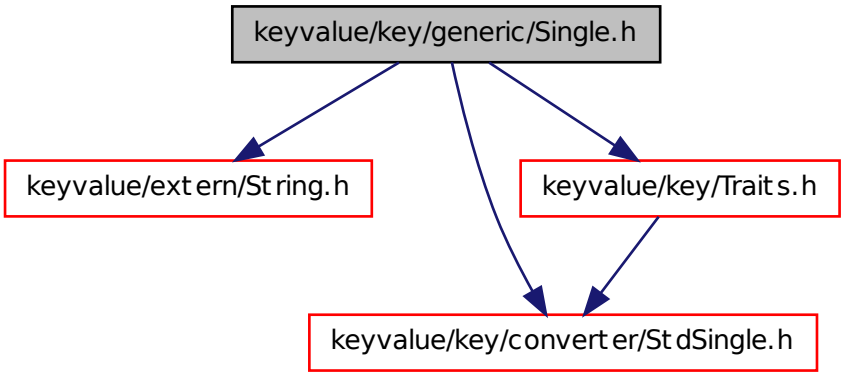
Declaration of `class` Positive.

11.36 keyvalue/key/generic/Single.h File Reference

Declaration and implementation of `template class` Single (generic key).

```
#include "keyvalue/extern/String.h"
#include "keyvalue/key/converter/StdSingle.h"
#include "keyvalue/key/Traits.h"
```

Include dependency graph for Single.h:



Classes

- class Single< ElementType >

  Generic key whose converter type is StdSingle.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::key

  All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

11.36.1 Detailed Description

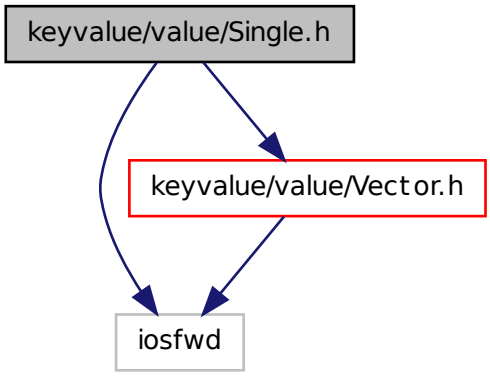
Declaration and implementation of `template class` Single (generic key).

11.37 keyvalue/value/Single.h File Reference

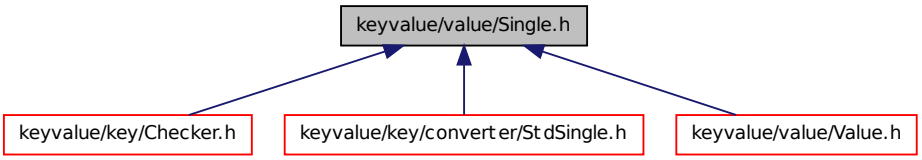
Declaration of `class` Single (container).

```
#include <iosfwd>
#include "keyvalue/value/Vector.h"
```

Include dependency graph for Single.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Single

  A 1 x 1 Matrix.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::value

  All KeyValue containers belong to this namespace.

Functions

- std::ostream & operator<< (std::ostream &os, const Single &rhs)

  ostream operator<<() for Single.

11.37.1 Detailed Description

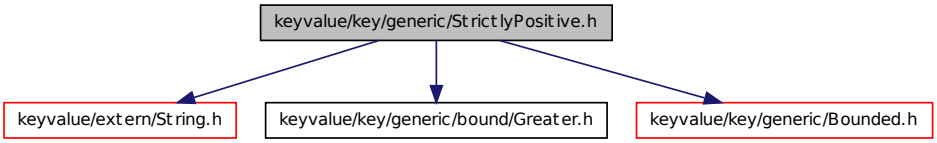
Declaration of `class` Single (container).

11.38 keyvalue/key/generic/StrictlyPositive.h File Reference

Declaration of `class` StrictlyPositive.

```
#include "keyvalue/extern/String.h"
#include "keyvalue/key/generic/bound/Greater.h"
#include "keyvalue/key/generic/Bounded.h"
```

Include dependency graph for StrictlyPositive.h:



Classes

- class StrictlyPositive

  A general key whose value is a strictly positive number.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::key

  All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

11.38.1 Detailed Description

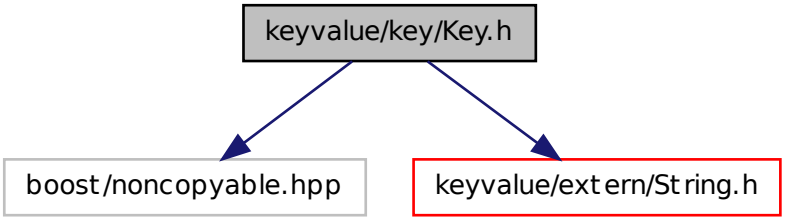
Declaration of `class` StrictlyPositive.

11.39 keyvalue/key/Key.h File Reference

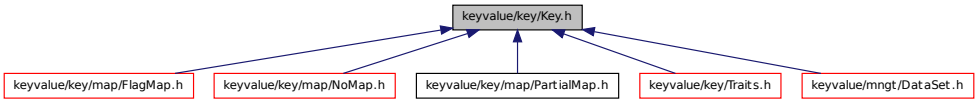
Declaration and implementation of `class` Key.

```
#include <boost/noncopyable.hpp>
#include "keyvalue/extern/String.h"
```

Include dependency graph for Key.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Key

  Base for all real keys.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::key

  All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

11.39.1 Detailed Description

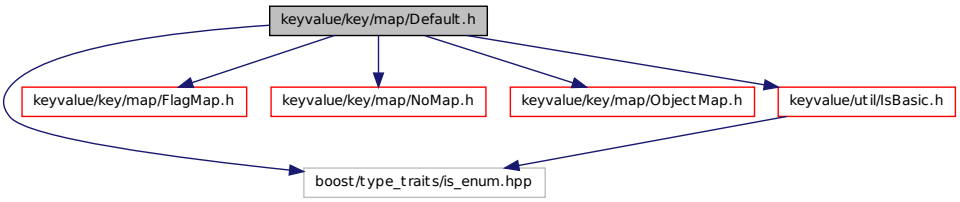
Declaration and implementation of `class` Key.

11.40 keyvalue/key/map/Default.h File Reference

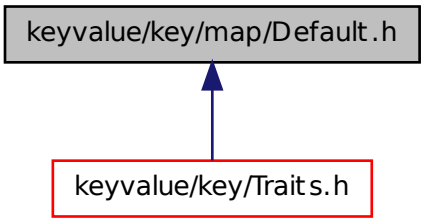
Declaration and implementation of `template class` Default.

```
#include <boost/type_traits/is_enum.hpp>
#include "keyvalue/key/map/FlagMap.h"
#include "keyvalue/key/map/NoMap.h"
#include "keyvalue/key/map/ObjectMap.h"
#include "keyvalue/util/IsBasic.h"
```

Include dependency graph for Default.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct DefaultMap< OutputType, isBasic >

  Meta-function which defines Traits' default map type.

- struct Default< OutputType >

  Meta-function which defines Traits' default map type.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::key

  All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

11.40.1 Detailed Description

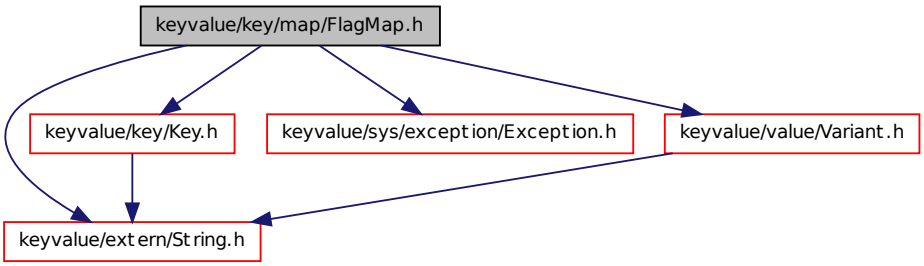
Declaration and implementation of `template class` Default.

11.41 keyvalue/key/map/FlagMap.h File Reference

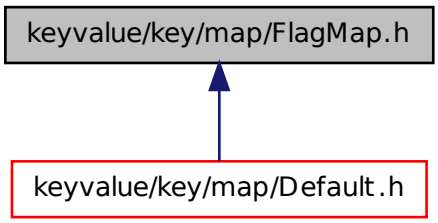
Declaration and implementation of `template class` FlagMap.

```
#include "keyvalue/extern/String.h"
#include "keyvalue/key/Key.h"
#include "keyvalue/sys/exception/Exception.h"
#include "keyvalue/value/Variant.h"
```

Include dependency graph for FlagMap.h:



This graph shows which files directly or indirectly include this file:



Classes

- class FlagMap< OutputType >

  Maps names to constants.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::key

  All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

11.41.1 Detailed Description

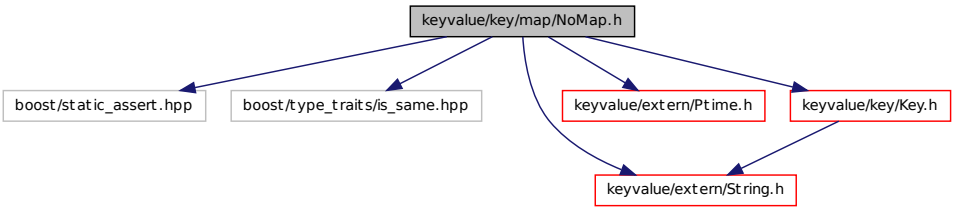
Declaration and implementation of `template class` FlagMap.

11.42 keyvalue/key/map/NoMap.h File Reference

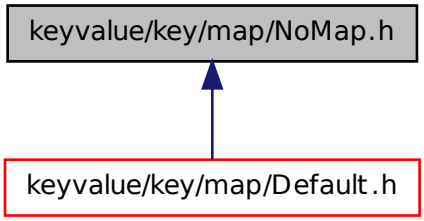
Declaration of `template class` NoMap.

```
#include <boost/static_assert.hpp>
#include <boost/type_traits/is_same.hpp>
#include "keyvalue/extern/String.h"
#include "keyvalue/extern/Ptime.h"
#include "keyvalue/key/Key.h"
```

Include dependency graph for NoMap.h:



This graph shows which files directly or indirectly include this file:



Classes

- class NoMap< OutputType >

  Identity map (aka, NoMap).

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::value

  All KeyValue containers belong to this namespace.

- namespace keyvalue::key

  All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

11.42.1 Detailed Description

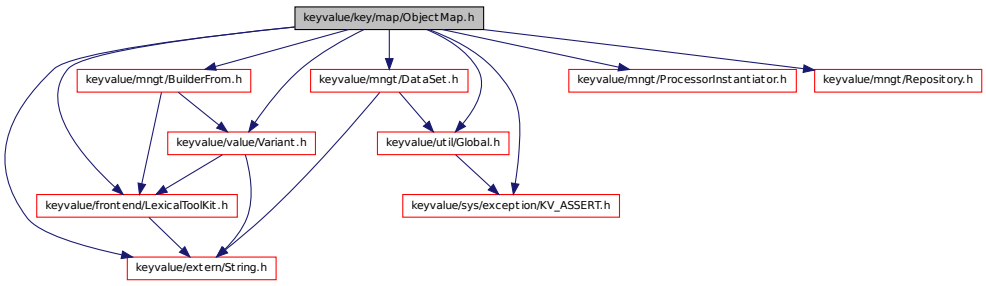
Declaration of `template class` NoMap.

11.43 keyvalue/key/map/ObjectMap.h File Reference

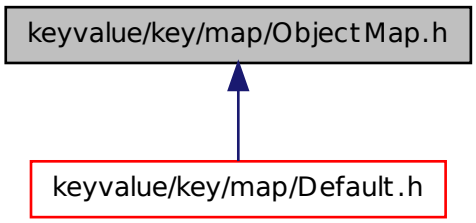
Declaration and implementation of `template class` ObjectMap.

```
#include "keyvalue/extern/String.h"
#include "keyvalue/frontend/LexicalToolKit.h"
#include "keyvalue/mngt/BuilderFrom.h"
#include "keyvalue/mngt/DataSet.h"
#include "keyvalue/mngt/ProcessorInstantiator.h"
#include "keyvalue/mngt/Repository.h"
#include "keyvalue/sys/exception/KV_ASSERT.h"
#include "keyvalue/util/Global.h"
#include "keyvalue/value/Variant.h"
```

Include dependency graph for ObjectMap.h:



This graph shows which files directly or indirectly include this file:



Classes

- class ObjectMap< ObjectType >

  Maps names to objects.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::key

  All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

11.43.1 Detailed Description

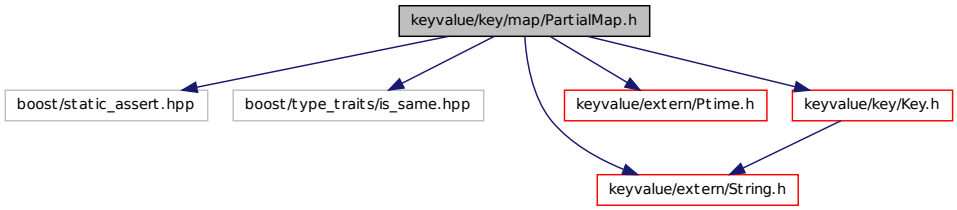
Declaration and implementation of `template class` ObjectMap.

11.44 keyvalue/key/map/PartialMap.h File Reference

Declaration of `template class` PartialMap.

```
#include <boost/static_assert.hpp>
#include <boost/type_traits/is_same.hpp>
#include "keyvalue/extern/String.h"
#include "keyvalue/extern/Ptime.h"
#include "keyvalue/key/Key.h"
```

Include dependency graph for PartialMap.h:



Classes

- class PartialMap< OutputType >

  Partially maps names to constants.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::value

  All KeyValue containers belong to this namespace.

- namespace keyvalue::key

  All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

11.44.1 Detailed Description

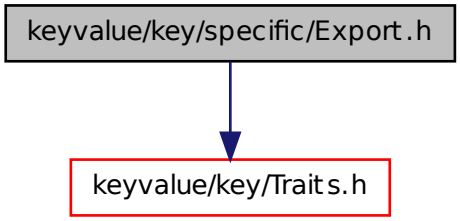
Declaration of `template class` PartialMap.

11.45 keyvalue/key/specific/Export.h File Reference

Declaration of `class` Export.

```
#include "keyvalue/key/Traits.h"
```

Include dependency graph for Export.h:



Classes

- class Export

  Export key.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::key

  All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

11.45.1 Detailed Description

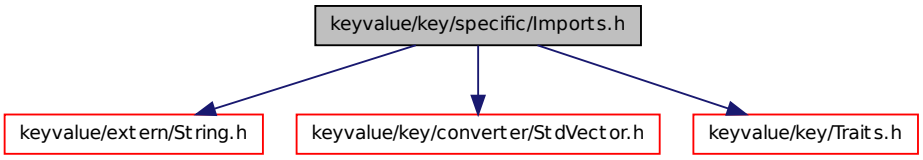
Declaration of `class` Export.

11.46 keyvalue/key/specific/Imports.h File Reference

Declaration of `class` Imports.

```
#include "keyvalue/extern/String.h"
#include "keyvalue/key/converter/StdVector.h"
#include "keyvalue/key/Traits.h"
```

Include dependency graph for Imports.h:



Classes

- class Imports

  Imports key.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::key

  All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

11.46.1 Detailed Description

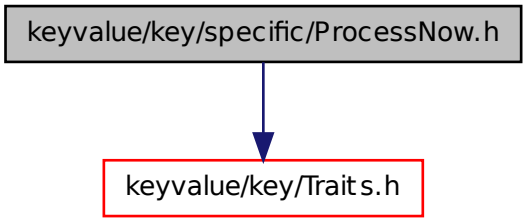
Declaration of `class` Imports.

11.47 keyvalue/key/specific/ProcessNow.h File Reference

Declaration of `class` ProcessNow.

```
#include "keyvalue/key/Traits.h"
```

Include dependency graph for ProcessNow.h:



Classes

- class ProcessNow

  ProcessNow key.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::key

  All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

11.47.1 Detailed Description

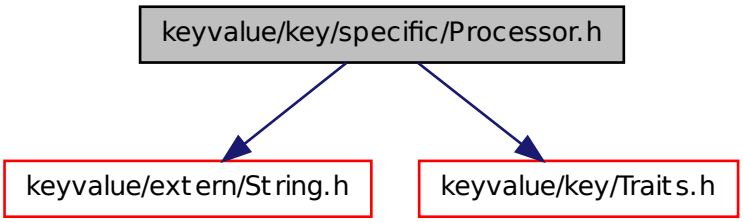
Declaration of `class` ProcessNow.

11.48 keyvalue/key/specific/Processor.h File Reference

Declaration of `class` Processor (key).

```
#include "keyvalue/extern/String.h"
#include "keyvalue/key/Traits.h"
```

Include dependency graph for Processor.h:



Classes

- class Processor

  Processor key.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::key

  All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

11.48.1 Detailed Description

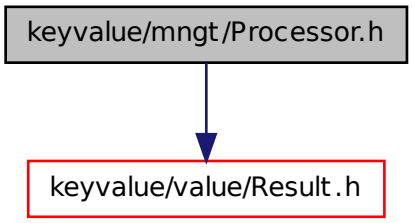
Declaration of `class` Processor (key).

11.49 keyvalue/mngt/Processor.h File Reference

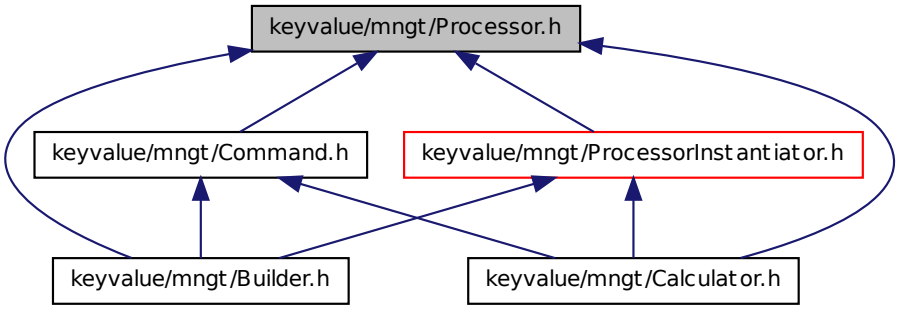
Declaration of `class` Processor.

```
#include "keyvalue/value/Result.h"
```

Include dependency graph for Processor.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Processor

  Protocol `class` which defines Processor interface.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::value

  All KeyValue containers belong to this namespace.

11.49.1 Detailed Description

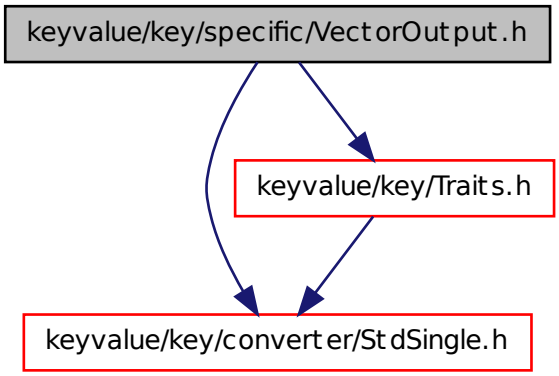
Declaration of `class` Processor.

11.50 keyvalue/key/specific/VectorOutput.h File Reference

Declaration of `class` VectorOutput.

```
#include "keyvalue/key/converter/StdSingle.h"
#include "keyvalue/key/Traits.h"
```

Include dependency graph for VectorOutput.h:



Classes

- struct VectorOutputBase

  VectorOutput's base `class`.

- class VectorOutput

  VectorOutput key.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::key

  All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

11.50.1 Detailed Description

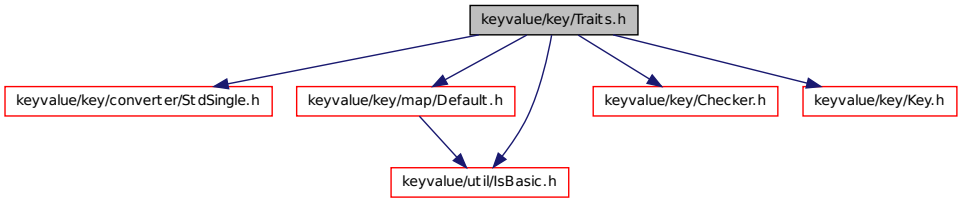
Declaration of `class` VectorOutput.

11.51 keyvalue/key/Traits.h File Reference

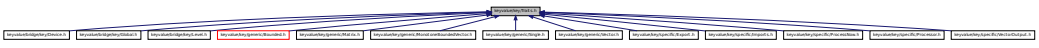
Declaration and implementation of `template class` Traits.

```
#include "keyvalue/key/converter/StdSingle.h"
#include "keyvalue/key/map/Default.h"
#include "keyvalue/key/Checker.h"
#include "keyvalue/key/Key.h"
#include "keyvalue/util/IsBasic.h"
```

Include dependency graph for Traits.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Traits< ElementType, ConverterType, MapType >

  Real keys must derive from adequate instantiations of this `template class`.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::key

  All key funcionalities, including key::Key, key::Traits, key mappings are inside this `namespace`.

11.51.1 Detailed Description

Declaration and implementation of `template class` Traits.

11.52 keyvalue/keyvalue.h File Reference

Main page of *KeyValue*'s reference manual.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

11.52.1 Detailed Description

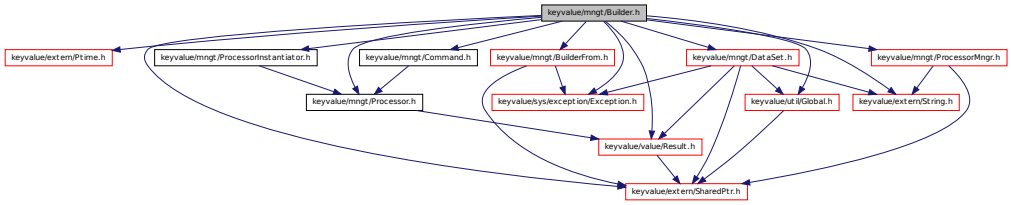
Main page of *KeyValue*'s reference manual.

11.53 keyvalue/mngt/Builder.h File Reference

Declaration of primary `template class` Builder.

```
#include "keyvalue/extern/Ptime.h"
#include "keyvalue/extern/SharedPtr.h"
#include "keyvalue/extern/String.h"
#include "keyvalue/mngt/BuilderFrom.h"
#include "keyvalue/mngt/Command.h"
#include "keyvalue/mngt/DataSet.h"
#include "keyvalue/mngt/Processor.h"
#include "keyvalue/mngt/ProcessorInstantiator.h"
#include "keyvalue/mngt/ProcessorMngr.h"
#include "keyvalue/sys/exception/Exception.h"
#include "keyvalue/util/Global.h"
#include "keyvalue/value/Result.h"
```

Include dependency graph for Builder.h:



Classes

- class Builder< ObjectType >

  Concrete Builder.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::tag

  All Processor tags are member of this `namespace`.

11.53.1 Detailed Description

Declaration of primary `template class` Builder. For details, see keyvalue/mngt/Builder

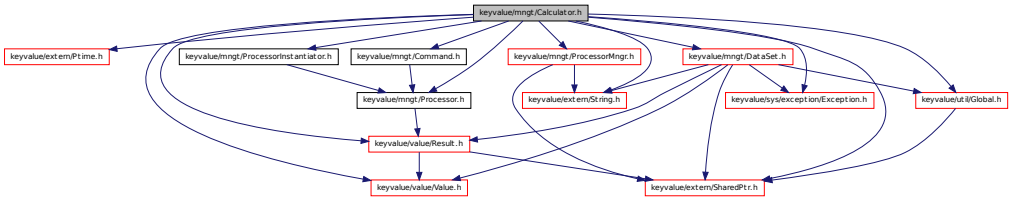
11.54 keyvalue/mngt/Calculator.h File Reference

Declaration of primary `template class` Calculator.

```
#include "keyvalue/extern/Ptime.h"
#include "keyvalue/extern/SharedPtr.h"
#include "keyvalue/extern/String.h"
#include "keyvalue/mngt/Command.h"
#include "keyvalue/mngt/DataSet.h"
#include "keyvalue/mngt/Processor.h"
#include "keyvalue/mngt/ProcessorInstantiator.h"
#include "keyvalue/mngt/ProcessorMngr.h"
#include "keyvalue/sys/exception/Exception.h"
#include "keyvalue/util/Global.h"
```

```
#include "keyvalue/value/Result.h"
#include "keyvalue/value/Value.h"
```

Include dependency graph for Calculator.h:



Classes

- class Calculator< Tag >

  Concrete Calculator.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

11.54.1 Detailed Description

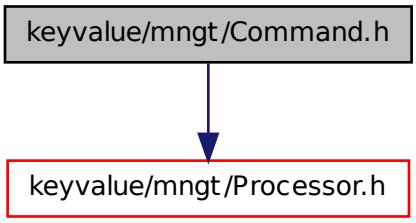
Declaration of primary `template class` Calculator.

11.55 keyvalue/mngt/Command.h File Reference

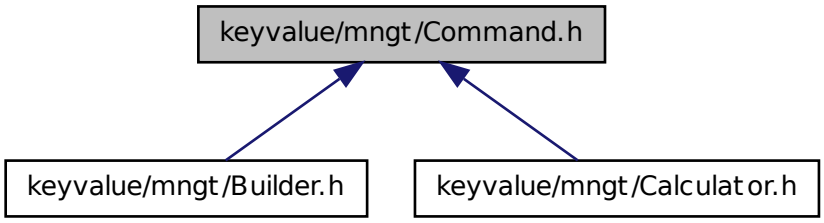
Declaration of `class` Command.

```
#include "keyvalue/mngt/Processor.h"
```

Include dependency graph for Command.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Command

  Command interface.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

11.55.1 Detailed Description

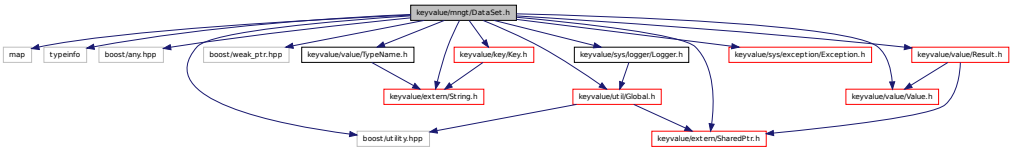
Declaration of `class` Command.

11.56 keyvalue/mngt/DataSet.h File Reference

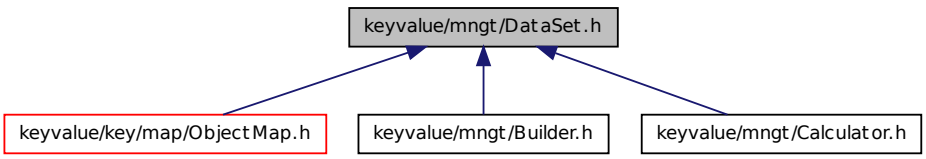
Declaration of `class` DataSet.

```
#include <map>
#include <typeinfo>
#include <boost/any.hpp>
#include <boost/utility.hpp>
#include <boost/weak_ptr.hpp>
#include "keyvalue/extern/String.h"
#include "keyvalue/extern/SharedPtr.h"
#include "keyvalue/key/Key.h"
#include "keyvalue/value/Result.h"
#include "keyvalue/value/TypeName.h"
#include "keyvalue/value/Value.h"
#include "keyvalue/sys/exception/Exception.h"
#include "keyvalue/sys/logger/Logger.h"
#include "keyvalue/util/Global.h"
```

Include dependency graph for DataSet.h:



This graph shows which files directly or indirectly include this file:



Classes

- class DataSet

  A named map from keys to values.

- struct DataSet::Record

  A wrapper around a value::Value.

- struct DataSet::Graph

  Simplified dependency graph.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

11.56.1 Detailed Description

Declaration of `class` DataSet.

11.57 keyvalue/mngt/DeclareBuilder.h File Reference

A helper file to declare Builders.

11.57.1 Detailed Description

A helper file to declare Builders.

11.58 keyvalue/mngt/DeclareCalculator.h File Reference

A helper file to declare Calculators.

11.58.1 Detailed Description

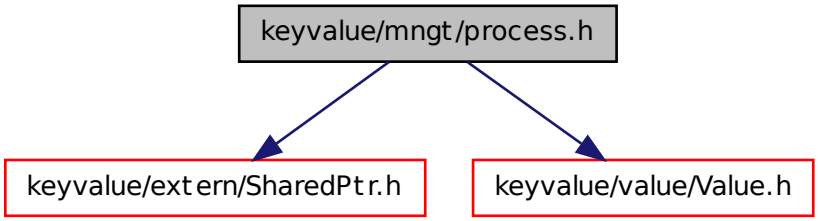
A helper file to declare Calculators. For details, see keyvalue/mngt/Calculator.h

11.59 keyvalue/mngt/process.h File Reference

Declaration of function keyvalue::process().

```
#include "keyvalue/extern/SharedPtr.h"
#include "keyvalue/value/Value.h"
```

Include dependency graph for process.h:



Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::frontend

  All `classes` *and functions needed by frontends belong to this namespace.*

Functions

- value::Value process (frontend::Queue &data)

  Processes a frontend::Queue.

- value::Value process (shared_ptr< DataSet > data)

  Processes a DataSet.

11.59.1 Detailed Description

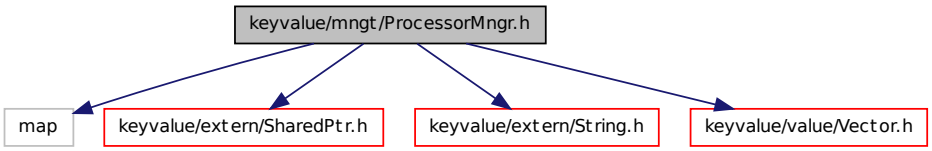
Declaration of function keyvalue::process().

11.60 keyvalue/mngt/ProcessorMngr.h File Reference

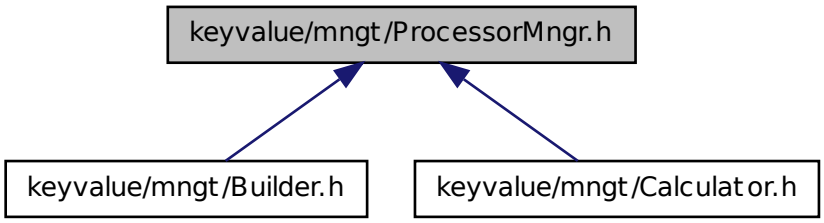
Declaration of `class` ProcessorMngr.

```
#include <map>
#include "keyvalue/extern/SharedPtr.h"
#include "keyvalue/extern/String.h"
#include "keyvalue/value/Vector.h"
```

Include dependency graph for ProcessorMngr.h:



This graph shows which files directly or indirectly include this file:



Classes

- class ProcessorMngr

  The processor manager.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::value

  All KeyValue containers belong to this namespace.

11.60.1 Detailed Description

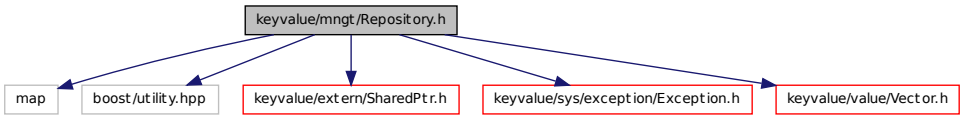
Declaration of `class` ProcessorMngr.

11.61 keyvalue/mngt/Repository.h File Reference

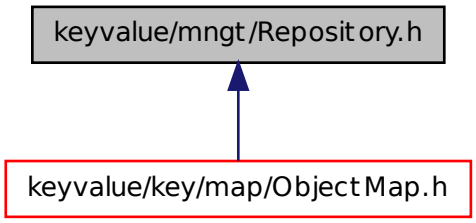
Declaration of `class` Repository.

```
#include <map>
#include <boost/utility.hpp>
#include "keyvalue/extern/SharedPtr.h"
#include "keyvalue/sys/exception/Exception.h"
#include "keyvalue/value/Vector.h"
```

Include dependency graph for Repository.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Repository

  The Repository of DataSets.

- class Repository::NotFound

  Exception thrown by get() when a DataSet is not found.

Namespaces

- namespace keyvalue

All functionalities of KeyValue library are inside this `namespace`.

11.61.1 Detailed Description

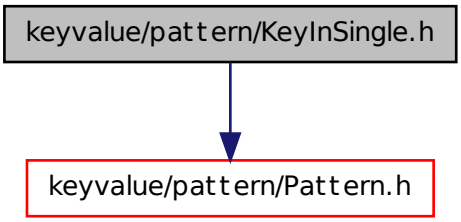
Declaration of `class` Repository.

11.62 keyvalue/pattern/KeyInSingle.h File Reference

Declaration of `class` KeyInSingle.

```
#include "keyvalue/pattern/Pattern.h"
```

Include dependency graph for KeyInSingle.h:



Classes

- class KeyInSingle

  Pattern where a value::Single object, recognized as a key, is followed by any value::Value.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::pattern

  Key-value pattern recognition `class` *are members of this namespace.*

11.62.1 Detailed Description

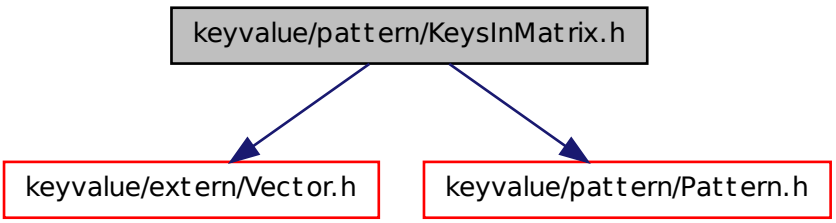
Declaration of `class` KeyInSingle.

11.63 keyvalue/pattern/KeysInMatrix.h File Reference

Declaration of `class` KeysInMatrix.

```
#include "keyvalue/extern/Vector.h"
#include "keyvalue/pattern/Pattern.h"
```

Include dependency graph for KeysInMatrix.h:



Classes

- class KeysInMatrix

  Pattern made by a value::Matrix where all elements either in the first row or in the first column are keys.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::pattern

  Key-value pattern recognition `class` are members of this namespace.

11.63.1 Detailed Description

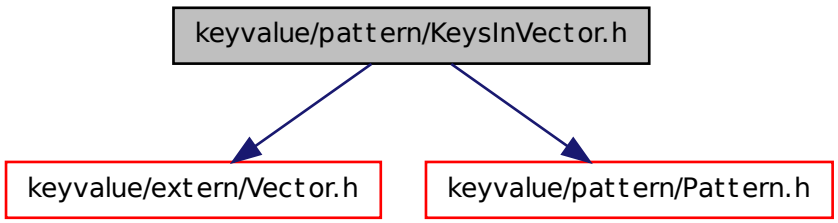
Declaration of `class` KeysInMatrix.

11.64 keyvalue/pattern/KeysInVector.h File Reference

Declaration of `class` KeysInVector.

```
#include "keyvalue/extern/Vector.h"
#include "keyvalue/pattern/Pattern.h"
```

Include dependency graph for KeysInVector.h:



Classes

- class KeysInVector

  Pattern made by a value::Vector whose entries are keys followed by a value::Vector or value::Matrix.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::pattern

  Key-value pattern recognition `class` *are members of this namespace.*

11.64.1 Detailed Description

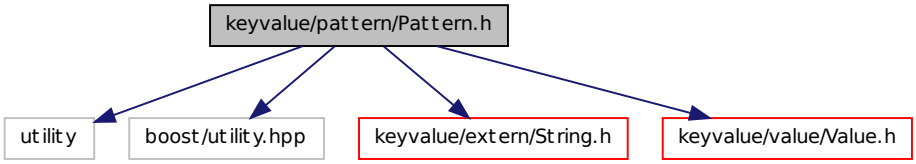
Declaration of `class` KeysInVector.

11.65 keyvalue/pattern/Pattern.h File Reference

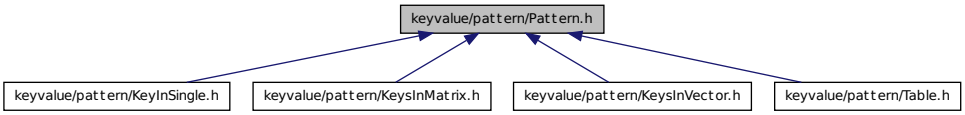
Declaration of `class` Pattern and functions keyvalue::pattern::isKey() and keyvalue::pattern::getKey().

```
#include <utility>
#include <boost/utility.hpp>
#include "keyvalue/extern/String.h"
#include "keyvalue/value/Value.h"
```

Include dependency graph for Pattern.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Pattern

  Protocol `class` which defines the interface for pattern recognition `classes`.

- class Pattern::QueueRaii

  This `class` implements RAII for a frontend::Queue.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::frontend

  All `classes` and functions needed by frontends belong to this namespace.

- namespace keyvalue::pattern

  Key-value pattern recognition `class` are members of this namespace.

Functions

- bool isKey (const value::Variant &variant)

  Checks if variant may be considered as a key.

- string getKey (const value::Variant &variant)
  Gets the key held by variant.

11.65.1 Detailed Description

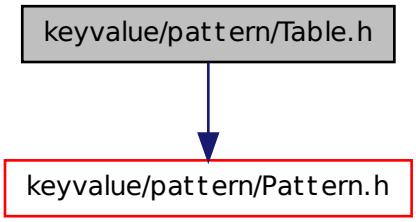
Declaration of `class` Pattern and functions keyvalue::pattern::isKey() and keyvalue::pattern::getKey().

11.66 keyvalue/pattern/Table.h File Reference

Declaration of `class` Table.

```
#include "keyvalue/pattern/Pattern.h"
```

Include dependency graph for Table.h:



Classes

- class Table
  Pattern where a value::Matrix contains the row, column and table keys.

Namespaces

- namespace keyvalue
  All functionalities of KeyValue library are inside this `namespace`.
- namespace keyvalue::pattern
  Key-value pattern recognition `class` are members of this namespace.

11.66.1 Detailed Description

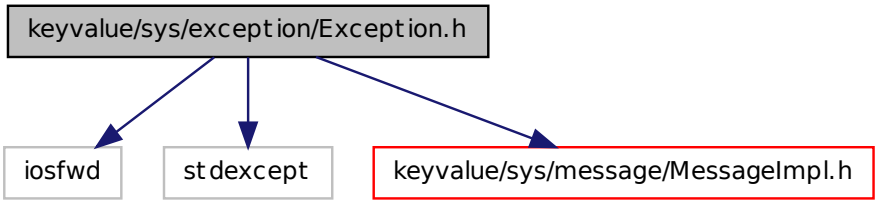
Declaration of `class` Table.

11.67 keyvalue/sys/exception/Exception.h File Reference

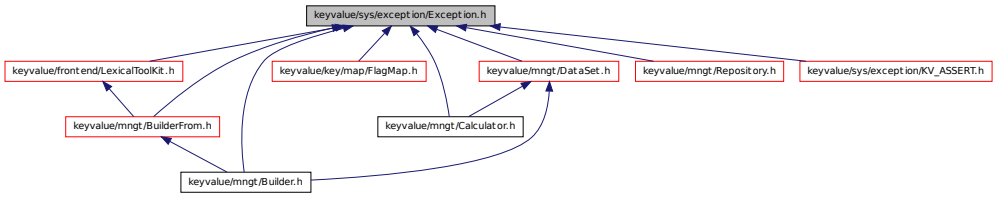
Declaration of `class`es Exception, RuntimeError and LogicError.

```
#include <iosfwd>
#include <stdexcept>
#include "keyvalue/sys/message/MessageImpl.h"
```

Include dependency graph for Exception.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Exception

  Base protocol `class` for all exceptions.

- class ExceptionImpl< StdExcept, MessageType >

  Concrete `template class` which implements Exception's pure virtual methods.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::exception

  All exception functionalities are inside this `namespace`.

Typedefs

- typedef ExceptionImpl< std::runtime_error, Error > **RuntimeError**
- typedef ExceptionImpl< std::logic_error, Logic > **LogicError**

Functions

- std::ostream & operator<< (std::ostream &os, const Exception &exception)

  ostream operator<<() for Exception.

- template<typename LhsType , typename RhsType >
  LhsType & operator& (const LhsType &lhs, const RhsType &rhs)

  Non member operator& for exceptions.

11.67.1 Detailed Description

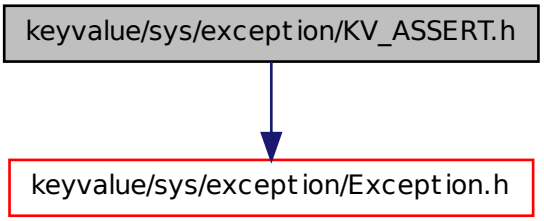
Declaration of `class`es Exception, RuntimeError and LogicError.

11.68 keyvalue/sys/exception/KV_ASSERT.h File Reference

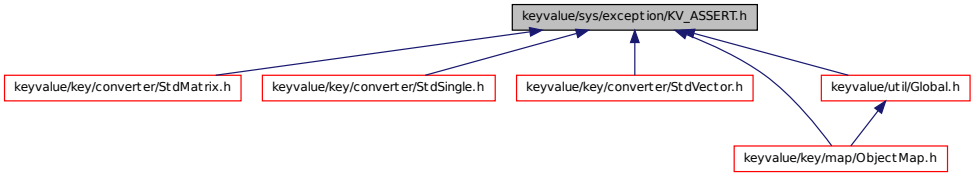
Implementation of macro KV_ASSERT.

```
#include "keyvalue/sys/exception/Exception.h"
```

Include dependency graph for KV_ASSERT.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define KV_ASSERT(cond, msg)

  The assert macro.

11.68.1 Detailed Description

Implementation of macro KV_ASSERT.

11.68.2 Define Documentation

11.68.2.1 #define KV_ASSERT(*cond, msg*)

Value:

```
{                                                \
  if (!(cond))                                                  \
    throw ::keyvalue::LogicError() & __FILE__ & " (" & __LINE__ & ") :\n" &\
      msg;                                                      \
}
```

The assert macro.

This macro is a replacement for the C standard `assert` macro. Its definition depends on whether the macro NDEBUG is defined or not.

When NDEBUG is not defined, KV_ASSERT checks if a condition holds and, when it does not, it throws a LogicError carrying information on the invalid condition and where it has occurred.

When NDEBUG is defined, KV_ASSERT does not do anything.

Parameters

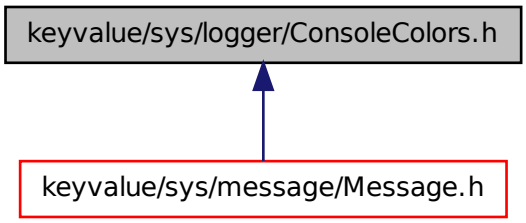
cond : Condition to be tested;

msg : Error message to be reported.

11.69 keyvalue/sys/logger/ConsoleColors.h File Reference

Definition of `enum` Color.

This graph shows which files directly or indirectly include this file:



Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::logger

  All logger functionalities are inside this namespace.

Enumerations

- enum Color {
  black = 0, **red**, **green**, **yellow**,
  blue, **pink**, **cyan**, **white**,
  grey, **bold_red**, **bold_green**, **bold_yellow**,
  bold_blue, **bold_pink**, **bold_cyan**, **bold_white** }

  Console colors.

11.69.1 Detailed Description

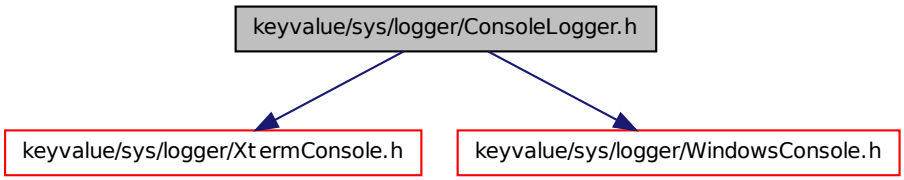
Definition of `enum` Color.

11.70 keyvalue/sys/logger/ConsoleLogger.h File Reference

Declaration of `class` ConsoleLogger.

```
#include "keyvalue/sys/logger/XtermConsole.h"
#include "keyvalue/sys/logger/WindowsConsole.h"
```

Include dependency graph for ConsoleLogger.h:



Classes

- class ConsoleLogger

  Implements a console Logger.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::logger

  All logger functionalities are inside this namespace.

Defines

- #define **CONSOLELOGGER** XtermConsole

11.70.1 Detailed Description

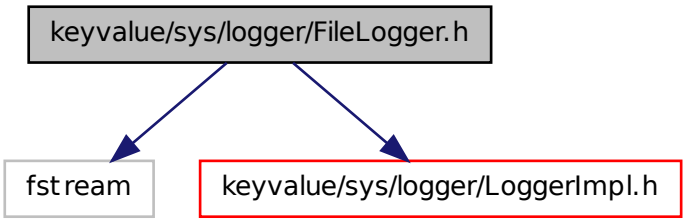
Declaration of `class` ConsoleLogger.

11.71 keyvalue/sys/logger/FileLogger.h File Reference

Declaration of `class` FileLogger.

```
#include <fstream>
#include "keyvalue/sys/logger/LoggerImpl.h"
```

Include dependency graph for FileLogger.h:



Classes

- class FileLogger

  Implements a file Logger.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::logger

  All logger functionalities are inside this namespace.

11.71.1 Detailed Description

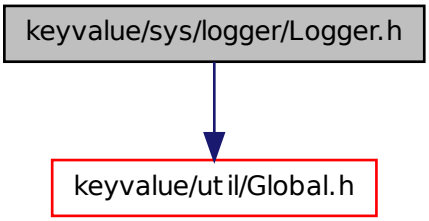
Declaration of `class` FileLogger.

11.72 keyvalue/sys/logger/Logger.h File Reference

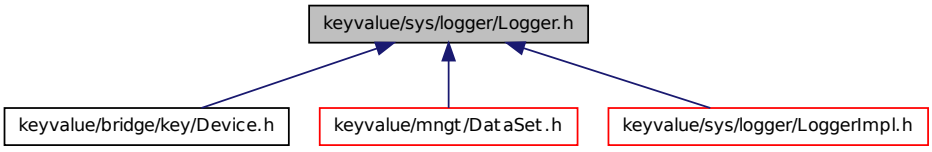
Declaration of `class` Logger.

```
#include "keyvalue/util/Global.h"
```

Include dependency graph for Logger.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Logger

  Protocol `class` which defines the interface for all types of logger.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::logger

  All logger functionalities are inside this namespace.

- namespace keyvalue::util

  Utility `classes` and functions are member of this `namespace`.

Functions

- void resetGlobal ()

  Resets util::Global<Logger> to its default (StdLogger).

11.72.1 Detailed Description

Declaration of `class` Logger.

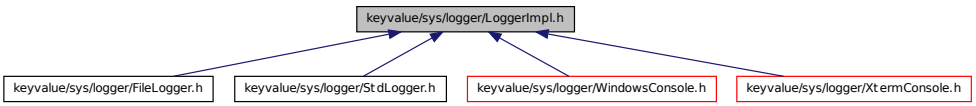
11.73 keyvalue/sys/logger/LoggerImpl.h File Reference

Declaration and implementation of `template class` LoggerImpl.

```
#include "keyvalue/bridge/Bridge.h"
#include "keyvalue/sys/logger/Logger.h"
#include "keyvalue/sys/logger/policy/AddPrefix.h"
#include "keyvalue/sys/logger/policy/ForwardToGlobalLogger.h"
#include "keyvalue/sys/logger/policy/IgnoreColor.h"
#include "keyvalue/sys/message/MessageImpl.h"
#include "keyvalue/util/Global.h"
```

Include dependency graph for LoggerImpl.h:

This graph shows which files directly or indirectly include this file:



Classes

- class LoggerImpl< PrefixPolicy, ColorPolicy, SafetyPolicy >

  Abstract `class` which implements main methods of concrete Loggers.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::logger

  All logger functionalities are inside this namespace.

11.73.1 Detailed Description

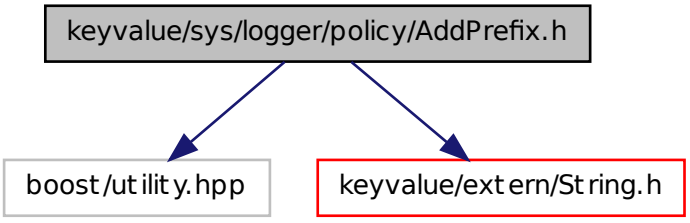
Declaration and implementation of `template class` LoggerImpl.

11.74 keyvalue/sys/logger/policy/AddPrefix.h File Reference

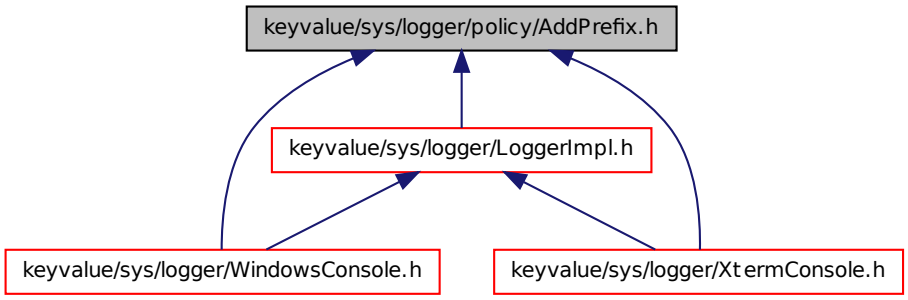
Declaration of `class` AddPrefix.

```
#include <boost/utility.hpp>
#include "keyvalue/extern/String.h"
```

Include dependency graph for AddPrefix.h:



This graph shows which files directly or indirectly include this file:



Classes

- class AddPrefix

  Defines a policy for Message prefixes, which is, to send the prefix to the underlying Logger's device.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::logger

  All logger functionalities are inside this namespace.

11.74.1 Detailed Description

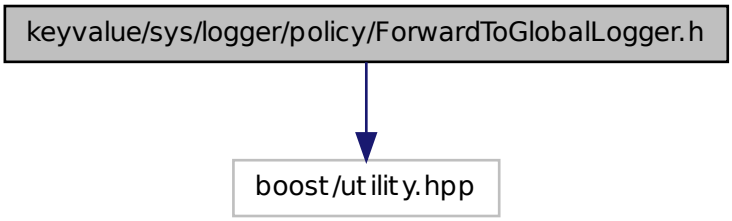
Declaration of `class` AddPrefix.

11.75 keyvalue/sys/logger/policy/ForwardToGlobalLogger.h File Reference

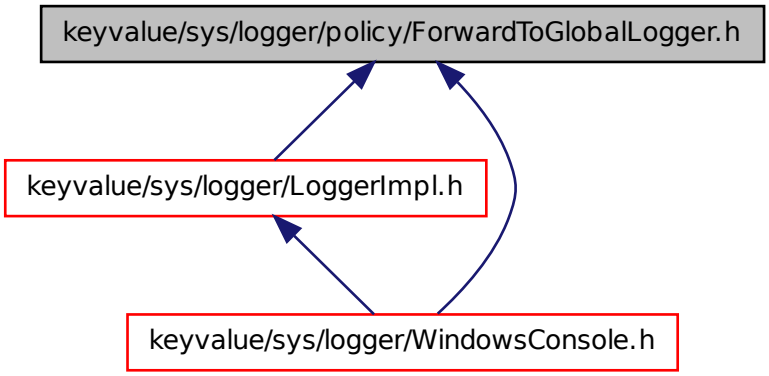
Declaration of `class` ForwardToGlobalLogger.

```
#include <boost/utility.hpp>
```

Include dependency graph for ForwardToGlobalLogger.h:



This graph shows which files directly or indirectly include this file:



Classes

- class ForwardToGlobalLogger

  Defines a policy to apply when the Logger fails to process a Message, namely, forward it to GlobalLogger.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::logger

All logger functionalities are inside this namespace.

11.75.1 Detailed Description

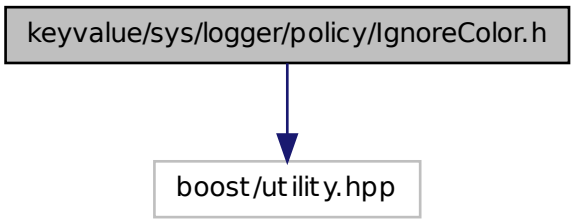
Declaration of `class` ForwardToGlobalLogger.

11.76 keyvalue/sys/logger/policy/IgnoreColor.h File Reference

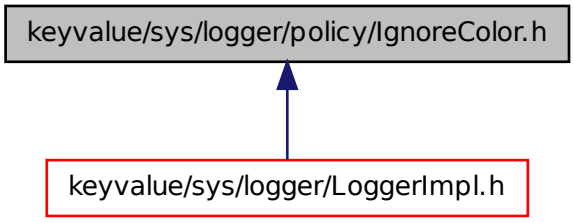
Declaration of `class` IgnoreColor.

```
#include <boost/utility.hpp>
```

Include dependency graph for IgnoreColor.h:



This graph shows which files directly or indirectly include this file:



Classes

- class IgnoreColor

Defines a policy for Message colors, which is, to ignore them.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::logger

  All logger functionalities are inside this namespace.

11.76.1 Detailed Description

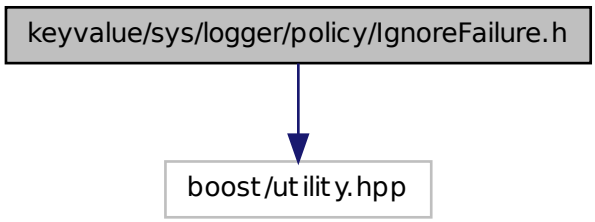
Declaration of `class` IgnoreColor.

11.77 keyvalue/sys/logger/policy/IgnoreFailure.h File Reference

Declaration of `class` IgnoreFailure.

```
#include <boost/utility.hpp>
```

Include dependency graph for IgnoreFailure.h:



Classes

- class IgnoreFailure

  Defines a policy to apply when the logger fails, which is, to ignore the failure.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::logger
  All logger functionalities are inside this namespace.

11.77.1 Detailed Description

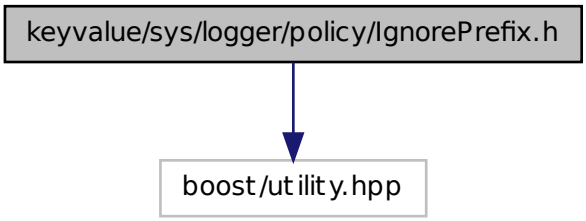
Declaration of `class` IgnoreFailure.

11.78 keyvalue/sys/logger/policy/IgnorePrefix.h File Reference

Declaration of `class` IgnorePrefix.

```
#include <boost/utility.hpp>
```

Include dependency graph for IgnorePrefix.h:



Classes

- class IgnorePrefix
  Defines a policy for Message prefixes, which is, to ignore them.

Namespaces

- namespace keyvalue
  All functionalities of KeyValue library are inside this `namespace`.
- namespace keyvalue::logger
  All logger functionalities are inside this namespace.

11.78.1 Detailed Description

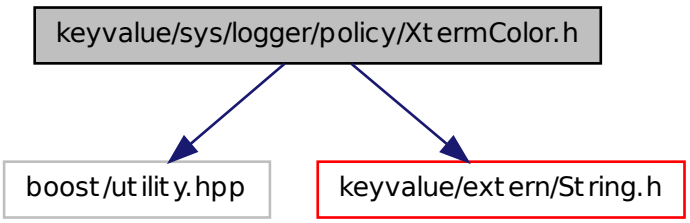
Declaration of `class` IgnorePrefix.

11.79 keyvalue/sys/logger/policy/XtermColor.h File Reference

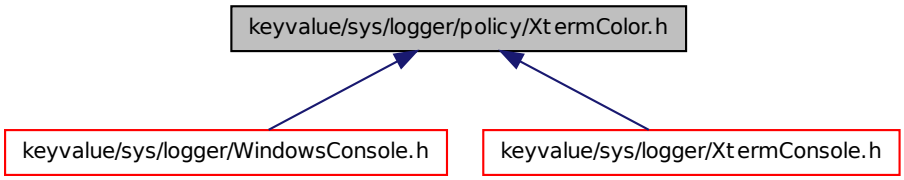
Declaration of `class` XtermColor.

```
#include <boost/utility.hpp>
#include "keyvalue/extern/String.h"
```

Include dependency graph for XtermColor.h:



This graph shows which files directly or indirectly include this file:



Classes

- class XtermColor

  Defines a policy for Message color, which is, to set the underlying device of the Logger to print in Message's color.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::logger

All logger functionalities are inside this namespace.

11.79.1 Detailed Description

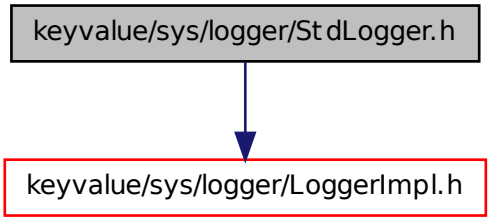
Declaration of `class` XtermColor.

11.80 keyvalue/sys/logger/StdLogger.h File Reference

Declaration of `class` StdLogger.

```
#include "keyvalue/sys/logger/LoggerImpl.h"
```

Include dependency graph for StdLogger.h:



Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::logger

  All logger functionalities are inside this namespace.

11.80.1 Detailed Description

Declaration of `class` StdLogger.

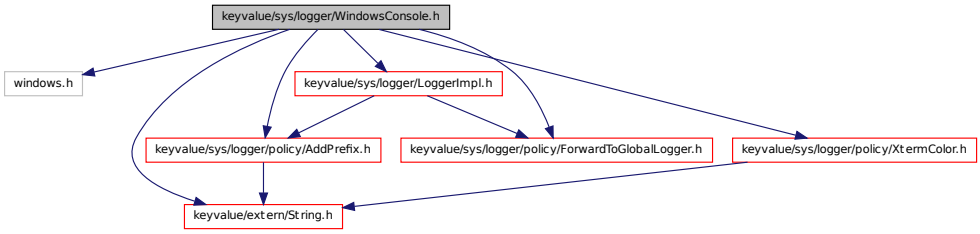
11.81 keyvalue/sys/logger/WindowsConsole.h File Reference

Declaration of `class` WindowsConsole.

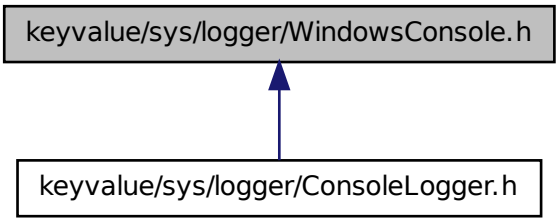
```
#include <windows.h>
```

```
#include "keyvalue/extern/String.h"
#include "keyvalue/sys/logger/LoggerImpl.h"
#include "keyvalue/sys/logger/policy/AddPrefix.h"
#include "keyvalue/sys/logger/policy/XtermColor.h"
#include "keyvalue/sys/logger/policy/ForwardToGlobalLogger.h"
```

Include dependency graph for WindowsConsole.h:



This graph shows which files directly or indirectly include this file:



Classes

- class WindowsConsole

  This `class` implements a Windows console Logger.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::logger

All logger functionalities are inside this namespace.

11.81.1 Detailed Description

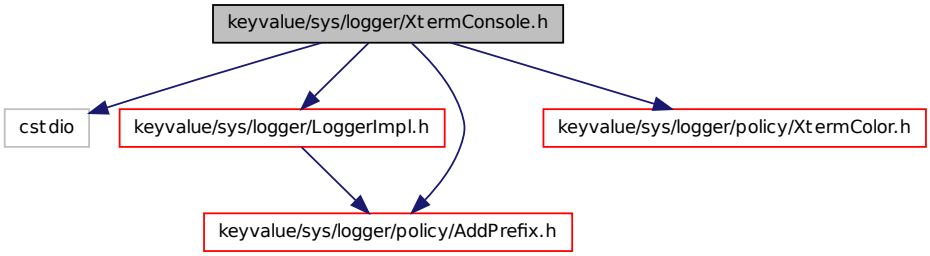
Declaration of `class` WindowsConsole.

11.82 keyvalue/sys/logger/XtermConsole.h File Reference

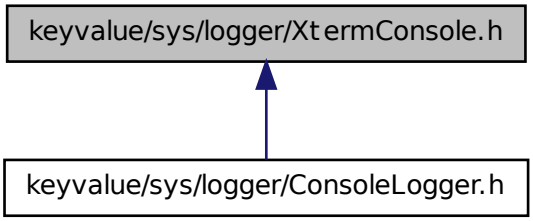
Declaration of `class` XtermConsole.

```
#include <cstdio>
#include "keyvalue/sys/logger/LoggerImpl.h"
#include "keyvalue/sys/logger/policy/AddPrefix.h"
#include "keyvalue/sys/logger/policy/XtermColor.h"
```

Include dependency graph for XtermConsole.h:



This graph shows which files directly or indirectly include this file:



Classes

- class XtermConsole

  Implements an xterm console Logger.

- class XtermConsole::FileRaii

  XtermConsole's helper `class` for file management.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::logger

  All logger functionalities are inside this namespace.

11.82.1 Detailed Description

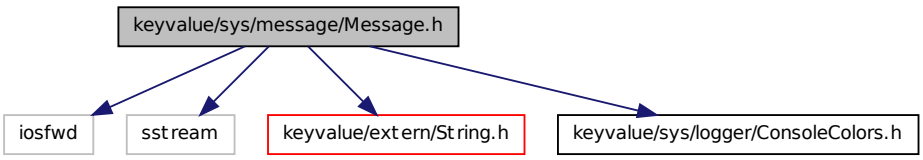
Declaration of `class` XtermConsole.

11.83 keyvalue/sys/message/Message.h File Reference

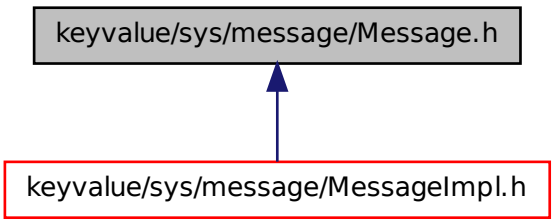
Declaration of `class` Message.

```
#include <iosfwd>
#include <sstream>
#include "keyvalue/extern/String.h"
#include "keyvalue/sys/logger/ConsoleColors.h"
```

Include dependency graph for Message.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Message

  Abstract `class` which defines the interface for all types of message.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

Functions

- std::ostream & operator<< (std::ostream &os, const Message &message)

  ostream operator<<() for Message.

11.83.1 Detailed Description

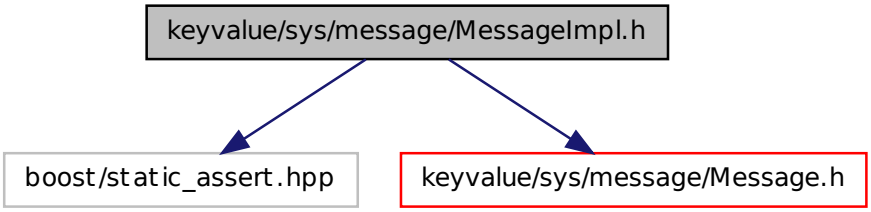
Declaration of `class` Message.

11.84 keyvalue/sys/message/MessageImpl.h File Reference

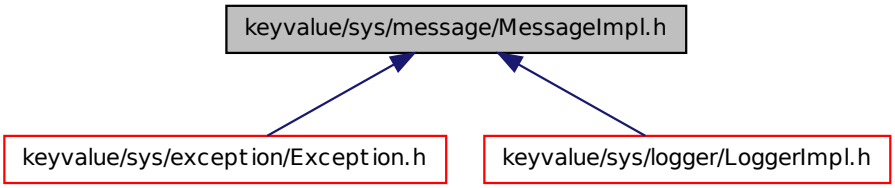
Declaration of `class`es MessageImpl, Error, Info, Warning, Report and Debug.

```
#include <boost/static_assert.hpp>
#include "keyvalue/sys/message/Message.h"
```

Include dependency graph for MessageImpl.h:



This graph shows which files directly or indirectly include this file:



Classes

- class MessageImpl< id >

  Implementation of `class` *Message.*

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

Typedefs

- typedef MessageImpl< 0 > **Error**
- typedef MessageImpl< 1 > **Logic**
- typedef MessageImpl< 2 > **Info**
- typedef MessageImpl< 3 > **Warning**

- typedef MessageImpl< 4 > **Report**
- typedef MessageImpl< 5 > **Debug**

11.84.1 Detailed Description

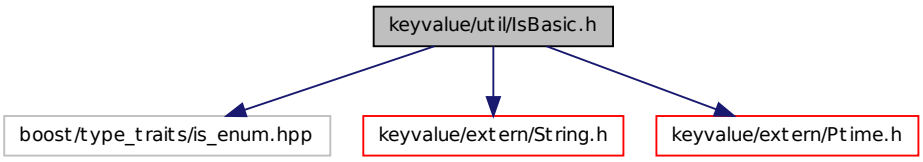
Declaration of `classes` MessageImpl, Error, Info, Warning, Report and Debug.

11.85 keyvalue/util/IsBasic.h File Reference

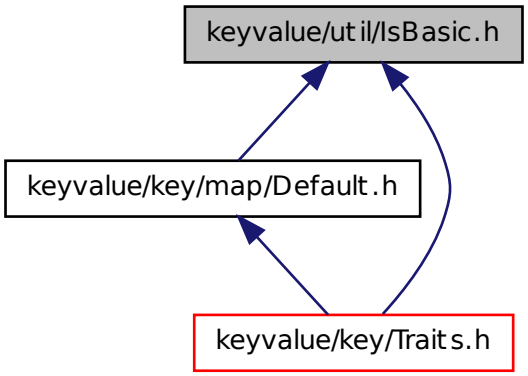
Declaration and implementation of `template class` IsBasic.

```
#include <boost/type_traits/is_enum.hpp>
#include "keyvalue/extern/String.h"
#include "keyvalue/extern/Ptime.h"
```

Include dependency graph for IsBasic.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct IsBasic< ElementType >

  This meta-function checks if a type is either `bool`, `double`, `string`, `ptime` *or* `unsigned int` *or not.*

- struct IsBasicOrEnum< ElementType >

  This meta-function checks if a type is either `bool`, `double`, `string`, `ptime`, `unsigned int` *or enum or not.*

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::util

  Utility `classes` *and functions are member of this* `namespace`.

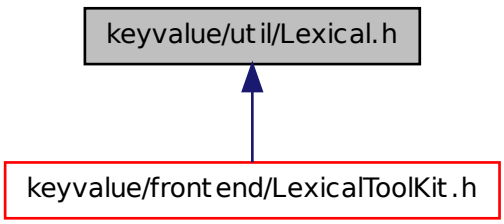
11.85.1 Detailed Description

Declaration and implementation of `template class` IsBasic.

11.86 keyvalue/util/Lexical.h File Reference

Declaration of `template` function Lexical.

This graph shows which files directly or indirectly include this file:



Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::util

  Utility `classes` and functions are member of this `namespace`.

Functions

- template<typename From , typename To >
  To Lexical (const From &input)

  Converts from type From to type To.

11.86.1 Detailed Description

Declaration of `template` function Lexical.

11.87 keyvalue/util/NullDeleter.h File Reference

Declaration of `class` NullDeleter.

Classes

- class NullDeleter

  A `shared_ptr` deleter that does not do anything.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::util

  Utility `classes` and functions are member of this `namespace`.

11.87.1 Detailed Description

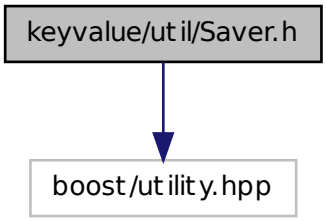
Declaration of `class` NullDeleter.

11.88 keyvalue/util/Saver.h File Reference

Definition and implementation of `template class` Saver.

```
#include <boost/utility.hpp>
```

Include dependency graph for Saver.h:



Classes

- class Saver< SavedType >

  Saves a variable reference at construction time to restore/set its value at destruction time.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::util

  Utility `classes` *and functions are member of this* `namespace`.

11.88.1 Detailed Description

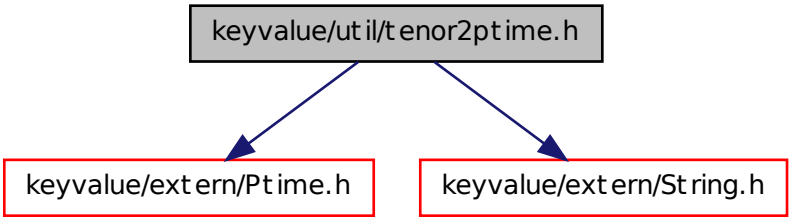
Definition and implementation of `template class` Saver.

11.89 keyvalue/util/tenor2ptime.h File Reference

Declaration of function tenor2ptime().

```
#include "keyvalue/extern/Ptime.h"
#include "keyvalue/extern/String.h"
```

Include dependency graph for tenor2ptime.h:



Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::util

  Utility `classes` *and functions are member of this* `namespace`.

Functions

- ptime tenor2ptime (const ptime &start, const string &tenor)

  Convert a tenor string (from a start date) into the corresponding date.

11.89.1 Detailed Description

Declaration of function tenor2ptime().

11.90 keyvalue/util/Util.h File Reference

Documentation of `namespace` util.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::util

  Utility `classes` *and functions are member of this* `namespace`.

11.90.1 Detailed Description

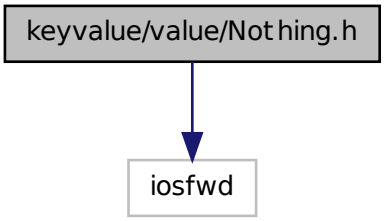
Documentation of `namespace` util.

11.91 keyvalue/value/Nothing.h File Reference

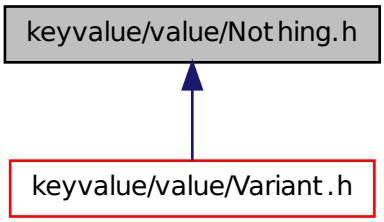
Declaration of `class` Nothing.

`#include <iosfwd>`

Include dependency graph for Nothing.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Nothing

  Empty `class` to represent empty data.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::value

  All KeyValue containers belong to this namespace.

Functions

- std::ostream & operator<< (std::ostream &os, const Nothing &nothing)

  ostream operator<<() for Nothing.

11.91.1 Detailed Description

Declaration of `class` Nothing.

11.92 keyvalue/value/Parent.h File Reference

Declaration of `template class` Parent.

Classes

- struct Parent< T >

  Primary Parent `template` *meta-function.*

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::value

  All KeyValue containers belong to this namespace.

11.92.1 Detailed Description

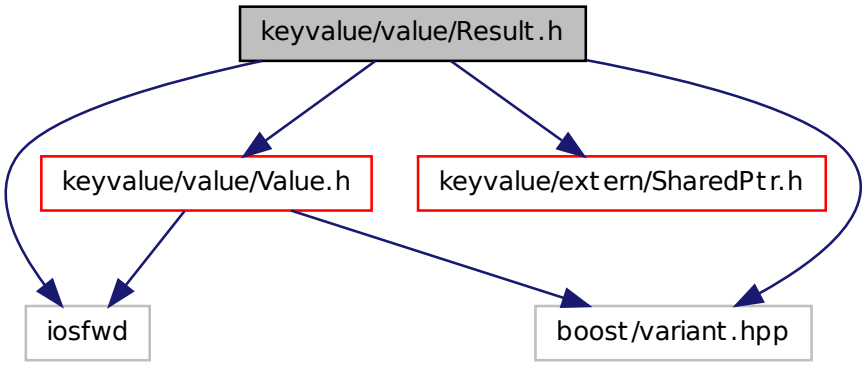
Declaration of `template class` Parent.

11.93 keyvalue/value/Result.h File Reference

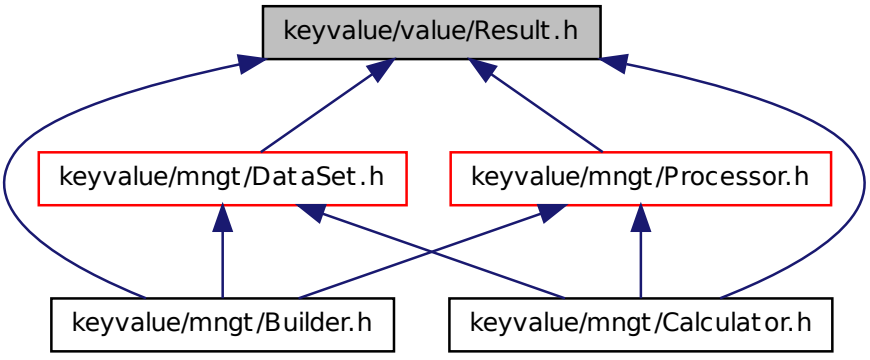
Declaration of classes Result and ObjectPtr.

```
#include <iosfwd>
#include <boost/variant.hpp>
#include "keyvalue/extern/SharedPtr.h"
#include "keyvalue/value/Value.h"
```

Include dependency graph for Result.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Result

  A single-valued container for ObjectPtr or Value.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::value

  All KeyValue containers belong to this namespace.

Typedefs

- typedef shared_ptr< void > **ObjectPtr**

Functions

- std::ostream & operator<< (std::ostream &os, const Result &rhs)

  ostream operator<<() for Result.

11.93.1 Detailed Description

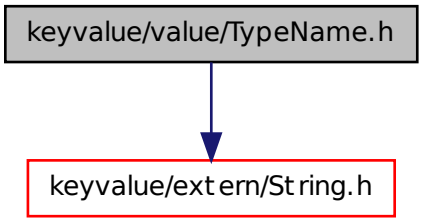
Declaration of `class`es Result and ObjectPtr.

11.94 keyvalue/value/TypeName.h File Reference

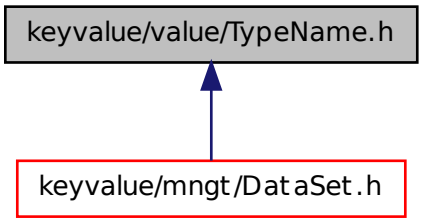
Definition of `template class` TypeName.

```
#include "keyvalue/extern/String.h"
```

Include dependency graph for TypeName.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct TypeName< Type >

  Meta function which returns the name of type.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::value

  All KeyValue containers belong to this namespace.

11.94.1 Detailed Description

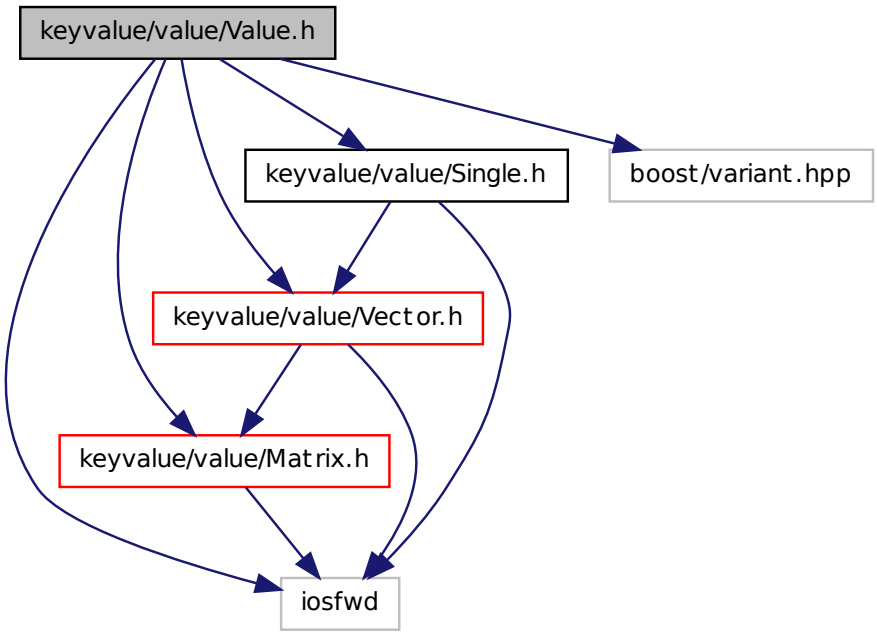
Definition of `template class` TypeName.

11.95 keyvalue/value/Value.h File Reference

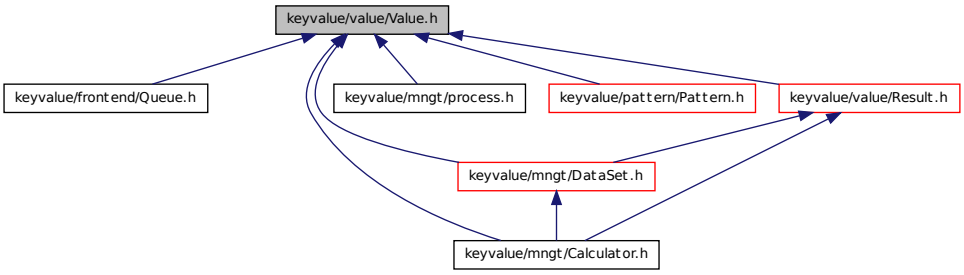
Declaration and (partial) implementation of `class` Value.

```
#include <iosfwd>
#include <boost/variant.hpp>
#include "keyvalue/value/Matrix.h"
#include "keyvalue/value/Single.h"
#include "keyvalue/value/Vector.h"
```

Include dependency graph for Value.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Value

  A single-valued container for Single, Vector or Matrix.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::value

  All KeyValue containers belong to this namespace.

Functions

- std::ostream & operator<< (std::ostream &os, const Value &rhs)

  ostream operator<<() for Value.

11.95.1 Detailed Description

Declaration and (partial) implementation of `class` Value.

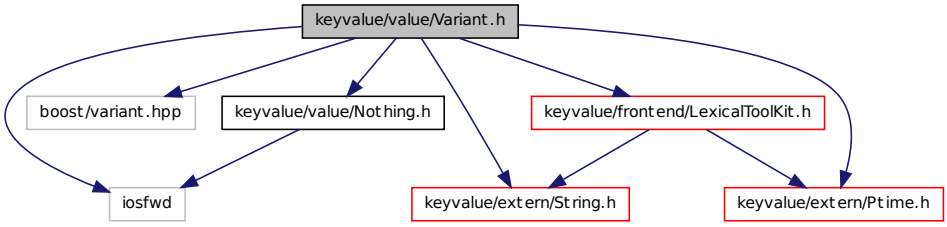
11.96 keyvalue/value/Variant.h File Reference

Declaration and (partial) implementation of `class` Variant.

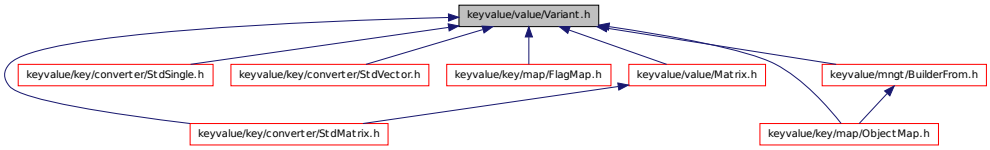
```
#include <iosfwd>
#include <boost/variant.hpp>
#include "keyvalue/extern/String.h"
```

```
#include "keyvalue/extern/Ptime.h"
#include "keyvalue/frontend/LexicalToolKit.h"
#include "keyvalue/value/Nothing.h"
```

Include dependency graph for Variant.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Variant

  Single-value multi-type container.

Namespaces

- namespace keyvalue

  All functionalities of KeyValue library are inside this `namespace`.

- namespace keyvalue::value

  All KeyValue containers belong to this namespace.

Functions

- std::ostream & operator<< (std::ostream &os, const Variant &rhs)

  ostream operator<<() for Variant.

11.96.1 Detailed Description

Declaration and (partial) implementation of `class` Variant.

Index

add
keyvalue::DataSet, 87
keyvalue::ProcessorMngr, 185
keyvalue::Repository, 190
apply
keyvalue::logger::ForwardToGlobalLogger, 106
keyvalue::logger::IgnoreFailure, 118

Bounded
keyvalue::key::Bounded, 66
build
keyvalue::BuilderFromVariant, 74

check
keyvalue::key::Decreasing, 90
keyvalue::key::Geq, 109
keyvalue::key::Greater, 115
keyvalue::key::Increasing, 121
keyvalue::key::Leq, 133
keyvalue::key::Less, 134
keyvalue::key::NoBound, 165
keyvalue::key::NonMonotone, 168
keyvalue::key::StrictlyDecreasing, 212
keyvalue::key::StrictlyIncreasing, 213
keyvalue::pattern::KeysInMatrix, 129
checkInput
keyvalue::key::Checker< ConverterType, value::Matrix >, 79
keyvalue::key::Checker< ConverterType, value::Single >, 80
keyvalue::key::Checker< ConverterType, value::Vector >, 82
checkOutput
keyvalue::key::Bounded, 66
keyvalue::key::Checker< ConverterType, value::Matrix >, 79
keyvalue::key::Checker< ConverterType, value::Single >, 81
keyvalue::key::Checker< ConverterType, value::Vector >, 82
keyvalue::key::Positive, 178
keyvalue::key::StrictlyPositive, 215
checkSize
keyvalue::key::Checker< ConverterType, value::Matrix >, 79
keyvalue::key::Checker< ConverterType, value::Vector >, 82
keyvalue::key::Matrix, 147
keyvalue::key::MonotoneBoundedVector, 163
keyvalue::key::Vector, 236
checkSoFar
keyvalue::key::Checker< ConverterType, value::Matrix >, 79
keyvalue::key::Checker< ConverterType, value::Single >, 80
keyvalue::key::Checker< ConverterType, value::Vector >, 82
keyvalue::key::MonotoneBoundedVector, 163
clear
keyvalue::Repository, 191
Color
keyvalue::logger, 56
convert
keyvalue::frontend::LexicalToolKit, 139

DataSet
keyvalue::DataSet, 87
Debug, 90
dummy_
keyvalue::ProcessorMngr, 186

endInitList
keyvalue::frontend::LexicalToolKit, 140
erase
keyvalue::Repository, 191
Error, 94
Exception
keyvalue::exception::Exception, 96
Export
keyvalue::key::Export, 100

find
keyvalue::DataSet, 87
keyvalue::Repository, 191
Flag
keyvalue::key::VectorOutput, 238
keyvalue::key::VectorOutputBase, 240
forward

keyvalue::logger::ForwardToGlobalLogger, 107

Geq
keyvalue::key::Geq, 108
get
keyvalue::key::Device, 93
keyvalue::key::FlagMap, 105
keyvalue::key::PartialMap, 174
keyvalue::key::VectorOutput, 238
keyvalue::util::Global, 111
keyvalue::value::Value, 223
keyvalue::value::Variant, 226, 227
getBound
keyvalue::key::Geq, 109
keyvalue::key::Greater, 115
keyvalue::key::Leq, 133
keyvalue::key::Less, 134
keyvalue::key::NoBound, 165
getCmdList
keyvalue::ProcessorMngr, 186
getColor
keyvalue::Message, 155
keyvalue::MessageImpl, 159
getCommand
keyvalue::ProcessorMngr, 186
getCommandName
keyvalue::Command, 84
getCompleteInfo
keyvalue::Bridge, 68
getCoreLibraryName
keyvalue::Bridge, 68
getDataSet
keyvalue::Repository, 191
getGraph
keyvalue::DataSet, 89
getInstance
keyvalue::ProcessorInstantiator, 184
keyvalue::util::Global, 111
getKey
keyvalue::pattern, 57
getLevel
keyvalue::logger::FileLogger, 102
keyvalue::logger::Logger, 141
keyvalue::logger::LoggerImpl, 143
keyvalue::logger::WindowsConsole, 243
keyvalue::logger::XtermConsole, 246
keyvalue::Message, 155
keyvalue::MessageImpl, 159
getList
keyvalue::Repository, 192
getMessage
keyvalue::exception::Exception, 96
keyvalue::exception::ExceptionImpl, 98
getName
keyvalue::Builder, 71
keyvalue::Calculator, 76
keyvalue::Command, 84
keyvalue::DataSet, 87
keyvalue::key::Bounded, 67
keyvalue::key::Checker< ConverterType, value::Matrix >, 79
keyvalue::key::Checker< ConverterType, value::Single >, 81
keyvalue::key::Checker< ConverterType, value::Vector >, 82
keyvalue::key::Decreasing, 91
keyvalue::key::Device, 93
keyvalue::key::Export, 100
keyvalue::key::FlagMap, 105
keyvalue::key::Geq, 109
keyvalue::key::Global, 113
keyvalue::key::Greater, 115
keyvalue::key::Imports, 120
keyvalue::key::Increasing, 121
keyvalue::key::Key, 124
keyvalue::key::Leq, 133
keyvalue::key::Less, 135
keyvalue::key::Level, 136
keyvalue::key::Matrix, 148
keyvalue::key::MonotoneBoundedVector, 164
keyvalue::key::NoBound, 166
keyvalue::key::NoMap, 167
keyvalue::key::NonMonotone, 168
keyvalue::key::ObjectMap, 170
keyvalue::key::PartialMap, 174
keyvalue::key::Positive, 178
keyvalue::key::ProcessNow, 180
keyvalue::key::Processor, 183
keyvalue::key::Single, 197
keyvalue::key::StrictlyDecreasing, 212
keyvalue::key::StrictlyIncreasing, 213
keyvalue::key::StrictlyPositive, 215
keyvalue::key::Traits, 220
keyvalue::key::Vector, 236, 237
keyvalue::key::VectorOutput, 239
keyvalue::Processor, 181
getNCols
keyvalue::value::Matrix, 151
keyvalue::value::Single, 202
keyvalue::value::Vector, 232
getNRows
keyvalue::value::Matrix, 151
keyvalue::value::Single, 202
keyvalue::value::Vector, 232
getObject
keyvalue::Builder, 71
keyvalue::BuilderFrom, 72

keyvalue::BuilderFromVariant, 73
getObjectPtr
keyvalue::value::Result, 194
getOutput
keyvalue::key::StdMatrix, 206
keyvalue::key::StdSingle, 209
keyvalue::key::StdVector, 211
getPath
keyvalue::frontend::FrontEnd, 107
getPrefix
keyvalue::Message, 155
keyvalue::MessageImpl, 159
getProcessor
keyvalue::ProcessorMngr, 185
getResult
keyvalue::Builder, 71
keyvalue::Calculator, 77
keyvalue::Command, 84
keyvalue::Processor, 181
getSaved
keyvalue::util::Saver, 196
getSimpleInfo
keyvalue::Bridge, 68
getSize
keyvalue::Repository, 191
keyvalue::value::Single, 200
keyvalue::value::Vector, 231
getString
keyvalue::Message, 155
keyvalue::MessageImpl, 159
getValue
keyvalue::Calculator, 77
keyvalue::DataSet, 88
keyvalue::value::Result, 194
Global
keyvalue::key::Global, 113
keyvalue::util::Global, 110, 111
Greater
keyvalue::key::Greater, 115

Imports
keyvalue::key::Imports, 120
Info, 121
InputType_
keyvalue::key::StdMatrix, 205
keyvalue::key::StdSingle, 207
keyvalue::key::StdVector, 210
insert
keyvalue::key::StdMatrix, 205
keyvalue::key::StdSingle, 208
keyvalue::key::StdVector, 211
intrusive_ptr_add_ref
keyvalue::value, 61
keyvalue::value::Matrix, 152
intrusive_ptr_release
keyvalue::value, 61
keyvalue::value::Matrix, 152
IO
keyvalue::frontend::LexicalToolKit, 138
is
keyvalue::value::Variant, 226
isEmpty
keyvalue::frontend::Queue, 187
keyvalue::key::StdMatrix, 205
keyvalue::key::StdSingle, 208
keyvalue::key::StdVector, 211
keyvalue::pattern::KeyInSingle, 126
keyvalue::pattern::KeysInMatrix, 129
keyvalue::pattern::KeysInVector, 131
keyvalue::pattern::Pattern, 176
keyvalue::pattern::Table, 218
isEnabled
keyvalue::frontend::LexicalToolKit, 139
isKey
keyvalue::pattern, 57

Key
keyvalue::key::Key, 123
keyvalue, 47
operator<<, 49
process, 49
keyvalue/ Directory Reference, 31
keyvalue/bridge/ Directory Reference, 22
keyvalue/bridge/Bridge.h, 249
keyvalue/bridge/key/ Directory Reference, 29
keyvalue/bridge/key/Device.h, 250
keyvalue/bridge/key/Global.h, 251
keyvalue/bridge/key/Level.h, 254
keyvalue/bridge/Register.h, 254
keyvalue/extern/ Directory Reference, 25
keyvalue/extern/Ptime.h, 255
keyvalue/extern/SharedPtr.h, 256
keyvalue/extern/String.h, 256
keyvalue/extern/Vector.h, 257
keyvalue/frontend/ Directory Reference, 26
keyvalue/frontend/FrontEnd.h, 261
keyvalue/frontend/LexicalToolKit.h, 262
keyvalue/frontend/LexicalToolKitPairs.h, 264
keyvalue/frontend/Queue.h, 264
keyvalue/key/ Directory Reference, 28
keyvalue/key/Checker.h, 265
keyvalue/key/converter/ Directory Reference, 23
keyvalue/key/converter/StdMatrix.h, 267
keyvalue/key/converter/StdSingle.h, 268
keyvalue/key/converter/StdVector.h, 269
keyvalue/key/generic/ Directory Reference, 27
keyvalue/key/generic/bound/ Directory Reference, 21

keyvalue/key/generic/bound/Geq.h, 271
keyvalue/key/generic/bound/Greater.h, 271
keyvalue/key/generic/bound/Leq.h, 272
keyvalue/key/generic/bound/Less.h, 273
keyvalue/key/generic/bound/NoBound.h, 273
keyvalue/key/generic/Bounded.h, 274
keyvalue/key/generic/Matrix.h, 276
keyvalue/key/generic/monotone/ Directory Reference, 37
keyvalue/key/generic/monotone/Decreasing.h, 278
keyvalue/key/generic/monotone/Increasing.h, 279
keyvalue/key/generic/monotone/NonMonotone.h, 279
keyvalue/key/generic/monotone/StrictlyDecreasing.h, 280
keyvalue/key/generic/monotone/StrictlyIncreasing.h, 280
keyvalue/key/generic/MonotoneBoundedVector.h, 281
keyvalue/key/generic/Positive.h, 281
keyvalue/key/generic/Single.h, 282
keyvalue/key/generic/StrictlyPositive.h, 285
keyvalue/key/generic/Vector.h, 258
keyvalue/key/Key.h, 286
keyvalue/key/map/ Directory Reference, 34
keyvalue/key/map/Default.h, 287
keyvalue/key/map/FlagMap.h, 288
keyvalue/key/map/NoMap.h, 289
keyvalue/key/map/ObjectMap.h, 291
keyvalue/key/map/PartialMap.h, 292
keyvalue/key/specific/ Directory Reference, 40
keyvalue/key/specific/Export.h, 293
keyvalue/key/specific/Imports.h, 294
keyvalue/key/specific/ProcessNow.h, 295
keyvalue/key/specific/Processor.h, 296
keyvalue/key/specific/VectorOutput.h, 299
keyvalue/key/Traits.h, 300
keyvalue/keyvalue.h, 301
keyvalue/mngt/ Directory Reference, 36
keyvalue/mngt/Builder.h, 301
keyvalue/mngt/Calculator.h, 302
keyvalue/mngt/Command.h, 303
keyvalue/mngt/DataSet.h, 305
keyvalue/mngt/DeclareBuilder.h, 306
keyvalue/mngt/DeclareCalculator.h, 306
keyvalue/mngt/process.h, 306
keyvalue/mngt/Processor.h, 297
keyvalue/mngt/ProcessorMngr.h, 307
keyvalue/mngt/Repository.h, 309
keyvalue/pattern/ Directory Reference, 38
keyvalue/pattern/KeyInSingle.h, 310
keyvalue/pattern/KeysInMatrix.h, 311
keyvalue/pattern/KeysInVector.h, 311
keyvalue/pattern/Pattern.h, 312
keyvalue/pattern/Table.h, 314
keyvalue/sys/ Directory Reference, 42
keyvalue/sys/exception/ Directory Reference, 24
keyvalue/sys/exception/Exception.h, 315
keyvalue/sys/exception/KV_ASSERT.h, 316
keyvalue/sys/logger/ Directory Reference, 32
keyvalue/sys/logger/ConsoleColors.h, 317
keyvalue/sys/logger/ConsoleLogger.h, 318
keyvalue/sys/logger/FileLogger.h, 319
keyvalue/sys/logger/Logger.h, 320
keyvalue/sys/logger/LoggerImpl.h, 322
keyvalue/sys/logger/policy/ Directory Reference, 39
keyvalue/sys/logger/policy/AddPrefix.h, 323
keyvalue/sys/logger/policy/ForwardToGlobalLogger.h, 324
keyvalue/sys/logger/policy/IgnoreColor.h, 326
keyvalue/sys/logger/policy/IgnoreFailure.h, 327
keyvalue/sys/logger/policy/IgnorePrefix.h, 328
keyvalue/sys/logger/policy/XtermColor.h, 329
keyvalue/sys/logger/StdLogger.h, 330
keyvalue/sys/logger/WindowsConsole.h, 330
keyvalue/sys/logger/XtermConsole.h, 332
keyvalue/sys/message/ Directory Reference, 35
keyvalue/sys/message/Message.h, 333
keyvalue/sys/message/MessageImpl.h, 334
keyvalue/util/ Directory Reference, 43
keyvalue/util/Global.h, 252
keyvalue/util/IsBasic.h, 336
keyvalue/util/Lexical.h, 337
keyvalue/util/NullDeleter.h, 338
keyvalue/util/Saver.h, 338
keyvalue/util/tenor2ptime.h, 339
keyvalue/util/Util.h, 340
keyvalue/value/ Directory Reference, 44
keyvalue/value/Matrix.h, 276
keyvalue/value/Nothing.h, 341
keyvalue/value/Parent.h, 342
keyvalue/value/Result.h, 343
keyvalue/value/Single.h, 283
keyvalue/value/TypeName.h, 344
keyvalue/value/Value.h, 346
keyvalue/value/Variant.h, 347
keyvalue/value/Vector.h, 259
keyvalue::Bridge, 67
  getCompleteInfo, 68
  getCoreLibraryName, 68
  getSimpleInfo, 68
keyvalue::Builder, 68
  getName, 71
  getObject, 71
  getResult, 71
keyvalue::BuilderFrom, 72
  getObject, 72
keyvalue::BuilderFromVariant, 73

build, 74
getObject, 73
keyvalue::Calculator, 74
getName, 76
getResult, 77
getValue, 77
keyvalue::Command, 83
getCommandName, 84
getName, 84
getResult, 84
keyvalue::DataSet, 85
add, 87
DataSet, 87
find, 87
getGraph, 89
getName, 87
getValue, 88
mustUpdate, 89
operator<<, 90
process, 88
keyvalue::DataSet::Graph, 113
keyvalue::DataSet::Record, 188
keyvalue::exception, 50
operator<<, 50
operator&, 50
keyvalue::exception::Exception, 94
Exception, 96
getMessage, 96
what, 96
keyvalue::exception::ExceptionImpl, 96
getMessage, 98
operator&, 98
what, 98
keyvalue::frontend, 51
keyvalue::frontend::FrontEnd, 107
getPath, 107
lexicalToolKit, 107
keyvalue::frontend::LexicalToolKit, 137
convert, 139
endInitList, 140
IO, 138
isEnabled, 139
LexicalToolKit, 138
set, 138
keyvalue::frontend::LexicalToolKit::Failure, 100
keyvalue::frontend::LexicalToolKit::Helper, 115
keyvalue::frontend::Queue, 186
isEmpty, 187
pop, 187
putBack, 188
remove, 187
keyvalue::key, 52
keyvalue::key::Bounded, 64
Bounded, 66
checkOutput, 66
getName, 67
setName, 67
keyvalue::key::Checker, 77
keyvalue::key::Checker< ConverterType, value::Matrix >, 78
checkInput, 79
checkOutput, 79
checkSize, 79
checkSoFar, 79
getName, 79
keyvalue::key::Checker< ConverterType, value::Single >, 80
checkInput, 80
checkOutput, 81
checkSoFar, 80
getName, 81
keyvalue::key::Checker< ConverterType, value::Vector >, 81
checkInput, 82
checkOutput, 82
checkSize, 82
checkSoFar, 82
getName, 82
keyvalue::key::Decreasing, 90
check, 90
getName, 91
keyvalue::key::Default, 91
keyvalue::key::DefaultMap, 91
keyvalue::key::Device, 92
get, 93
getName, 93
setName, 94
keyvalue::key::Export, 99
Export, 100
getName, 100
setName, 100
keyvalue::key::FlagMap, 104
get, 105
getName, 105
map, 105
keyvalue::key::Geq, 108
check, 109
Geq, 108
getBound, 109
getName, 109
keyvalue::key::Global, 112
getName, 113
Global, 113
setName, 113
keyvalue::key::Greater, 114
check, 115
getBound, 115
getName, 115

Greater, 115
keyvalue::key::Imports, 118
getName, 120
Imports, 120
setName, 120
keyvalue::key::Increasing, 120
check, 121
getName, 121
keyvalue::key::Key, 122
getName, 124
Key, 123
setName, 124
keyvalue::key::Leq, 132
check, 133
getBound, 133
getName, 133
Leq, 133
keyvalue::key::Less, 133
check, 134
getBound, 134
getName, 135
Less, 134
keyvalue::key::Level, 135
getName, 136
Level, 136
setName, 136
keyvalue::key::Matrix, 145
checkSize, 147
getName, 148
Matrix, 147
setName, 148
keyvalue::key::MonotoneBoundedVector, 160
checkSize, 163
checkSoFar, 163
getName, 164
map, 164
MonotoneBoundedVector, 163
setName, 164
keyvalue::key::NoBound, 165
check, 165
getBound, 165
getName, 166
keyvalue::key::NoMap, 166
getName, 167
map, 167
keyvalue::key::NonMonotone, 167
check, 168
getName, 168
keyvalue::key::ObjectMap, 169
getName, 170
map, 170
keyvalue::key::PartialMap, 172
get, 174
getName, 174
map, 173
keyvalue::key::Positive, 176
checkOutput, 178
getName, 178
Positive, 178
setName, 178
keyvalue::key::ProcessNow, 179
getName, 180
ProcessNow, 180
setName, 180
keyvalue::key::Processor, 182
getName, 183
Processor, 183
setName, 183
keyvalue::key::Single, 196
getName, 197
setName, 197
Single, 197
keyvalue::key::StdMatrix, 203
getOutput, 206
InputType_, 205
insert, 205
isEmpty, 205
mustUpdate, 206
OutputType_, 205
pop, 205
StdMatrix, 205
keyvalue::key::StdSingle, 206
getOutput, 209
InputType_, 207
insert, 208
isEmpty, 208
mustUpdate, 209
OutputType_, 207
pop, 208
StdSingle, 208
keyvalue::key::StdVector, 209
getOutput, 211
InputType_, 210
insert, 211
isEmpty, 211
mustUpdate, 211
OutputType_, 210
pop, 211
StdVector, 211
keyvalue::key::StrictlyDecreasing, 212
check, 212
getName, 212
keyvalue::key::StrictlyIncreasing, 213
check, 213
getName, 213
keyvalue::key::StrictlyPositive, 214
checkOutput, 215
getName, 215

setName, 216
StrictlyPositive, 215
keyvalue::key::Traits, 218
getName, 220
setName, 220
Traits, 220
keyvalue::key::Vector, 234
checkSize, 236
getName, 236, 237
setName, 237
Vector, 236
keyvalue::key::VectorOutput, 237
Flag, 238
get, 238
getName, 239
setName, 239
keyvalue::key::VectorOutputBase, 239
Flag, 240
keyvalue::logger, 55
Color, 56
resetGlobal, 56
keyvalue::logger::AddPrefix, 63
keyvalue::logger::ConsoleLogger, 85
keyvalue::logger::FileLogger, 101
getLevel, 102
log, 103
send, 102
sendHeader, 103
setLevel, 103
keyvalue::logger::ForwardToGlobalLogger, 106
apply, 106
forward, 107
keyvalue::logger::IgnoreColor, 116
keyvalue::logger::IgnoreFailure, 117
apply, 118
keyvalue::logger::IgnorePrefix, 118
keyvalue::logger::Logger, 140
getLevel, 141
log, 141
setLevel, 141
keyvalue::logger::LoggerImpl, 141
getLevel, 143
log, 144
process, 144
send, 144
sendHeader, 144
setLevel, 143
keyvalue::logger::WindowsConsole, 241
getLevel, 243
log, 243
send, 242
sendHeader, 243
setLevel, 243
keyvalue::logger::XtermColor, 244
keyvalue::logger::XtermConsole, 244
getLevel, 246
log, 247
send, 246
sendHeader, 247
setLevel, 247
keyvalue::logger::XtermConsole::FileRaii, 103
keyvalue::Message, 153
getColor, 155
getLevel, 155
getPrefix, 155
getString, 155
Message, 155
operator&, 155
keyvalue::MessageImpl, 156
getColor, 159
getLevel, 159
getPrefix, 159
getString, 159
MessageImpl, 158
operator&, 159
keyvalue::pattern, 56
getKey, 57
isKey, 57
keyvalue::pattern::KeyInSingle, 124
isEmpty, 126
parse, 126
pop, 126
keyvalue::pattern::KeysInMatrix, 127
check, 129
isEmpty, 129
parse, 128
pop, 128
keyvalue::pattern::KeysInVector, 129
isEmpty, 131
parse, 131
pop, 131
keyvalue::pattern::Pattern, 174
isEmpty, 176
parse, 176
pop, 176
keyvalue::pattern::Pattern::QueueRaii, 188
QueueRaii, 188
keyvalue::pattern::Table, 216
isEmpty, 218
parse, 217
pop, 217
keyvalue::Processor, 180
getName, 181
getResult, 181
keyvalue::ProcessorInstantiator, 184
getInstance, 184
keyvalue::ProcessorMngr, 184
add, 185

dummy_, 186
getCmdList, 186
getCommand, 186
getProcessor, 185
keyvalue::Repository, 189
add, 190
clear, 191
erase, 191
find, 191
getDataSet, 191
getList, 192
getSize, 191
keyvalue::Repository::NotFound, 168
keyvalue::tag, 58
keyvalue::util, 58
Lexical, 59
tenor2ptime, 59
keyvalue::util::Global, 109
get, 111
getInstance, 111
Global, 110, 111
set, 111
keyvalue::util::IsBasic, 121
keyvalue::util::IsBasicOrEnum, 122
keyvalue::util::NullDeleter, 169
keyvalue::util::Saver, 194
getSaved, 196
Saver, 195
keyvalue::value, 60
intrusive_ptr_add_ref, 61
intrusive_ptr_release, 61
operator<<, 62
keyvalue::value::Matrix, 148
getNCols, 151
getNRows, 151
intrusive_ptr_add_ref, 152
intrusive_ptr_release, 152
Matrix, 150
operator(), 151
operator==, 152
resize, 151
transposeVector, 152
keyvalue::value::Nothing, 169
keyvalue::value::Parent, 171
keyvalue::value::Result, 192
getObjectPtr, 194
getValue, 194
operator=, 193
Result, 193
keyvalue::value::Single, 198
getNCols, 202
getNRows, 202
getSize, 200
operator(), 201, 202
operator=, 200
operator==, 202
resize, 201
Single, 200
transpose, 202
transposeVector, 203
keyvalue::value::TypeName, 221
keyvalue::value::Value, 221
get, 223
operator=, 223
operator==, 223
Value, 222
keyvalue::value::Variant, 223
get, 226, 227
is, 226
operator<<, 227
operator=, 226
Variant, 225
keyvalue::value::Vector, 228
getNCols, 232
getNRows, 232
getSize, 231
operator(), 232, 233
operator==, 233
resize, 231, 233
transpose, 232
transposeVector, 234
Vector, 231
KV_ASSERT
KV_ASSERT.h, 317
KV_ASSERT.h
KV_ASSERT, 317
KV_LEXICAL_TOOL_KIT_PAIR
LexicalToolKit.h, 264

Leq
keyvalue::key::Leq, 133
Less
keyvalue::key::Less, 134
Level
keyvalue::key::Level, 136
Lexical
keyvalue::util, 59
LexicalToolKit
keyvalue::frontend::LexicalToolKit, 138
lexicalToolKit
keyvalue::frontend::FrontEnd, 107
LexicalToolKit.h
KV_LEXICAL_TOOL_KIT_PAIR, 264
log
keyvalue::logger::FileLogger, 103
keyvalue::logger::Logger, 141
keyvalue::logger::LoggerImpl, 144
keyvalue::logger::WindowsConsole, 243

keyvalue::logger::XtermConsole, 247
Logic, 145
LogicError, 145

map
keyvalue::key::FlagMap, 105
keyvalue::key::MonotoneBoundedVector, 164
keyvalue::key::NoMap, 167
keyvalue::key::ObjectMap, 170
keyvalue::key::PartialMap, 173
Matrix
keyvalue::key::Matrix, 147
keyvalue::value::Matrix, 150
Message
keyvalue::Message, 155
MessageImpl
keyvalue::MessageImpl, 158
MonotoneBoundedVector
keyvalue::key::MonotoneBoundedVector, 163
mustUpdate
keyvalue::DataSet, 89
keyvalue::key::StdMatrix, 206
keyvalue::key::StdSingle, 209
keyvalue::key::StdVector, 211

ObjectPtr, 171
operator<<
keyvalue, 49
keyvalue::DataSet, 90
keyvalue::exception, 50
keyvalue::value, 62
keyvalue::value::Variant, 227
operator()
keyvalue::value::Matrix, 151
keyvalue::value::Single, 201, 202
keyvalue::value::Vector, 232, 233
operator=
keyvalue::value::Result, 193
keyvalue::value::Single, 200
keyvalue::value::Value, 223
keyvalue::value::Variant, 226
operator==
keyvalue::value::Matrix, 152
keyvalue::value::Single, 202
keyvalue::value::Value, 223
keyvalue::value::Vector, 233
operator&
keyvalue::exception, 50
keyvalue::exception::ExceptionImpl, 98
keyvalue::Message, 155
keyvalue::MessageImpl, 159
OutputType_
keyvalue::key::StdMatrix, 205
keyvalue::key::StdSingle, 207
keyvalue::key::StdVector, 210

parse
keyvalue::pattern::KeyInSingle, 126
keyvalue::pattern::KeysInMatrix, 128
keyvalue::pattern::KeysInVector, 131
keyvalue::pattern::Pattern, 176
keyvalue::pattern::Table, 217
pop
keyvalue::frontend::Queue, 187
keyvalue::key::StdMatrix, 205
keyvalue::key::StdSingle, 208
keyvalue::key::StdVector, 211
keyvalue::pattern::KeyInSingle, 126
keyvalue::pattern::KeysInMatrix, 128
keyvalue::pattern::KeysInVector, 131
keyvalue::pattern::Pattern, 176
keyvalue::pattern::Table, 217
Positive
keyvalue::key::Positive, 178
process
keyvalue, 49
keyvalue::DataSet, 88
keyvalue::logger::LoggerImpl, 144
ProcessNow
keyvalue::key::ProcessNow, 180
Processor
keyvalue::key::Processor, 183
putBack
keyvalue::frontend::Queue, 188

QueueRaii
keyvalue::pattern::Pattern::QueueRaii, 188

remove
keyvalue::frontend::Queue, 187
Report, 189
resetGlobal
keyvalue::logger, 56
resize
keyvalue::value::Matrix, 151
keyvalue::value::Single, 201
keyvalue::value::Vector, 231, 233
Result
keyvalue::value::Result, 193
RuntimeError, 194

Saver
keyvalue::util::Saver, 195
send
keyvalue::logger::FileLogger, 102
keyvalue::logger::LoggerImpl, 144
keyvalue::logger::WindowsConsole, 242
keyvalue::logger::XtermConsole, 246

sendHeader
keyvalue::logger::FileLogger, 103
keyvalue::logger::LoggerImpl, 144
keyvalue::logger::WindowsConsole, 243
keyvalue::logger::XtermConsole, 247
set
keyvalue::frontend::LexicalToolKit, 138
keyvalue::util::Global, 111
setLevel
keyvalue::logger::FileLogger, 103
keyvalue::logger::Logger, 141
keyvalue::logger::LoggerImpl, 143
keyvalue::logger::WindowsConsole, 243
keyvalue::logger::XtermConsole, 247
setName
keyvalue::key::Bounded, 67
keyvalue::key::Device, 94
keyvalue::key::Export, 100
keyvalue::key::Global, 113
keyvalue::key::Imports, 120
keyvalue::key::Key, 124
keyvalue::key::Level, 136
keyvalue::key::Matrix, 148
keyvalue::key::MonotoneBoundedVector, 164
keyvalue::key::Positive, 178
keyvalue::key::ProcessNow, 180
keyvalue::key::Processor, 183
keyvalue::key::Single, 197
keyvalue::key::StrictlyPositive, 216
keyvalue::key::Traits, 220
keyvalue::key::Vector, 237
keyvalue::key::VectorOutput, 239
Single
keyvalue::key::Single, 197
keyvalue::value::Single, 200
StdMatrix
keyvalue::key::StdMatrix, 205
StdSingle
keyvalue::key::StdSingle, 208
StdVector
keyvalue::key::StdVector, 211
StrictlyPositive
keyvalue::key::StrictlyPositive, 215

tenor2ptime
keyvalue::util, 59
Traits
keyvalue::key::Traits, 220
transpose
keyvalue::value::Single, 202
keyvalue::value::Vector, 232
transposeVector
keyvalue::value::Matrix, 152
keyvalue::value::Single, 203
keyvalue::value::Vector, 234

Value
keyvalue::value::Value, 222
Variant
keyvalue::value::Variant, 225
Vector
keyvalue::key::Vector, 236
keyvalue::value::Vector, 231

Warning, 240
what
keyvalue::exception::Exception, 96
keyvalue::exception::ExceptionImpl, 98