HP Vertica C++ SDK Documentation Version 6.1.x

Wed May 29 2013

Contents

Namespace Index **1**
Namespace List . . . 1

Hierarchical Index **3**
Class Hierarchy . . . 3

Class Index **5**
Class List . . . 5

File Index **9**
File List . . . 9

Namespace Documentation **11**
Basics Namespace Reference . . . 11
Detailed Description . . . 11
Vertica Namespace Reference . . . 12
Detailed Description . . . 17
Typedef Documentation . . . 17
BaseDataOID . . . 17
DateADT . . . 17
Interval . . . 17
IntervalYM . . . 18
TimeADT . . . 18
Timestamp . . . 18
TimestampTz . . . 18
TimeTzADT . . . 18
Enumeration Type Documentation . . . 18
InputState . . . 18
StreamState . . . 19
volatility . . . 19
Function Documentation . . . 19
describeIntervalTypeMod . . . 20
getDateFromUnixTime . . . 20
getGMTTz . . . 20
getTime . . . 20
getTimeFromUnixTime . . . 20
getTimestampFromUnixTime . . . 20
getTimestampTzFromUnixTime . . . 20
getTimeTz . . . 20

getUnixTimeFromDate . . . 20
getUnixTimeFromTime . . . 21
getUnixTimeFromTimestamp . . . 21
getZoneTz . . . 21
log . . . 21
setTime . . . 21
setTimeTz . . . 21
vsmemcpy . . . 21

Class Documentation **23**
Basics::BigInt Struct Reference . . . 23
Detailed Description . . . 25
Member Function Documentation . . . 25
isEqualNN . . . 25
isZero . . . 26
numericToFloat . . . 26
setFromFloat . . . 26
ucompareNN . . . 26
Basics::BigInt::long_double_parts Union Reference . . . 26
Detailed Description . . . 27
Basics::FiveToScale Struct Reference . . . 27
Detailed Description . . . 27
FIunion Union Reference . . . 27
Vertica::AggregateFunction Class Reference . . . 27
Detailed Description . . . 28
Member Function Documentation . . . 28
aggregateArrs . . . 28
combine . . . 29
destroy . . . 29
initAggregate . . . 29
setup . . . 29
terminate . . . 29
updateCols . . . 30
Vertica::AggregateFunctionFactory Class Reference . . . 30
Detailed Description . . . 31
Member Enumeration Documentation . . . 31
UDXType . . . 31
Member Function Documentation . . . 31
createAggregateFunction . . . 31
getIntermediateTypes . . . 32
getParameterType . . . 32
getPerInstanceResources . . . 32
getPrototype . . . 32
getReturnType . . . 32
getUDXFactoryType . . . 33
Vertica::AnalyticFunction Class Reference . . . 33
Detailed Description . . . 34

Member Function Documentation . . . 34
cancel . . . 34
destroy . . . 34
isCanceled . . . 34
processPartition . . . 34
setup . . . 35
Vertica::AnalyticFunctionFactory Class Reference . . . 35
Detailed Description . . . 36
Member Enumeration Documentation . . . 36
UDXType . . . 36
Member Function Documentation . . . 36
createAnalyticFunction . . . 36
getParameterType . . . 36
getPerInstanceResources . . . 36
getPrototype . . . 37
getReturnType . . . 37
getUDXFactoryType . . . 37
Vertica::AnalyticPartitionReader Class Reference . . . 38
Detailed Description . . . 40
Member Function Documentation . . . 41
addCol . . . 41
getBoolPtr . . . 41
getBoolRef . . . 41
getColPtr . . . 41
getColRef . . . 42
getDatePtr . . . 42
getDateRef . . . 42
getFloatPtr . . . 43
getFloatRef . . . 43
getIntervalPtr . . . 43
getIntervalRef . . . 43
getIntervalYMPtr . . . 44
getIntervalYMRef . . . 44
getIntPtr . . . 44
getIntRef . . . 45
getNumCols . . . 45
getNumericPtr . . . 45
getNumericRef . . . 46
getNumRows . . . 46
getStringPtr . . . 46
getStringRef . . . 46
getTimePtr . . . 47
getTimeRef . . . 47
getTimestampPtr . . . 47
getTimestampRef . . . 48
getTimestampTzPtr . . . 48
getTimestampTzRef . . . 48

getTimeTzPtr . . . 49
getTimeTzRef . . . 49
getTypeMetaData . . . 49
getTypeMetaData . . . 49
isNull . . . 49
readNextBlock . . . 50
Vertica::AnalyticPartitionWriter Class Reference . . . 50
Detailed Description . . . 52
Member Function Documentation . . . 52
addCol . . . 52
copyFromInput . . . 52
getColPtr . . . 52
getColRef . . . 53
getNumCols . . . 53
getNumRows . . . 53
getTypeMetaData . . . 53
getTypeMetaData . . . 53
getWriteableBlock . . . 54
setInt . . . 54
setNull . . . 54
Vertica::BlockReader Class Reference . . . 54
Detailed Description . . . 57
Member Function Documentation . . . 57
addCol . . . 57
getBoolPtr . . . 57
getBoolRef . . . 57
getColPtr . . . 58
getColRef . . . 58
getDatePtr . . . 58
getDateRef . . . 58
getFloatPtr . . . 59
getFloatRef . . . 59
getIntervalPtr . . . 59
getIntervalRef . . . 60
getIntervalYMPtr . . . 60
getIntervalYMRef . . . 60
getIntPtr . . . 60
getIntRef . . . 61
getNumCols . . . 61
getNumericPtr . . . 61
getNumericRef . . . 62
getNumRows . . . 62
getStringPtr . . . 62
getStringRef . . . 62
getTimePtr . . . 63
getTimeRef . . . 63
getTimestampPtr . . . 63

getTimestampRef . . . 64
getTimestampTzPtr . . . 64
getTimestampTzRef . . . 64
getTimeTzPtr . . . 64
getTimeTzRef . . . 65
getTypeMetaData . . . 65
getTypeMetaData . . . 65
isNull . . . 65
next . . . 66
Vertica::BlockWriter Class Reference . . . 66
Detailed Description . . . 68
Member Function Documentation . . . 68
addCol . . . 68
getColPtr . . . 68
getColRef . . . 68
getNumCols . . . 69
getNumericRef . . . 69
getNumRows . . . 69
getStringRef . . . 69
getTypeMetaData . . . 69
getTypeMetaData . . . 69
next . . . 70
setBool . . . 70
setDate . . . 70
setFloat . . . 70
setInt . . . 70
setInterval . . . 70
setIntervalYM . . . 71
setTime . . . 71
setTimestamp . . . 71
setTimestampTz . . . 71
setTimeTz . . . 72
Vertica::ColumnTypes Class Reference . . . 72
Detailed Description . . . 73
Vertica::DataBuffer Struct Reference . . . 73
Detailed Description . . . 73
Vertica::DefaultSourceIterator Class Reference . . . 74
Member Function Documentation . . . 74
createNextSource . . . 74
getNumberOfSources . . . 74
getSizeOfSource . . . 75
Vertica::FilterFactory Class Reference . . . 75
Detailed Description . . . 76
Member Enumeration Documentation . . . 76
UDXType . . . 76
Member Function Documentation . . . 76
getParameterType . . . 76

getPerInstanceResources . . . 76
getPrototype . . . 77
getReturnType . . . 77
getUDXFactoryType . . . 77
plan . . . 77
prepare . . . 78
Vertica::FIunion Union Reference . . . 78
Vertica::IntermediateAggs Class Reference . . . 78
Detailed Description . . . 81
Member Function Documentation . . . 81
addCol . . . 81
getBoolPtr . . . 81
getBoolRef . . . 81
getColPtr . . . 82
getColRef . . . 82
getDatePtr . . . 82
getDateRef . . . 82
getFloatPtr . . . 83
getFloatRef . . . 83
getIntervalPtr . . . 83
getIntervalRef . . . 83
getIntervalYMPtr . . . 84
getIntervalYMRef . . . 84
getIntPtr . . . 84
getIntRef . . . 85
getNumCols . . . 85
getNumericPtr . . . 85
getNumericRef . . . 85
getNumRows . . . 86
getStringPtr . . . 86
getStringRef . . . 86
getTimePtr . . . 86
getTimeRef . . . 87
getTimestampPtr . . . 87
getTimestampRef . . . 87
getTimestampTzPtr . . . 88
getTimestampTzRef . . . 88
getTimeTzPtr . . . 88
getTimeTzRef . . . 89
getTypeMetaData . . . 89
getTypeMetaData . . . 89
Vertica::IterativeSourceFactory Class Reference . . . 89
Detailed Description . . . 90
Member Enumeration Documentation . . . 90
UDXType . . . 90
Member Function Documentation . . . 90
getParameterType . . . 91

getPerInstanceResources . . . 91
getPrototype . . . 91
getReturnType . . . 91
getUDXFactoryType . . . 91
plan . . . 92
prepare . . . 92
Vertica::MultiPhaseTransformFunctionFactory Class Reference . . . 92
Member Enumeration Documentation . . . 93
UDXType . . . 93
Member Function Documentation . . . 93
getParameterType . . . 93
getPerInstanceResources . . . 93
getPhases . . . 94
getPrototype . . . 94
getReturnType . . . 94
getUDXFactoryType . . . 95
Vertica::MultipleIntermediateAggs Class Reference . . . 95
Detailed Description . . . 97
Member Function Documentation . . . 97
addCol . . . 97
getBoolPtr . . . 98
getBoolRef . . . 98
getColPtr . . . 98
getColRef . . . 98
getDatePtr . . . 99
getDateRef . . . 99
getFloatPtr . . . 99
getFloatRef . . . 99
getIntervalPtr . . . 100
getIntervalRef . . . 100
getIntervalYMPtr . . . 100
getIntervalYMRef . . . 101
getIntPtr . . . 101
getIntRef . . . 101
getNumCols . . . 102
getNumericPtr . . . 102
getNumericRef . . . 102
getNumRows . . . 102
getStringPtr . . . 103
getStringRef . . . 103
getTimePtr . . . 103
getTimeRef . . . 104
getTimestampPtr . . . 104
getTimestampRef . . . 104
getTimestampTzPtr . . . 104
getTimestampTzRef . . . 105
getTimeTzPtr . . . 105

getTimeTzRef . . . 105
getTypeMetaData . . . 106
getTypeMetaData . . . 106
isNull . . . 106
next . . . 106
Vertica::NodeSpecifyingPlanContext Class Reference . . . 106
Detailed Description . . . 107
Member Function Documentation . . . 107
getClusterNodes . . . 107
getReader . . . 107
getTargetNodes . . . 107
getWriter . . . 108
setTargetNodes . . . 108
Vertica::ParamReader Class Reference . . . 108
Detailed Description . . . 111
Member Function Documentation . . . 111
addCol . . . 111
addParameter . . . 111
getBoolPtr . . . 111
getBoolRef . . . 112
getColPtr . . . 112
getColRef . . . 112
getDatePtr . . . 112
getDateRef . . . 113
getFloatPtr . . . 113
getFloatRef . . . 113
getIntervalPtr . . . 114
getIntervalRef . . . 114
getIntervalYMPtr . . . 114
getIntervalYMRef . . . 114
getIntPtr . . . 115
getIntRef . . . 115
getNumCols . . . 115
getNumericPtr . . . 116
getNumericRef . . . 116
getNumRows . . . 116
getStringPtr . . . 116
getStringRef . . . 117
getTimePtr . . . 117
getTimeRef . . . 117
getTimestampPtr . . . 118
getTimestampRef . . . 118
getTimestampTzPtr . . . 118
getTimestampTzRef . . . 119
getTimeTzPtr . . . 119
getTimeTzRef . . . 119
getTypeMetaData . . . 119

getTypeMetaData . . . 120
Member Data Documentation . . . 120
paramNameToIndex . . . 120
Vertica::ParamWriter Class Reference . . . 120
Detailed Description . . . 123
Member Function Documentation . . . 124
addCol . . . 124
addParameter . . . 124
getBoolPtr . . . 124
getBoolRef . . . 124
getColPtr . . . 125
getColRef . . . 125
getDatePtr . . . 125
getDateRef . . . 125
getFloatPtr . . . 126
getFloatRef . . . 126
getIntervalPtr . . . 126
getIntervalRef . . . 127
getIntervalYMPtr . . . 127
getIntervalYMRef . . . 127
getIntPtr . . . 128
getIntRef . . . 128
getLongStringRef . . . 128
getNumCols . . . 129
getNumericPtr . . . 129
getNumericRef . . . 129
getNumRows . . . 129
getStringPtr . . . 129
getStringRef . . . 130
getTimePtr . . . 130
getTimeRef . . . 130
getTimestampPtr . . . 130
getTimestampRef . . . 131
getTimestampTzPtr . . . 131
getTimestampTzRef . . . 131
getTimeTzPtr . . . 132
getTimeTzRef . . . 132
getTypeMetaData . . . 132
getTypeMetaData . . . 132
setAllocator . . . 133
setBool . . . 133
setDate . . . 133
setFloat . . . 133
setInt . . . 133
setInterval . . . 134
setIntervalYM . . . 134
setTime . . . 134

setTimestamp . . . 134
setTimestampTz . . . 134
setTimeTz . . . 135
Member Data Documentation . . . 135
paramNameToIndex . . . 135
Vertica::ParserFactory Class Reference . . . 135
Detailed Description . . . 136
Member Enumeration Documentation . . . 136
UDXType . . . 136
Member Function Documentation . . . 137
getParameterType . . . 137
getParserReturnType . . . 137
getPerInstanceResources . . . 138
getPrototype . . . 138
getReturnType . . . 138
getUDXFactoryType . . . 138
plan . . . 139
prepare . . . 139
Vertica::PartitionOrderColumnInfo Struct Reference . . . 140
Detailed Description . . . 140
Vertica::PartitionReader Class Reference . . . 140
Detailed Description . . . 143
Member Function Documentation . . . 143
addCol . . . 143
getBoolPtr . . . 143
getBoolRef . . . 144
getColPtr . . . 144
getColRef . . . 144
getDatePtr . . . 144
getDateRef . . . 145
getFloatPtr . . . 145
getFloatRef . . . 145
getIntervalPtr . . . 146
getIntervalRef . . . 146
getIntervalYMPtr . . . 146
getIntervalYMRef . . . 146
getIntPtr . . . 147
getIntRef . . . 147
getNumCols . . . 148
getNumericPtr . . . 148
getNumericRef . . . 148
getNumRows . . . 148
getStringPtr . . . 148
getStringRef . . . 149
getTimePtr . . . 149
getTimeRef . . . 149
getTimestampPtr . . . 150

getTimestampRef . . . 150
getTimestampTzPtr . . . 150
getTimestampTzRef . . . 151
getTimeTzPtr . . . 151
getTimeTzRef . . . 151
getTypeMetaData . . . 151
getTypeMetaData . . . 152
isNull . . . 152
readNextBlock . . . 152
Vertica::PartitionWriter Class Reference . . . 152
Detailed Description . . . 154
Member Function Documentation . . . 154
addCol . . . 154
copyFromInput . . . 155
getColPtr . . . 155
getColRef . . . 155
getNumCols . . . 155
getNumRows . . . 155
getTypeMetaData . . . 156
getTypeMetaData . . . 156
getWriteableBlock . . . 156
setInt . . . 156
setNull . . . 156
Vertica::PerColumnParamReader Class Reference . . . 156
Detailed Description . . . 157
Member Function Documentation . . . 157
getColumnNames . . . 157
getColumnParamReader . . . 157
Vertica::PlanContext Class Reference . . . 157
Detailed Description . . . 158
Member Function Documentation . . . 158
getClusterNodes . . . 158
getReader . . . 158
getWriter . . . 158
Vertica::RejectedRecord Struct Reference . . . 158
Detailed Description . . . 159
Vertica::ScalarFunction Class Reference . . . 159
Detailed Description . . . 159
Member Function Documentation . . . 160
destroy . . . 160
processBlock . . . 160
setup . . . 160
Vertica::ScalarFunctionFactory Class Reference . . . 160
Detailed Description . . . 161
Member Enumeration Documentation . . . 162
UDXType . . . 162
Member Function Documentation . . . 162

createScalarFunction . . . 162
getParameterType . . . 162
getPerInstanceResources . . . 162
getPrototype . . . 163
getReturnType . . . 163
getUDXFactoryType . . . 163
Member Data Documentation . . . 163
vol . . . 163
Vertica::ServerInterface Class Reference . . . 163
Detailed Description . . . 165
Constructor & Destructor Documentation . . . 165
ServerInterface . . . 165
Member Function Documentation . . . 165
getCurrentNodeName . . . 165
getLocale . . . 165
getParamReader . . . 165
log . . . 165
setParamReader . . . 166
vlog . . . 166
Member Data Documentation . . . 166
allocator . . . 166
locale . . . 166
nodeName . . . 166
paramReader . . . 166
sqlName . . . 166
vlogPtr . . . 166
Vertica::SizedColumnTypes Class Reference . . . 167
Detailed Description . . . 171
Member Function Documentation . . . 171
addBinary . . . 171
addBinaryOrderColumn . . . 171
addBinaryPartitionColumn . . . 172
addBool . . . 172
addBoolOrderColumn . . . 172
addBoolPartitionColumn . . . 172
addChar . . . 172
addCharOrderColumn . . . 173
addCharPartitionColumn . . . 173
addDate . . . 173
addDateOrderColumn . . . 173
addDatePartitionColumn . . . 174
addFloat . . . 174
addFloatOrderColumn . . . 174
addFloatPartitionColumn . . . 174
addInt . . . 174
addInterval . . . 175
addIntervalOrderColumn . . . 175

addIntervalPartitionColumn . . . 175
addIntervalYM . . . 175
addIntervalYMOrderColumn . . . 176
addIntervalYMPartitionColumn . . . 176
addIntOrderColumn . . . 176
addIntPartitionColumn . . . 176
addLongVarbinary . . . 177
addLongVarbinaryOrderColumn . . . 177
addLongVarbinaryPartitionColumn . . . 177
addLongVarchar . . . 177
addLongVarcharOrderColumn . . . 178
addLongVarcharPartitionColumn . . . 178
addNumeric . . . 178
addNumericOrderColumn . . . 178
addNumericPartitionColumn . . . 179
addTime . . . 179
addTimeOrderColumn . . . 179
addTimePartitionColumn . . . 179
addTimestamp . . . 180
addTimestampOrderColumn . . . 180
addTimestampPartitionColumn . . . 180
addTimestampTz . . . 180
addTimestampTzOrderColumn . . . 181
addTimestampTzPartitionColumn . . . 181
addTimeTz . . . 181
addTimeTzOrderColumn . . . 181
addTimeTzPartitionColumn . . . 182
addUserDefinedType . . . 182
addVarbinary . . . 182
addVarbinaryOrderColumn . . . 182
addVarbinaryPartitionColumn . . . 183
addVarchar . . . 183
addVarcharOrderColumn . . . 183
addVarcharPartitionColumn . . . 183
getArgumentColumns . . . 184
getColumnName . . . 184
getColumnType . . . 184
getColumnType . . . 184
getOrderByColumns . . . 185
getPartitionByColumns . . . 185
isOrderByColumn . . . 185
isPartitionByColumn . . . 185
setPartitionOrderColumnIdx . . . 185
setPartitionOrderColumnIdx . . . 186
Vertica::SourceFactory Class Reference . . . 186
Detailed Description . . . 187
Member Enumeration Documentation . . . 187

UDXType . . . 187
Member Function Documentation . . . 187
getParameterType . . . 187
getPerInstanceResources . . . 188
getPrototype . . . 188
getReturnType . . . 188
getUDXFactoryType . . . 188
plan . . . 189
prepare . . . 189
prepareUDSources . . . 189
Vertica::SourceIterator Class Reference . . . 190
Detailed Description . . . 190
Member Function Documentation . . . 191
createNextSource . . . 191
getNumberOfSources . . . 191
getSizeOfSource . . . 191
Vertica::StreamWriter Class Reference . . . 191
Detailed Description . . . 193
Member Function Documentation . . . 193
addCol . . . 193
copyFromInput . . . 194
getColPtr . . . 194
getColRef . . . 194
getNumCols . . . 194
getNumRows . . . 195
getTypeMetaData . . . 195
getTypeMetaData . . . 195
getWriteableBlock . . . 195
setInt . . . 195
setNull . . . 195
Vertica::TransformFunction Class Reference . . . 195
Detailed Description . . . 196
Member Function Documentation . . . 196
cancel . . . 196
destroy . . . 196
isCanceled . . . 197
processPartition . . . 197
setup . . . 197
Vertica::TransformFunctionFactory Class Reference . . . 197
Detailed Description . . . 198
Member Enumeration Documentation . . . 198
UDXType . . . 198
Member Function Documentation . . . 198
createTransformFunction . . . 198
getParameterType . . . 199
getPerInstanceResources . . . 199
getPrototype . . . 199

getReturnType . . . 199
getUDXFactoryType . . . 200
Vertica::TransformFunctionPhase Class Reference . . . 200
Detailed Description . . . 200
Member Function Documentation . . . 201
createTransformFunction . . . 201
getReturnType . . . 201
Vertica::UDFilter Class Reference . . . 202
Detailed Description . . . 202
Member Function Documentation . . . 202
destroy . . . 202
process . . . 202
setup . . . 203
Vertica::UDLFactory Class Reference . . . 204
Member Enumeration Documentation . . . 204
UDXType . . . 204
Member Function Documentation . . . 205
getParameterType . . . 205
getPerInstanceResources . . . 205
getPrototype . . . 205
getReturnType . . . 205
getUDXFactoryType . . . 205
Vertica::UDParser Class Reference . . . 206
Detailed Description . . . 206
Member Function Documentation . . . 206
destroy . . . 206
getRejectedRecord . . . 207
process . . . 207
setup . . . 208
Member Data Documentation . . . 208
writer . . . 208
Vertica::UDSource Class Reference . . . 208
Detailed Description . . . 209
Member Function Documentation . . . 209
destroy . . . 209
getSize . . . 209
getUri . . . 209
process . . . 210
setup . . . 210
Vertica::UDXFactory Class Reference . . . 210
Detailed Description . . . 211
Member Enumeration Documentation . . . 211
UDXType . . . 211
Member Function Documentation . . . 212
getParameterType . . . 212
getPerInstanceResources . . . 212
getPrototype . . . 212

getReturnType . . . 212
getUDXFactoryType . . . 213
Vertica::UDXObject Class Reference . . . 213
Detailed Description . . . 214
Constructor & Destructor Documentation . . . 214
~UDXObject . . . 214
Member Function Documentation . . . 214
destroy . . . 214
setup . . . 214
Vertica::UDXObjectCancelable Class Reference . . . 214
Member Function Documentation . . . 215
cancel . . . 215
destroy . . . 215
isCanceled . . . 215
setup . . . 215
Vertica::UDxRegistrar Struct Reference . . . 215
Vertica::UnsizedUDSource Class Reference . . . 215
Detailed Description . . . 216
Member Function Documentation . . . 216
getUri . . . 216
Vertica::VerticaBlock Class Reference . . . 216
Detailed Description . . . 218
Member Function Documentation . . . 218
addCol . . . 218
getColPtr . . . 218
getColRef . . . 218
getNumCols . . . 218
getNumRows . . . 219
getTypeMetaData . . . 219
getTypeMetaData . . . 219
Vertica::VerticaBuildInfo Struct Reference . . . 219
Vertica::VerticaType Class Reference . . . 219
Detailed Description . . . 222
Member Function Documentation . . . 222
getUnderlyingType . . . 222
Vertica::VInterval Class Reference . . . 222
Detailed Description . . . 222
Member Function Documentation . . . 222
breakUp . . . 222
Vertica::VIntervalYM Class Reference . . . 223
Detailed Description . . . 223
Member Function Documentation . . . 223
breakUp . . . 223
Vertica::VNumeric Class Reference . . . 224
Detailed Description . . . 225
Constructor & Destructor Documentation . . . 225
VNumeric . . . 225

Member Function Documentation . . . 225
compare . . . 225
compareUnsigned . . . 225
copy . . . 226
copy . . . 226
toFloat . . . 226
Vertica::VResources Struct Reference . . . 226
Detailed Description . . . 226
Member Data Documentation . . . 227
nFileHandles . . . 227
scratchMemory . . . 227
Vertica::VString Class Reference . . . 227
Detailed Description . . . 228
Member Function Documentation . . . 228
compare . . . 228
copy . . . 228
copy . . . 228
copy . . . 229
copy . . . 229
copy . . . 229
data . . . 229
data . . . 229
equal . . . 230
isNull . . . 230
length . . . 230
str . . . 230
Vertica::VTAllocator Class Reference . . . 231
Detailed Description . . . 231
Member Function Documentation . . . 231
alloc . . . 231

File Documentation **233**
Vertica.h File Reference . . . 233
Detailed Description . . . 239
Macro Definition Documentation . . . 239
InlineAggregate . . . 239
RegisterFactory . . . 239

Index **240**

Namespace Index

Namespace List

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

Basics
Basic utilities mainly for data types 11
Vertica 12

Hierarchical Index

Class Hierarchy

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

- Basics::BigInt . . . 23
- Basics::BigInt::long_double_parts . . . 26
- Basics::FiveToScale . . . 27
- FIunion . . . 27
- Vertica::ColumnTypes . . . 72
- Vertica::DataBuffer . . . 73
- Vertica::FIunion . . . 78
- Vertica::PartitionOrderColumnInfo . . . 140
- Vertica::PerColumnParamReader . . . 156
- Vertica::PlanContext . . . 157
  - Vertica::NodeSpecifyingPlanContext . . . 106
- Vertica::RejectedRecord . . . 158
- Vertica::ServerInterface . . . 163
- Vertica::SizedColumnTypes . . . 167
- Vertica::SourceIterator . . . 190
  - Vertica::DefaultSourceIterator . . . 74
- Vertica::TransformFunctionPhase . . . 200
- Vertica::UDFilter . . . 202
- Vertica::UDParser . . . 206
- Vertica::UDXFactory . . . 210
  - Vertica::AggregateFunctionFactory . . . 30
  - Vertica::AnalyticFunctionFactory . . . 35
  - Vertica::MultiPhaseTransformFunctionFactory . . . 92
  - Vertica::ScalarFunctionFactory . . . 160
  - Vertica::TransformFunctionFactory . . . 197
  - Vertica::UDLFactory . . . 204
    - Vertica::FilterFactory . . . 75
    - Vertica::IterativeSourceFactory . . . 89
      - Vertica::SourceFactory . . . 186
    - Vertica::ParserFactory . . . 135
- Vertica::UDXObject . . . 213
  - Vertica::AggregateFunction . . . 27

Vertica::ScalarFunction 159
Vertica::UDXObjectCancelable 214
Vertica::AnalyticFunction 33
Vertica::TransformFunction 195
Vertica::UDxRegistrar 215
Vertica::UnsizedUDSource 215
Vertica::UDSource 208
Vertica::VerticaBlock 216
Vertica::BlockReader 54
Vertica::MultipleIntermediateAggs 95
Vertica::PartitionReader 140
Vertica::AnalyticPartitionReader 38
Vertica::BlockWriter 66
Vertica::IntermediateAggs 78
Vertica::ParamReader 108
Vertica::ParamWriter 120
Vertica::PartitionWriter 152
Vertica::AnalyticPartitionWriter 50
Vertica::StreamWriter 191
Vertica::VerticaBuildInfo 219
Vertica::VerticaType 219
Vertica::VInterval 222
Vertica::VIntervalYM 223
Vertica::VNumeric 224
Vertica::VResources 226
Vertica::VString 227
Vertica::VTAllocator 231

Class Index

Class List

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

Basics::BigInt 23
Basics::BigInt::long_double_parts 26
Basics::FiveToScale
    Used to scale by factors of 5 27
FIunion 27
Vertica::AggregateFunction 27
Vertica::AggregateFunctionFactory 30
Vertica::AnalyticFunction 33
Vertica::AnalyticFunctionFactory 35
Vertica::AnalyticPartitionReader 38
Vertica::AnalyticPartitionWriter 50
Vertica::BlockReader
    Iterator interface for reading rows in a Vertica block 54
Vertica::BlockWriter
    Iterator interface for writing rows to a Vertica block 66
Vertica::ColumnTypes
    Represents (unsized) types of the columns used as input/output of a User Defined Function/Transform Function 72
Vertica::DataBuffer 73
Vertica::DefaultSourceIterator 74
Vertica::FilterFactory 75
Vertica::FIunion 78
Vertica::IntermediateAggs
    : A wrapper around a single intermediate aggregate value 78
Vertica::IterativeSourceFactory 89
Vertica::MultiPhaseTransformFunctionFactory 92
Vertica::MultipleIntermediateAggs
    : A wrapper around multiple intermediate aggregates 95
Vertica::NodeSpecifyingPlanContext 106
Vertica::ParamReader
    : A wrapper around Parameters that have a name->value correspondence 108

Vertica::ParamWriter
Iterator interface for writing rows to a Vertica block 120
Vertica::ParserFactory 135
Vertica::PartitionOrderColumnInfo
Represents the partition by and order by column information for each phase in a multi-phase transform function 140
Vertica::PartitionReader 140
Vertica::PartitionWriter 152
Vertica::PerColumnParamReader
: A wrapper around a map from column to ParamReader 156
Vertica::PlanContext 157
Vertica::RejectedRecord 158
Vertica::ScalarFunction 159
Vertica::ScalarFunctionFactory 160
Vertica::ServerInterface 163
Vertica::SizedColumnTypes
Represents types and information to determine the size of the columns as input/output of a User Defined Function/Transform 167
Vertica::SourceFactory 186
Vertica::SourceIterator 190
Vertica::StreamWriter 191
Vertica::TransformFunction 195
Vertica::TransformFunctionFactory 197
Vertica::TransformFunctionPhase 200
Vertica::UDFilter 202
Vertica::UDLFactory 204
Vertica::UDParser 206
Vertica::UDSource 208
Vertica::UDXFactory 210
Vertica::UDXObject 213
Vertica::UDXObjectCancelable 214
Vertica::UDxRegistrar 215
Vertica::UnsizedUDSource 215
Vertica::VerticaBlock
: Represents an in-memory block of tuples 216
Vertica::VerticaBuildInfo 219
Vertica::VerticaType
Represents types of data that are passed into and returned back from user's code 219
Vertica::VInterval
Representation of an Interval in Vertica 222
Vertica::VIntervalYM
Representation of an IntervalYM in Vertica. An Interval can be broken up into years and months 223
Vertica::VNumeric
Representation of NUMERIC, fixed point data types in Vertica . . . 224
Vertica::VResources 226

Vertica::VString
Representation of a String in Vertica. All character data is internally encoded as UTF-8 characters and is not NULL terminated 227
Vertica::VTAllocator 231

File Index

File List

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

Vertica.h
Contains the classes needed to write User-Defined things in Vertica 233

Namespace Documentation

Basics Namespace Reference

basic utilities mainly for data types.

Classes

- struct BigInt
- struct FiveToScale

  Used to scale by factors of 5.

Functions

- bool **Divide32** (uint64 hi, uint64 lo, uint64 d, uint64 &_q, uint64 &_r)
- static int32 getNumericLength (int32 typmod)

  Get Numeric data length from typmod.
- int32 getNumericPrecision (int32 typmod)

  Get Numeric precision from typmod.
- int32 getNumericScale (int32 typmod)

  Get Numeric scale from typmod.
- int32 getNumericWordCount (int32 precision)

  Get Numeric word count from precision.
- bool isSimilarNumericTypmod (int32 a, int32 b)

  Return true if these have the same EE representation.

Variables

- const FiveToScale **fiveToScales** [ ]

Detailed Description

basic utilities mainly for data types.

Vertica Namespace Reference

Classes

- class AggregateFunction
- class AggregateFunctionFactory
- class AnalyticFunction
- class AnalyticFunctionFactory
- class AnalyticPartitionReader
- class AnalyticPartitionWriter
- class BlockReader

  Iterator interface for reading rows in a Vertica block.
- class BlockWriter

  Iterator interface for writing rows to a Vertica block.
- class ColumnTypes

  Represents (unsized) types of the columns used as input/output of a User Defined Function/Transform Function.
- struct DataBuffer
- class DefaultSourceIterator
- class FilterFactory
- union FIunion
- class IntermediateAggs

  : A wrapper around a single intermediate aggregate value
- class IterativeSourceFactory
- class MultiPhaseTransformFunctionFactory
- class MultipleIntermediateAggs

  : A wrapper around multiple intermediate aggregates
- class NodeSpecifyingPlanContext
- class ParamReader

  : A wrapper around Parameters that have a name->value correspondence
- class ParamWriter

  Iterator interface for writing rows to a Vertica block.
- class ParserFactory
- struct PartitionOrderColumnInfo

  Represents the partition by and order by column information for each phase in a multi-phase transform function.
- class PartitionReader
- class PartitionWriter
- class PerColumnParamReader

  : A wrapper around a map from column to ParamReader.
- class PlanContext

- struct RejectedRecord
- class ScalarFunction
- class ScalarFunctionFactory
- class ServerInterface
- class SizedColumnTypes

  Represents types and information to determine the size of the columns as input/output of a User Defined Function/Transform.
- class SourceFactory
- class SourceIterator
- class StreamWriter
- class TransformFunction
- class TransformFunctionFactory
- class TransformFunctionPhase
- class UDFilter
- class UDLFactory
- class UDParser
- class UDSource
- class UDXFactory
- class UDXObject
- class UDXObjectCancelable
- struct UDxRegistrar
- class UnsizedUDSource
- class VerticaBlock

  : Represents an in-memory block of tuples
- struct VerticaBuildInfo
- class VerticaType

  Represents types of data that are passed into and returned back from user's code.
- class VInterval

  Representation of an Interval in Vertica.
- class VIntervalYM

  Representation of an IntervalYM in Vertica. An Interval can be broken up into years and months.
- class VNumeric

  Representation of NUMERIC, fixed point data types in Vertica.
- struct VResources
- class VString

  Representation of a String in Vertica. All character data is internally encoded as UTF-8 characters and is not NULL terminated.
- class VTAllocator

Typedefs

- typedef unsigned long long int BaseDataOID
- typedef uint8 byte

  unsigned 8-bit integer
- typedef int64 DateADT
- typedef long double ifloat

  vertica 80-bit intermediate result
- typedef signed short int16

  signed 16-bit integer
- typedef signed int int32

  signed 32-bit integer
- typedef signed long long int64

  signed 64-bit integer
- typedef signed char int8

  8-bit integer
- typedef int64 Interval
- typedef int64 IntervalYM
- typedef unsigned long long int **Oid**
- typedef int64 TimeADT
- typedef int64 Timestamp
- typedef int64 TimestampTz

  Vertica timestamp data type.
- typedef int64 TimeTzADT
- typedef unsigned short uint16

  unsigned 16-bit integer
- typedef unsigned int uint32

  unsigned 32-bit integer
- typedef unsigned long long uint64

  unsigned 64-bit integer
- typedef unsigned char uint8

  unsigned 8-bit integer
- typedef uint8 vbool

  vertica 8-bit boolean (t,f,null)
- typedef double vfloat

  Represents Vertica 64-bit floating-point type for NULL checking use vfloatIsNull(), assignment to NULL use vfloat_null for NaN checking use vfloatIsNaN(), assignment to NaN use vfloat_NaN.
- typedef signed long long vint

  vertica 64-bit integer (not int64)
- typedef unsigned long long vpos

64-bit vertica position

- typedef uint32 vsize

  vertica 32-bit block size

- typedef void(∗ **VT_THROW_EXCEPTION**)(int errcode, const std::string &message, const char ∗filename, int lineno)

Enumerations

- enum **DTtype** {
  RESERV = 0, **MONTH** = 1, **YEAR** = 2, **DAY** = 3,
  NEG_INTERVAL = 4, **TZ** = 5, **DTZ** = 6, **DTZMOD** = 7,
  IGNORE_DTF = 8, **AMPM** = 9, **HOUR** = 10, **MINUTE** = 11,
  SECOND = 12, **DOY** = 13, **DOW** = 14, **UNITS** = 15,
  ADBC = 16, **AGO** = 17, **ISODATE** = 20, **ISOTIME** = 21,
  UNKNOWN_FIELD = 31 }
- enum InputState { **OK**, **END_OF_FILE** }
- enum StreamState {
  INPUT_NEEDED, **OUTPUT_NEEDED**, **DONE**, **REJECT**,
  KEEP_GOING }
- enum **strictness** { **DEFAULT_STRICTNESS**, **CALLED_ON_NULL_INPUT**, **RETURN_NULL_ON_NULL_INPUT**, **STRICT** }
- enum VBool {
  VFalse = 0, **VTrue** = 1, **VUnknown** = 2, **VFalse** = 0,
  VTrue = 1, **VUnknown** = 2 }

  Enumeration for Boolean data values.

- enum VBool {
  VFalse = 0, **VTrue** = 1, **VUnknown** = 2, **VFalse** = 0,
  VTrue = 1, **VUnknown** = 2 }

  Enumeration for Boolean data values.

- enum volatility { **DEFAULT_VOLATILITY**, **VOLATILE**, **IMMUTABLE**, **STABLE** }

Functions

- void DateADT **dateIn** (const char ∗str, bool report_error)
- DateADT **dateInFormatted** (const char ∗str, const std::string format, bool report_error)
- static void describeIntervalTypeMod (char ∗result, int32 typemod)
- void **dummy** ()
- static DateADT getDateFromUnixTime (time_t time)
- static int64 getGMTTz (TimeTzADT time)
- static uint64 getTime (TimeADT time)
- static TimeADT getTimeFromUnixTime (time_t time)

- static Timestamp getTimestampFromUnixTime (time_t time)
- static TimestampTz getTimestampTzFromUnixTime (time_t time)
- static int64 getTimeTz (TimeTzADT time)
- Oid **getUDTypeOid** (const char *typeName)
- Oid **getUDTypeUnderlyingOid** (Oid typeOid)
- static time_t getUnixTimeFromDate (DateADT date)
- static time_t getUnixTimeFromTime (TimeADT t)
- static time_t getUnixTimeFromTimestamp (Timestamp timestamp)
- static time_t **getUnixTimeFromTimestampTz** (TimestampTz timestamp)
- static int32 getZoneTz (TimeTzADT time)
- Interval **intervalIn** (const char *str, int32 typmod, bool report_error)
- bool **isUDType** (Oid typeOid)
- void log (const char *format,...) __attribute__((format(printf
- int **makeErrMsg** (std::stringstream &err_msg, const char *fmt,...)
- static TimeADT setTime (int64 time)
- static TimeTzADT setTimeTz (int64 time, int32 zone)
- TimeADT **timeIn** (const char *str, int32 typmod, bool report_error)
- TimeADT **timeInFormatted** (const char *str, int32 typmod, const std::string format, bool report_error)
- Timestamp **timestampIn** (const char *str, int32 typmod, bool report_error)
- Timestamp **timestampInFormatted** (const char *str, int32 typmod, const std::string format, bool report_error)
- TimestampTz **timestamptzIn** (const char *str, int32 typmod, bool report_error)
- TimestampTz **timestamptzInFormatted** (const char *str, int32 typmod, const std::string format, bool report_error)
- TimeTzADT **timetzIn** (const char *str, int32 typmod, bool report_error)
- TimeTzADT **timetzInFormatted** (const char *str, int32 typmod, const std::string format, bool report_error)
- bool **vfloatIsNaN** (const vfloat *f)
- bool **vfloatIsNaN** (const vfloat f)
- bool **vfloatIsNull** (const vfloat *vf)
- bool **vfloatIsNull** (const vfloat vf)
- void **vsmemclr** (void *dst, size_t len)
- void vsmemcpy (void *dst, const void *src, size_t len)
- bool vsmemne (const void *p1, const void *p2, size_t len)

  Version of memcmp for detecting not equal only. Inline function for comparing small things of arbitrary length.

Variables

- const byte byte_null = 0xff

  Indicates NULL byte.
- const char char_null = 0xff

  Indicates NULL char.
- const vbool vbool_false = VFalse

  Value for Boolean FALSE.
- const vbool vbool_null = VUnknown

  Value for Boolean NULL.
- const vbool vbool_true = VTrue

  Value for Boolean TRUE.
- const vfloat **vfloat_null** = vfn.vf
- union Vertica::FIunion **vfn** = {0x7fffffffffffffeLL}
- union FIunion **vfNaN** = {0xFFF8000000000000LL}
- const vint vint_null = 0x8000000000000000LL

  Value for Integer NULLs.
- VT_THROW_EXCEPTION **vt_throw_exception**

Detailed Description

Defines classes for Vertica User Defined Functions

Typedef Documentation

typedef unsigned long long int Vertica::BaseDataOID

The Postgres types understood by Vertica I/O, e.g. Load.cpp and Root.cpp See P-G/src/include/catalog/pg_type.h, which must match

typedef int64 Vertica::DateADT

The value in DateADT is the number of days

typedef int64 Vertica::Interval

Represents delta time.

The value in Interval is number of microseconds, which is limited to +/- 2^63-1 microseconds = 106751991 days 4 hours 54.775807 seconds or +/-9223372036854775807 months = 768614336404564650 years 7 months

Interval can be directly added/subtracted from Timestamp/TimestampTz to have a meaningful result

typedef int64 Vertica::IntervalYM

Represents delta time.

The value in IntervalYM is number of months

So Interval can be directly added/subtracted from Timestamp/TimestampTz to have a meaningful result, but IntervalYM cannot

typedef int64 Vertica::TimeADT

TimeADT represents time within a day

The value in TimeADT number of microseconds within 24 hours

typedef int64 Vertica::Timestamp

Represents absolute time and date.

The value in Timestamp is the number of microseconds since 2000-01-01 00:00:00 with no timezone implied

typedef int64 Vertica::TimestampTz

Vertica timestamp data type.

Represents absolute time and date with time zone.

The value in TimestampTz is the number of microseconds since 2000-01-01 00:00:00 GMT

typedef int64 Vertica::TimeTzADT

Represents time within a day in a timezone

The value in TimeADT consists of 2 parts:

1. The lower 24 bits (defined as ZoneFieldWidth) contains the timezone plus 24 hours, specified in seconds SQL-2008 limits the timezone itself to range between +/-14 hours

1. The rest of higher bits contains the time in GMT, value specified as number of microseconds within 24 hours

Enumeration Type Documentation

enum Vertica::InputState

InputState

Applies only to input streams; namely, UDFilter and UDParser.

OK Currently at the start of or in the middle of a stream.

END_OF_FILE Reached the end of the input stream. No further data is available. Returning a StreamState of INPUT_NEEDED at this point is invalid, as there is no more input.

enum Vertica::StreamState

StreamState

Indicates the state of a stream after process() has handled some input and some output data.

The different enum values have the following meanings:

INPUT_NEEDED Indicates that a stream is unable to continue without being given more input. It may not have consumed all of its available input yet. It does not need to have consumed every byte of input. Not valid for output-only streams, ie., UDSources.

OUTPUT_NEEDED Indicates that a stream is unable to write more output without being given a new or larger output buffer. Basically that it has "filled" the buffer as much as it is reasonably able to (which may be every byte full, some bytes full, even completely empty – in which case Vertica assumes that the UDL needs a larger buffer). Not valid for input-only streams, ie., UDParsers.

DONE The stream has completed; it will not be writing any more output nor consuming any more input.

REJECT Only valid for UDParsers. Indicates that the Parser has consumed exactly one row, and that that row is invalid and should be processed as a rejected row.

KEEP_GOING Not commonly used. The stream has neither filled all of its output buffer nor consumed all of its input buffer, but would like to yield to the server. Typically it has neither consumed data nor produced data. This state should be used instead of a "wait" loop; a stream that is waiting for some external operation to complete should periodically return KEEP_GOING rather than simply blocking forever.

See the UDSource, UDFilter, and UDParser classes for how these streams are used.

enum Vertica::volatility

Enums to allow programmatic specification of volatility and strictness

Function Documentation

static void Vertica::describeIntervalTypeMod (char * *result,* int32 *typemod*) `[inline],[static]`

Format interval "<subtype>" where <subtype> often starts with a space Call with char buf[64] to hold the result.

static DateADT Vertica::getDateFromUnixTime (time_t *time*) `[inline],[static]`

*Convert Unix time_t to Date

static int64 Vertica::getGMTTz (TimeTzADT *time*) `[inline],[static]`

Get the GMT time of the TimeTz value (in microseconds)

static uint64 Vertica::getTime (TimeADT *time*) `[inline],[static]`

Get the time from a Time value

static TimeADT Vertica::getTimeFromUnixTime (time_t *time*) `[inline],[static]`

*Convert Unix time_t to Time

static Timestamp Vertica::getTimestampFromUnixTime (time_t *time*) `[inline],[static]`

*Convert Unix time_t to Timestamp

static TimestampTz Vertica::getTimestampTzFromUnixTime (time_t *time*) `[inline],[static]`

*Convert Unix time_t to TimestampTz

static int64 Vertica::getTimeTz (TimeTzADT *time*) `[inline],[static]`

Get the time at the timezone specified in TimeTz value itself (in microseconds)

static time_t Vertica::getUnixTimeFromDate (DateADT *date*) `[inline],[static]`

Convert Date to Unix Time

static time_t Vertica::getUnixTimeFromTime (TimeADT *t*) `[inline],[static]`

Convert Time to Unix Time

static time_t Vertica::getUnixTimeFromTimestamp (Timestamp *timestamp*) `[inline],[static]`

Convert Timestamp to Unix time_t value (which is number of seconds since the Unix epoch) The internal representation of Timestamps is number of microseconds since the Postgres epoch date. This will truncate the timestamp to number of seconds (instead of microseconds)

static int32 Vertica::getZoneTz (TimeTzADT *time*) `[inline],[static]`

Get timezone from a TimeTz value (in seconds) (seconds field should be 0)

void Vertica::log (const char * *format,* ...)

Write a message to the vertica.log system log.

Parameters

format	a printf style format string specifying the log message format.

static TimeADT Vertica::setTime (int64 *time*) `[inline],[static]`

Produce a Time value

static TimeTzADT Vertica::setTimeTz (int64 *time,* int32 *zone*) `[inline],[static]`

Produce a TimeTz value from time (in microseconds) at a timezone (in seconds, seconds field must be 0)

void Vertica::vsmemcpy (void * *dst,* const void * *src,* size_t *len*) `[inline]`

Version of memcpy optimized for vertica's small data types. Inline function for copying small things of arbitrary length.

Class Documentation

Basics::BigInt Struct Reference

Classes

- union long_double_parts

Static Public Member Functions

- static uint64 accumulateNN (void *outBuf, const void *buf1, int nWords)

  Add number on to a temporary No null handling Returns carry.
- static uint64 addNN (void *outBuf, const void *buf1, const void *buf2, int nWords)

  Add together 2 numbers w/ same number of digits No null handling Returns carry.
- static void **castNumeric** (uint64 *wordso, int nwdso, int32 typmodo, uint64 *wordsi, int nwdsi, int32 typmodi)
- static bool **checkOverflowNN** (const void *po, int nwdso, int32 typmodo)
- static bool **checkOverflowNN** (const void *po, int nwo, int nwdso, int32 typmodo)
- static int compareNN (const void *buf1, const void *buf2, int nWords)

  Compare integers, return -1, 0, 1.
- static long double convertPosToFloatNN (const void *bbuf, int bwords)

  Convert Numeric to a float helper function Input should not be negative / null.
- static void copy (void *buf, const void *src, int nWords)

  cpoy nWords from src to buf
- static uint64 **divUnsignedN1** (void *qbuf, int qw, int round, const void *ubuf, int uw, uint64 v)
- static void **divUnsignedNN** (void *qbuf, int qw, int round, uint64 *rbuf, int rw, const void *ubuf, int uw, const void *vbuf, int vw)
- static void fromIntNN (void *buf, int nWords, int64 val)

  Load from 64-bit signed int (does not handle NULL inside)
- static void incrementNN (void *buf, int nWords)

  Increment by 1.

- static void invertSign (void *buf, int nWords)

  Invert the sign of a number, performed in place, using the invert and +1 method, turns out NULLs are OK in this sense.
- static bool isEqualNN (const void *buf1, const void *buf2, int nWords)

  Check integers for equality.
- static bool isNeg (const void *buf, int nWords)

  Check if integer is less than zero.
- static bool isNull (const void *buf, int nWords)

  Check an integer for NULL.
- static bool isZero (const void *buf, int nWords)

  Check an integer for 0.
- static void mulUnsignedN1 (void *obuf, const void *ubuf, int uw, uint64 v)

  Multiply, unsigned only. uw words by 1 word -> uw+1 words Output array must be uw+1 words long; may not overlap inputs.
- static void mulUnsignedNN (void *obuf, const void *buf1, int nw1, const void *buf2, int nw2)

  Multiply, unsigned only. Output array must be nw1+nw2 long; may not overlap inputs PERF NOTE: Operates like a school boy, could do Karatsuba (or better)
- static void numericDivide (const uint64 *pa, int nwdsa, int32 typmoda, const uint64 *pb, int nwdsb, int32 typmodb, uint64 *outNum, int nwdso, int32 typmodo)

  Divide Numerics Handles negative / null input.
- static void numericMultiply (const uint64 *pa, int nwdsa, const uint64 *pb, int nwdsb, uint64 *outNum, int nwdso)

  Multiply Numerics , result in outNum Handles negative / null input.
- static void **NumericRescaleDown** (uint64 *wordso, int nwdso, int32 typmodo, uint64 *wordsi, int nwdsi, int32 typmodi)
- static void **NumericRescaleSameScaleSmallerPrec** (uint64 *wordso, int nwdso, int32 typmodo, uint64 *wordsi, int nwdsi, int32 typmodi)
- static void **NumericRescaleUp** (uint64 *wordso, int nwdso, int32 typmodo, uint64 *wordsi, int nwdsi, int32 typmodi)
- static ifloat numericToFloat (const void *buf, int nwds, ifloat tenthtoscale)

  Convert Numeric to a float Handles negative / null input.
- static bool rescaleNumeric (void *out, int ow, int pout, int sout, void *in, int iw, int pin, int sin)

  Rescale a given numeric to a specific prec/scale/nwds The input should have minimal precision to avoid unnecessary overflow; for example, "0" should have precision 0 (as generated by charToNumeric). Accepts signed numerics. Will set its "in" argument to abs(in).
- static bool setFromFloat (void *bbuf, int bwords, const FiveToScale &fiveToScale, long double value, bool round)

  Convert floating point multiplied by 10^scale to an integer This truncates if round is false; otherwise the numeric result is rounded.

- static void setNull (void *buf, int nWords)

  Set an integer to NULL.
- static void **setNumericBoundsFromType** (uint64 *numUpperBound, uint64 *numLowerBound, int nwdso, int32 typmod)
- static void setZero (void *buf, int nWords)

  Set an integer to 0.
- static void shiftLeftNN (void *buf, unsigned nw, unsigned bitsToShift)

  Shift a BigInt to the left (<<) by the given number of bits. The given BigInt must be positive and non-null.
- static void shiftRightNN (void *buf, unsigned nw, unsigned bitsToShift)

  Shift a BigInt to the right (>>) by the given number of bits. The given BigInt must be positive and non-null.
- static void subNN (void *outBuf, const void *buf1, const void *buf2, int nWords)

  Subtract 2 numbers w/ same number of digits No null handling.
- static bool toIntNN (int64 &out, const void *buf, int nWords)

  Convert to int (return false if there was an overflow)
- static int ucompareNN (const void *buf1, const void *buf2, int nWords)
- static int usedWordsUnsigned (const void *buf, int nWords)

  Calculate number of words that are actually used to represent the value (amount left after stripping leading 0's)

Static Public Attributes

- static const int **powerOf10Sizes** [1200]
- static const uint64 *const **powerOf10Words** [1200]

Detailed Description

Holds integer utilities

Member Function Documentation

static bool Basics::BigInt::isEqualNN (const void * *buf1,* const void * *buf2,* int *nWords*) [inline], [static]

Check integers for equality.

Note

Optimized for small nWords; scan from end and break out of loop

static bool Basics::BigInt::isZero (const void * *buf,* int *nWords*) `[inline], [static]`

Check an integer for 0.

Note

Optimized for small nWords; scan from end and break out of loop

static ifloat Basics::BigInt::numericToFloat (const void * *buf,* int *nwds,* ifloat *tenthtoscale*) `[inline], [static]`

Convert Numeric to a float Handles negative / null input.

Parameters

tenthtoscale	10^-scale of the Numeric

static bool Basics::BigInt::setFromFloat (void * *bbuf,* int *bwords,* const FiveToScale & *fiveToScale,* long double *value,* bool *round*) `[inline], [static]`

Convert floating point multiplied by 10^scale to an integer This truncates if round is false; otherwise the numeric result is rounded.

Returns

false if high bits were lost, true if reasonable fidelity was achieved

static int Basics::BigInt::ucompareNN (const void * *buf1,* const void * *buf2,* int *nWords*) `[inline], [static]`

Compare integers, return -1, 0, 1

Basics::BigInt::long_double_parts Union Reference

Public Member Functions

- **long_double_parts** (long double v=0.0)

Public Attributes

- struct {

```
    unsigned short int exponent
    unsigned long long int mantissa
    unsigned short int remainder [3]
};
```

- long double **value**

Detailed Description

x86 only 80-bit extended precision float 64 bit mantissa, stored little endian (just like a 64-bit int) Note that the mantissa does not have an implicit "1" bit thus eliminating the concept of denormal numbers. 16 bit exponent, also little endian, within the next 64 bits. High bit is sign Lower 15 bits are "excess 16383" Example: 1.0 is stored as, in binary: mantissa: 1000000000000000000000000000000000000000000000000000000000000000, exponent: 0011111111111111

Basics::FiveToScale Struct Reference

Used to scale by factors of 5.

Public Attributes

- unsigned **scale**
- const uint64 * **value**
- unsigned **wordCount**

Detailed Description

Used to scale by factors of 5.

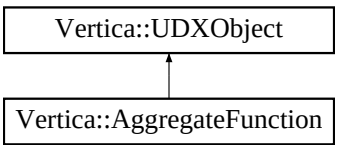
Flunion Union Reference

Public Attributes

- vfloat **vf**
- vint **vi**

Vertica::AggregateFunction Class Reference

Inheritance diagram for Vertica::AggregateFunction:



Public Member Functions

- virtual void aggregateArrs (ServerInterface &srvInterface, void **dstTuples, int doff, const void *arr, int stride, const void *rcounts, int rcstride, int count, IntermediateAggs &intAggs, std::vector< int > &intOffsets, BlockReader &arg_reader)=0
- virtual void combine (ServerInterface &srvInterface, IntermediateAggs &aggs_output, MultipleIntermediateAggs &aggs_other)=0
- virtual void destroy (ServerInterface &srvInterface, const SizedColumnTypes &argTypes)
- virtual void initAggregate (ServerInterface &srvInterface, IntermediateAggs &aggs)=0
- virtual void setup (ServerInterface &srvInterface, const SizedColumnTypes &argTypes)
- virtual void terminate (ServerInterface &srvInterface, BlockWriter &res_writer, IntermediateAggs &aggs)=0

Static Public Member Functions

- static void updateCols (BlockReader &arg_reader, char *arg, int count, IntermediateAggs &intAggs, char *aggPtr, std::vector< int > &intOffsets)

Detailed Description

Interface for User Defined Aggregate, the actual code to process a block of data coming in from the stream

Member Function Documentation

virtual void Vertica::AggregateFunction::aggregateArrs (ServerInterface & *srvInterface,* void ** *dstTuples,* int *doff,* const void * *arr,* int *stride,* const void * *rcounts,* int *rcstride,* int *count,* IntermediateAggs & *intAggs,* std::vector< int > & *intOffsets,* BlockReader & *arg_reader*) `[pure virtual]`

Called by the server to perform aggregation on multiple blocks of data

Note

User should not explicity implement this function. It is implemented by calling the InlineAggregate() macro. User should follow the convention of implementing void aggregate(ServerInterface &srvInterface, BlockReader &arg_reader, IntermediateAggs &aggs) along with initAggregate, combine, and terminate. For references on what a fully implemented Aggregate Function looks like, check the examples in the example folder.

which the inlined aggregateArrs implemention will invoke

virtual void Vertica::AggregateFunction::combine (ServerInterface & *srvInterface,* IntermediateAggs & *aggs_output,* MultipleIntermediateAggs & *aggs_other*) `[pure virtual]`

Called when intermediate aggregates need to be combined with each other

virtual void Vertica::UDXObject::destroy (ServerInterface & *srvInterface,* const SizedColumnTypes & *argTypes*) `[inline], [virtual], [inherited]`

Perform per instance destruction. This function may throw errors

virtual void Vertica::AggregateFunction::initAggregate (ServerInterface & *srvInterface,* IntermediateAggs & *aggs*) `[pure virtual]`

Called by the server to set the starting values of the Intermediate aggregates.

Note

This can be called multiple times on multiple machines, so starting values should be idempotent.

virtual void Vertica::UDXObject::setup (ServerInterface & *srvInterface,* const SizedColumnTypes & *argTypes*) `[inline], [virtual], [inherited]`

Perform per instance initialization. This function may throw errors.

virtual void Vertica::AggregateFunction::terminate (ServerInterface & *srvInterface,* BlockWriter & *res_writer,* IntermediateAggs & *aggs*) `[pure virtual]`

Called by the server to get the output to the aggregate function

void Vertica::AggregateFunction::updateCols (BlockReader & *arg_reader,* char * *arg,* int *count,* IntermediateAggs & *intAggs,* char * *aggPtr,* std::vector< int > & *intOffsets*) [inline], [static]

Helper function for aggregateArrs.

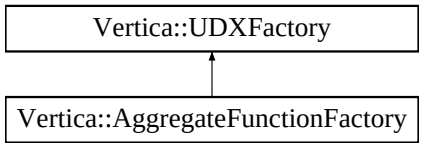
Note

User should not call this function.

Helper function for inlining aggregates by swapping out the pointers for the next block of input args and that block's corresponding intermediateAggs

Vertica::AggregateFunctionFactory Class Reference

Inheritance diagram for Vertica::AggregateFunctionFactory:



Public Types

- enum UDXType { **FUNCTION**, **TRANSFORM**, **ANALYTIC**, **MULTI_TRANSFORM**, **AGGREGATE**, **LOAD_SOURCE**, **LOAD_FILTER**, **LOAD_PARSER**, **TYPE** }

Public Member Functions

- virtual AggregateFunction * createAggregateFunction (ServerInterface &srvInterface)=0
- virtual void getIntermediateTypes (ServerInterface &srvInterface, const SizedColumnTypes &inputTypes, SizedColumnTypes &intermediateTypeMetaData)=0
- virtual void getParameterType (ServerInterface &srvInterface, SizedColumnTypes &parameterTypes)
- virtual void getPerInstanceResources (ServerInterface &srvInterface, VResources &res)
- virtual void getPrototype (ServerInterface &srvInterface, ColumnTypes &argTypes, ColumnTypes &returnType)=0
- virtual void getReturnType (ServerInterface &srvInterface, const SizedColumnTypes &argTypes, SizedColumnTypes &returnType)=0

Protected Member Functions

- virtual UDXType getUDXFactoryType ()

Protected Attributes

- Oid **libOid**
- std::string **sqlName**

Detailed Description

Interface to provide User Defined Aggregate compile time information

Member Enumeration Documentation

enum Vertica::UDXFactory::UDXType `[inherited]`

The type of UDX instance this factory produces

Member Function Documentation

virtual AggregateFunction* Vertica::AggregateFunctionFactory::createAggregateFunction (ServerInterface & *srvInterface*) `[pure virtual]`

Called when Vertica needs a new AggregateFunction object to process a function call.

Returns

an AggregateFunction object which implements the UDx API described by this metadata.

Parameters

srvInterface	a ServerInterface object used to communicate with Vertica

Note

More than one object may be instantiated per query.

virtual void Vertica::AggregateFunctionFactory::getIntermediateTypes (ServerInterface & *srvInterface*, const SizedColumnTypes & *inputTypes*, SizedColumnTypes & *intermediateTypeMetaData*) `[pure virtual]`

Returns the intermediate types used for this aggregate. Called by the server to set the types of the Intermediate aggregates.

virtual void Vertica::UDXFactory::getParameterType (ServerInterface & *srvInterface*, SizedColumnTypes & *parameterTypes*) `[inline], [virtual], [inherited]`

Function to tell Vertica the name and types of parameters that this function uses. Vertica will use this to warn function callers that certain parameters they provide are not affecting anything, or that certain parameters that are not being set are reverting to default values.

Reimplemented in Vertica::ParserFactory.

virtual void Vertica::UDXFactory::getPerInstanceResources (ServerInterface & *srvInterface*, VResources & *res*) `[inline], [virtual], [inherited]`

Set the resource required for each instance of the UDX Object subclass

Parameters

srvInterface	a ServerInterface object used to communicate with Vertica
res	a VResources object used to tell Vertica what resources are needed by the UDX

virtual void Vertica::UDXFactory::getPrototype (ServerInterface & *srvInterface*, ColumnTypes & *argTypes*, ColumnTypes & *returnType*) `[pure virtual], [inherited]`

Provides the argument and return types of the UDX

Implemented in Vertica::UDLFactory, and Vertica::MultiPhaseTransformFunctionFactory.

virtual void Vertica::UDXFactory::getReturnType (ServerInterface & *srvInterface*, const SizedColumnTypes & *argTypes*, SizedColumnTypes & *returnType*) `[pure virtual], [inherited]`

Function to tell Vertica what the return types (and length/precision if necessary) of this UDX are.

For CHAR/VARCHAR types, specify the max length,

For NUMERIC types, specify the precision and scale.

For Time/Timestamp types (with or without time zone), specify the precision, -1 means unspecified/don't care

For IntervalYM/IntervalDS types, specify the precision and range

For all other types, no length/precision specification needed

Parameters

argTypes	Provides the data types of arguments that this UDT was called with. This may be used to modify the return types accordingly.
returnType	User code must fill in the names and data types returned by the UDT.

Implemented in Vertica::UDLFactory, Vertica::MultiPhaseTransformFunctionFactory, and Vertica::ScalarFunctionFactory.

virtual UDXType Vertica::AggregateFunctionFactory::getUDXFactoryType ( )

`[inline], [protected], [virtual]`

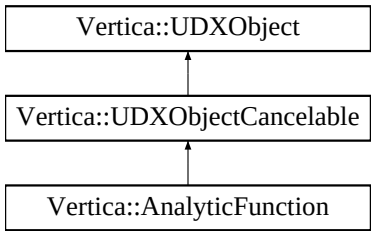
Returns

the object type internally used by Vertica

Implements Vertica::UDXFactory.

Vertica::AnalyticFunction Class Reference

Inheritance diagram for Vertica::AnalyticFunction:



Public Member Functions

- virtual void cancel (ServerInterface &srvInterface)
- virtual void destroy (ServerInterface &srvInterface, const SizedColumnTypes &argTypes)
- bool isCanceled ()
- virtual void processPartition (ServerInterface &srvInterface, AnalyticPartitionReader &input_reader, AnalyticPartitionWriter &output_writer)=0
- virtual void setup (ServerInterface &srvInterface, const SizedColumnTypes &argTypes)

Protected Attributes

- volatile bool **canceled**

Detailed Description

Interface for User Defined Analytic, the actual code to process a partition of data coming in as a stream

Member Function Documentation

virtual void Vertica::UDXObjectCancelable::cancel (ServerInterface & *srvInterface*) `[inline], [virtual], [inherited]`

This function is invoked from a different thread when the execution is canceled This baseclass cancel should be called in any override.

virtual void Vertica::UDXObject::destroy (ServerInterface & *srvInterface*, const SizedColumnTypes & *argTypes*) `[inline], [virtual], [inherited]`

Perform per instance destruction. This function may throw errors

bool Vertica::UDXObjectCancelable::isCanceled ( ) `[inline], [inherited]`

Returns true if execution was canceled.

virtual void Vertica::AnalyticFunction::processPartition (ServerInterface & *srvInterface*, AnalyticPartitionReader & *input_reader*, AnalyticPartitionWriter & *output_writer*) `[pure virtual]`

Invoke a user defined analytic on a set of rows. As the name suggests, a batch of rows are passed in for every invocation to amortize performance.

Parameters

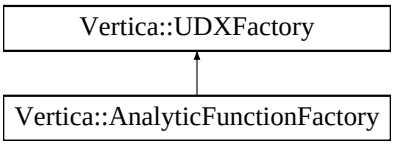
srvInterface	a ServerInterface object used to communicate with Vertica
input_reader	input rows
output_writer	output location

virtual void Vertica::UDXObject::setup (ServerInterface & *srvInterface,* const SizedColumnTypes & *argTypes*) `[inline],[virtual],[inherited]`

Perform per instance initialization. This function may throw errors.

Vertica::AnalyticFunctionFactory Class Reference

Inheritance diagram for Vertica::AnalyticFunctionFactory:



Public Types

- enum UDXType {
  FUNCTION, **TRANSFORM**, **ANALYTIC**, **MULTI_TRANSFORM**, **AGGREGATE**, **LOAD_SOURCE**, **LOAD_FILTER**, **LOAD_PARSER**, **TYPE** }

Public Member Functions

- virtual AnalyticFunction ∗ createAnalyticFunction (ServerInterface &srvInterface)=0
- virtual void getParameterType (ServerInterface &srvInterface, SizedColumnTypes &parameterTypes)
- virtual void getPerInstanceResources (ServerInterface &srvInterface, VResources &res)
- virtual void getPrototype (ServerInterface &srvInterface, ColumnTypes &argTypes, ColumnTypes &returnType)=0
- virtual void getReturnType (ServerInterface &srvInterface, const SizedColumnTypes &argTypes, SizedColumnTypes &returnType)=0

Protected Member Functions

- virtual UDXType getUDXFactoryType ()

Protected Attributes

- Oid **libOid**
- std::string **sqlName**

Detailed Description

Interface to provide User Defined Analytic compile time information

Member Enumeration Documentation

enum Vertica::UDXFactory::UDXType `[inherited]`

The type of UDX instance this factory produces

Member Function Documentation

virtual AnalyticFunction* Vertica::AnalyticFunctionFactory::createAnalyticFunction (ServerInterface & *srvInterface*) `[pure virtual]`

Called when Vertica needs a new AnalyticFunction object to process a function call.

Returns

an AnalyticFunction object which implements the UDx API described by this metadata.

Parameters

srvInterface	a ServerInterface object used to communicate with Vertica

Note

More than one object may be instantiated per query.

virtual void Vertica::UDXFactory::getParameterType (ServerInterface & *srvInterface,* SizedColumnTypes & *parameterTypes*) `[inline], [virtual], [inherited]`

Function to tell Vertica the name and types of parameters that this function uses. Vertica will use this to warn function callers that certain parameters they provide are not affecting anything, or that certain parameters that are not being set are reverting to default values.

Reimplemented in Vertica::ParserFactory.

virtual void Vertica::UDXFactory::getPerInstanceResources (ServerInterface & *srvInterface,* VResources & *res*) `[inline], [virtual], [inherited]`

Set the resource required for each instance of the UDX Object subclass

Parameters

srvInterface	a ServerInterface object used to communicate with Vertica
res	a VResources object used to tell Vertica what resources are needed by the UDX

virtual void Vertica::UDXFactory::getPrototype (ServerInterface & *srvInterface*, ColumnTypes & *argTypes*, ColumnTypes & *returnType*) `[pure virtual], [inherited]`

Provides the argument and return types of the UDX

Implemented in Vertica::UDLFactory, and Vertica::MultiPhaseTransformFunctionFactory.

virtual void Vertica::UDXFactory::getReturnType (ServerInterface & *srvInterface*, const SizedColumnTypes & *argTypes*, SizedColumnTypes & *returnType*) `[pure virtual], [inherited]`

Function to tell Vertica what the return types (and length/precision if necessary) of this UDX are.

For CHAR/VARCHAR types, specify the max length,

For NUMERIC types, specify the precision and scale.

For Time/Timestamp types (with or without time zone), specify the precision, -1 means unspecified/don't care

For IntervalYM/IntervalDS types, specify the precision and range

For all other types, no length/precision specification needed

Parameters

argTypes	Provides the data types of arguments that this UDT was called with. This may be used to modify the return types accordingly.
returnType	User code must fill in the names and data types returned by the UDT.

Implemented in Vertica::UDLFactory, Vertica::MultiPhaseTransformFunctionFactory, and Vertica::ScalarFunctionFactory.

virtual UDXType Vertica::AnalyticFunctionFactory::getUDXFactoryType ( ) `[inline], [protected], [virtual]`

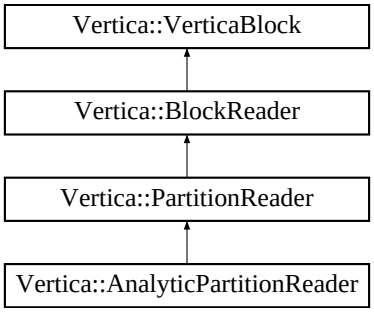
Returns

the object type internally used by Vertica

Implements Vertica::UDXFactory.

Vertica::AnalyticPartitionReader Class Reference

Inheritance diagram for Vertica::AnalyticPartitionReader:



Public Member Functions

- **AnalyticPartitionReader** (size_t narg, EE::UserDefinedProcess *udx_object)
- const vbool * getBoolPtr (size_t idx)

  Get a pointer to a BOOLEAN value from the input row.
- const vbool & getBoolRef (size_t idx)

  Get a reference to a BOOLEAN value from the input row.
- template<class T >
  const T * getColPtr (size_t idx)
- template<class T >
  T * **getColPtrForWrite** (size_t idx)
- template<class T >
  const T & getColRef (size_t idx)
- template<class T >
  T & **getColRefForWrite** (size_t idx)
- const DateADT * getDatePtr (size_t idx)

  Get a pointer to a DATE value from the input row.
- const DateADT & getDateRef (size_t idx)

  Get a reference to a DATE value from the input row.
- const vfloat * getFloatPtr (size_t idx)

  Get a pointer to a FLOAT value from the input row.
- const vfloat & getFloatRef (size_t idx)

  Get a reference to a FLOAT value from the input row.
- const Interval * getIntervalPtr (size_t idx)

  Get a pointer to an INTERVAL value from the input row.
- const Interval & getIntervalRef (size_t idx)

  Get a reference to an INTERVAL value from the input row.
- const IntervalYM * getIntervalYMPtr (size_t idx)

Get a pointer to a INTERVAL YEAR TO MONTH value from the input row.

- const IntervalYM & getIntervalYMRef (size_t idx)

  Get a reference to an INTERVAL YEAR TO MONTH value from the input row.
- const vint * getIntPtr (size_t idx)

  Get a pointer to an INTEGER value from the input row.
- const vint & getIntRef (size_t idx)

  Get a reference to an INTEGER value from the input row.
- size_t getNumCols () const
- const VNumeric * getNumericPtr (size_t idx)

  Get a pointer to a VNumeric value from the input row.
- const VNumeric & getNumericRef (size_t idx)

  Get a reference to a VNumeric value from the input row.
- int getNumRows () const
- const VString * getStringPtr (size_t idx)

  Get a pointer to a VString value from the input row.
- const VString & getStringRef (size_t idx)

  Get a reference to an VString value from the input row.
- const TimeADT * getTimePtr (size_t idx)

  Get a pointer to a TIME value from the input row.
- const TimeADT & getTimeRef (size_t idx)

  Get a reference to a TIME value from the input row.
- const Timestamp * getTimestampPtr (size_t idx)

  Get a pointer to a TIMESTAMP value from the input row.
- const Timestamp & getTimestampRef (size_t idx)

  Get a reference to a TIMESTAMP value from the input row.
- const TimestampTz * getTimestampTzPtr (size_t idx)

  Get a pointer to a TIMESTAMP WITH TIMEZONE value from the input row.
- const TimestampTz & getTimestampTzRef (size_t idx)

  Get a reference to a TIMESTAMP WITH TIMEZONE value from the input row.
- const TimeTzADT * getTimeTzPtr (size_t idx)

  Get a pointer to a TIME WITH TIMEZONE value from the input row.
- const TimeTzADT & getTimeTzRef (size_t idx)

  Get a reference to a TIME WITH TIMEZONE value from the input row.
- const SizedColumnTypes & getTypeMetaData () const
- SizedColumnTypes & getTypeMetaData ()
- bool **isNewOrderByKey** ()
- bool isNull (int col)

  Check if the idx'th argument is null.
- bool **next** ()
- virtual bool readNextBlock ()

Protected Member Functions

- void addCol (char *arg, int colstride, const VerticaType &dt, const std::string fieldName="")
- void **addCol** (const char *arg, int colstride, const VerticaType &dt, const std::string fieldName="")
- virtual void **setupColsAndStrides** ()
- void **validateStringColumn** (size_t idx, const VString &s, const VerticaType &t)

Protected Attributes

- union {
  EE::UserDefinedAnalytic * **udan**
  EE::UserDefinedTransform * **udt**
  EE::UserDefinedProcess * **udx_object**
  };

- std::vector< char * > **cols**
- std::vector< int > **colstrides**
- int **count**
- EE::DataHolder * **dh**
- int **index**
- size_t **ncols**
- std::vector< bool > **newOrderMarkers**
- vpos **pstart**
- std::vector< VString > **svWrappers**
- SizedColumnTypes **typeMetaData**
- std::vector< VNumeric > **vnWrappers**

Friends

- class **AnalyticPartitionReaderHelper**
- class **EE::UserDefinedAnalytic**
- class **EE::UserDefinedProcess**

Detailed Description

AnalyticPartitionReader provides an iterator-based read interface over all the partition_by keys, order_by keys, and function arguments in a partition.

Member Function Documentation

void Vertica::VerticaBlock::addCol (char * *arg,* int *colstride,* const VerticaType & *dt,* const std::string *fieldName* = "") `[inline],[protected],[inherited]`

Add the location for reading a particular argument.

Parameters

arg	The base location to find data.
colstride	The stride between data instances.
dt	The type of input.
fieldname	the name of the field

const vbool* Vertica::BlockReader::getBoolPtr (size_t *idx*) `[inline], [inherited]`

Get a pointer to a BOOLEAN value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as a BOOLEAN.

const vbool& Vertica::BlockReader::getBoolRef (size_t *idx*) `[inline], [inherited]`

Get a reference to a BOOLEAN value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as an BOOLEAN.

template<class T > const T* Vertica::VerticaBlock::getColPtr (size_t *idx*) `[inline],[inherited]`

Returns

a pointer to the idx'th argument, cast appropriately.

Example:

```
const vint *a = arg_reader->getColPtr<vint>(0);
```

template<class T > const T& Vertica::VerticaBlock::getColRef (size_t *idx*) `[inline], [inherited]`

Returns

a pointer to the idx'th argument, cast appropriately.

Example: const vint a = arg_reader->getColRef<vint>(0);

const DateADT* Vertica::BlockReader::getDatePtr (size_t *idx*) `[inline], [inherited]`

Get a pointer to a DATE value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as a DATE.

const DateADT& Vertica::BlockReader::getDateRef (size_t *idx*) `[inline], [inherited]`

Get a reference to a DATE value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as an DATE.

const vfloat* Vertica::BlockReader::getFloatPtr (size_t *idx*) `[inline], [inherited]`

Get a pointer to a FLOAT value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as a FLOAT.

const vfloat& Vertica::BlockReader::getFloatRef (size_t *idx*) `[inline], [inherited]`

Get a reference to a FLOAT value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

A reference to the idx'th argument, cast as an FLOAT.

const Interval* Vertica::BlockReader::getIntervalPtr (size_t *idx*) `[inline], [inherited]`

Get a pointer to an INTERVAL value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as an INTERVAL.

const Interval& Vertica::BlockReader::getIntervalRef (size_t *idx*) `[inline], [inherited]`

Get a reference to an INTERVAL value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as an INTERVAL.

const IntervalYM* Vertica::BlockReader::getIntervalYMPtr (size_t *idx*) `[inline], [inherited]`

Get a pointer to a INTERVAL YEAR TO MONTH value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A point to the retrieved value cast as a INTERVAL YEAR TO MONTH.

const IntervalYM& Vertica::BlockReader::getIntervalYMRef (size_t *idx*) `[inline], [inherited]`

Get a reference to an INTERVAL YEAR TO MONTH value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as an INTERVAL YEAR TO MONTH.

const vint* Vertica::BlockReader::getIntPtr (size_t *idx*) `[inline], [inherited]`

Get a pointer to an INTEGER value from the input row.

Returns

a pointer to the idx'th argument, cast appropriately.

Parameters

idx	The column number to retrieve from the input row.

Example:

```
const vint *a = arg_reader->getIntPtr(0);
```

const vint& Vertica::BlockReader::getIntRef (size_t *idx*) [inline], [inherited]

Get a reference to an INTEGER value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as an INTEGER.

Example:

```
const vint a = arg_reader->getIntRef(0);
```

size_t Vertica::VerticaBlock::getNumCols ( ) const [inline], [inherited]

Returns

the number of columns held by this block.

const VNumeric* Vertica::BlockReader::getNumericPtr (size_t *idx*) [inline], [inherited]

Get a pointer to a VNumeric value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

A pointer to the retrieved value cast as a Numeric.

const VNumeric& Vertica::BlockReader::getNumericRef (size_t *idx*) `[inline], [inherited]`

Get a reference to a VNumeric value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as an VNumeric.

int Vertica::VerticaBlock::getNumRows ( ) const `[inline], [inherited]`

Returns

the number of rows held by this block.

const VString* Vertica::BlockReader::getStringPtr (size_t *idx*) `[inline], [inherited]`

Get a pointer to a VString value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

A pointer to the retrieved value cast as a VString.

const VString& Vertica::BlockReader::getStringRef (size_t *idx*) `[inline], [inherited]`

Get a reference to an VString value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as an VString.

const TimeADT* Vertica::BlockReader::getTimePtr (size_t *idx*) `[inline], [inherited]`

Get a pointer to a TIME value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as a TIME.

const TimeADT& Vertica::BlockReader::getTimeRef (size_t *idx*) `[inline], [inherited]`

Get a reference to a TIME value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as a TIME.

const Timestamp* Vertica::BlockReader::getTimestampPtr (size_t *idx*) `[inline], [inherited]`

Get a pointer to a TIMESTAMP value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as a TIMESTAMP.

const Timestamp& Vertica::BlockReader::getTimestampRef (size_t *idx*) `[inline], [inherited]`

Get a reference to a TIMESTAMP value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as a TIMESTAMP.

const TimestampTz* Vertica::BlockReader::getTimestampTzPtr (size_t *idx*) `[inline], [inherited]`

Get a pointer to a TIMESTAMP WITH TIMEZONE value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as a TIMESTAMP WITH TIMEZONE .

const TimestampTz& Vertica::BlockReader::getTimestampTzRef (size_t *idx*) `[inline], [inherited]`

Get a reference to a TIMESTAMP WITH TIMEZONE value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as a TIMESTAMP WITH TIMEZONE.

const TimeTzADT* Vertica::BlockReader::getTimeTzPtr (size_t *idx*) `[inline], [inherited]`

Get a pointer to a TIME WITH TIMEZONE value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as a TIME WITH TIMEZONE.

const TimeTzADT& Vertica::BlockReader::getTimeTzRef (size_t *idx*) `[inline], [inherited]`

Get a reference to a TIME WITH TIMEZONE value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as a TIME WITH TIMEZONE.

const SizedColumnTypes& Vertica::VerticaBlock::getTypeMetaData ( ) const `[inline], [inherited]`

Returns

information about the types and numbers of arguments

SizedColumnTypes& Vertica::VerticaBlock::getTypeMetaData ( ) `[inline], [inherited]`

Returns

information about the types and numbers of arguments

bool Vertica::BlockReader::isNull (int *col*) `[inline], [inherited]`

Check if the idx'th argument is null.

Parameters

col	The column number in the row to check for null

Returns

true is the col value is null false otherwise

virtual bool Vertica::AnalyticPartitionReader::readNextBlock ( ) [virtual]

Reads in the next block of data and positions cursor at the beginning.

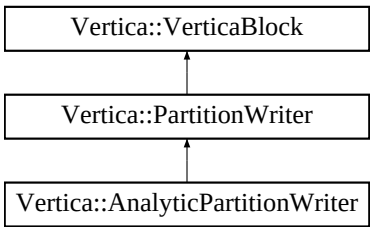
Returns

false if there's no more input data

Reimplemented from Vertica::PartitionReader.

Vertica::AnalyticPartitionWriter Class Reference

Inheritance diagram for Vertica::AnalyticPartitionWriter:



Public Member Functions

- **AnalyticPartitionWriter** (size_t narg, EE::UserDefinedProcess *udx_object)
- void copyFromInput (size_t dstIdx, PartitionReader &input_reader, size_t srcIdx)
- template<class T >
  const T * getColPtr (size_t idx)
- template<class T >
  T * **getColPtrForWrite** (size_t idx)
- template<class T >
  const T & getColRef (size_t idx)
- template<class T >
  T & **getColRefForWrite** (size_t idx)

- size_t getNumCols () const
- VNumeric & **getNumericRef** (size_t idx)
- int getNumRows () const
- VString & **getStringRef** (size_t idx)
- VString & **getStringRefNoClear** (size_t idx)
- const SizedColumnTypes & getTypeMetaData () const
- SizedColumnTypes & getTypeMetaData ()
- void * **getVoidPtr** (size_t idx)
- virtual bool getWriteableBlock ()
- bool **next** ()
- void **setBool** (size_t idx, vbool r)
- void **setDate** (size_t idx, DateADT r)
- void **setFloat** (size_t idx, vfloat r)
- void setInt (size_t idx, vint r)
- void **setInterval** (size_t idx, Interval r)
- void setNull (size_t idx)

  Set the idx'th argument to null.
- void **setTime** (size_t idx, TimeADT r)
- void **setTimestamp** (size_t idx, Timestamp r)
- void **setTimestampTz** (size_t idx, TimestampTz r)
- void **setTimeTz** (size_t idx, TimeTzADT r)
- void **validateColumn** (size_t idx)

Protected Member Functions

- void addCol (char *arg, int colstride, const VerticaType &dt, const std::string fieldName="")
- void **addCol** (const char *arg, int colstride, const VerticaType &dt, const std::string fieldName="")
- void **validateStringColumn** (size_t idx, const VString &s, const VerticaType &t)

Protected Attributes

- union {
  EE::UserDefinedAnalytic * **udan**
  EE::UserDefinedTransform * **udt**
  EE::UserDefinedProcess * **udx_object**
  };

- std::vector< char * > **cols**
- std::vector< int > **colstrides**

- int **count**
- int **index**
- size_t **ncols**
- int **pstart**
- std::vector< VString > **svWrappers**
- SizedColumnTypes **typeMetaData**
- std::vector< VNumeric > **vnWrappers**

Detailed Description

AnalyticPartitionWriter provides an iterator-based write interface over all input data in a single partition. It automatically makes space as needed.

Member Function Documentation

void Vertica::VerticaBlock::addCol (char * *arg*, int *colstride*, const VerticaType & *dt*, const std::string *fieldName* = "") `[inline], [protected], [inherited]`

Add the location for reading a particular argument.

Parameters

arg	The base location to find data.
colstride	The stride between data instances.
dt	The type of input.
fieldname	the name of the field

void Vertica::PartitionWriter::copyFromInput (size_t *dstIdx*, PartitionReader & *input_reader*, size_t *srcIdx*) `[inline], [inherited]`

Copies a column from the input reader to the output writer. The data types and sizes of the source and destination columns must match exactly.

Parameters

dstIdx	The destination column index (in the output writer)
input_reader	The input reader from which to copy a column
srcIdx	The source column index (in the input reader)

template<class T > const T* Vertica::VerticaBlock::getColPtr (size_t *idx*) `[inline], [inherited]`

Returns

a pointer to the idx'th argument, cast appropriately.

Example:

```
const vint *a = arg_reader->getColPtr<vint>(0);
```

template<class T > const T& Vertica::VerticaBlock::getColRef (size_t *idx*) `[inline], [inherited]`

Returns

a pointer to the idx'th argument, cast appropriately.

Example: const vint a = arg_reader->getColRef<vint>(0);

size_t Vertica::VerticaBlock::getNumCols ( ) const `[inline], [inherited]`

Returns

the number of columns held by this block.

int Vertica::VerticaBlock::getNumRows ( ) const `[inline], [inherited]`

Returns

the number of rows held by this block.

const SizedColumnTypes& Vertica::VerticaBlock::getTypeMetaData ( ) const `[inline], [inherited]`

Returns

information about the types and numbers of arguments

SizedColumnTypes& Vertica::VerticaBlock::getTypeMetaData ( ) `[inline], [inherited]`

Returns

information about the types and numbers of arguments

virtual bool Vertica::AnalyticPartitionWriter::getWriteableBlock ( ) [virtual]

Gets a writeable block of data and positions cursor at the beginning.

Reimplemented from Vertica::PartitionWriter.

void Vertica::PartitionWriter::setInt (size_t *idx*, vint *r*) [inline], [inherited]

Setter methods

void Vertica::PartitionWriter::setNull (size_t *idx*) [inline], [inherited]

Set the idx'th argument to null.

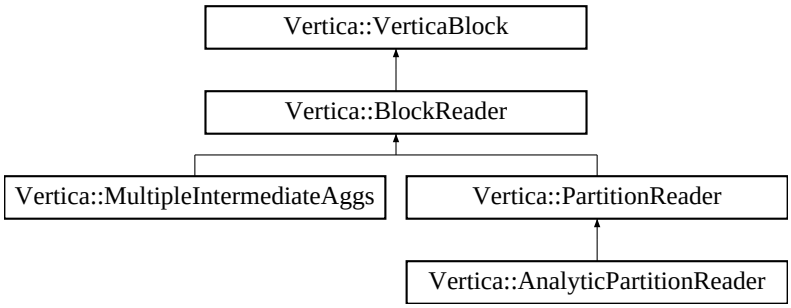
Parameters

idx	The column number in the row to set to null

Vertica::BlockReader Class Reference

Iterator interface for reading rows in a Vertica block.

Inheritance diagram for Vertica::BlockReader:



Public Member Functions

- **BlockReader** (size_t narg, int rowcount)
- const vbool * getBoolPtr (size_t idx)

  Get a pointer to a BOOLEAN value from the input row.
- const vbool & getBoolRef (size_t idx)

  Get a reference to a BOOLEAN value from the input row.
- template<class T >
  const T * getColPtr (size_t idx)

- template<class T >
  T * **getColPtrForWrite** (size_t idx)
- template<class T >
  const T & getColRef (size_t idx)
- template<class T >
  T & **getColRefForWrite** (size_t idx)
- const DateADT * getDatePtr (size_t idx)

  Get a pointer to a DATE value from the input row.
- const DateADT & getDateRef (size_t idx)

  Get a reference to a DATE value from the input row.
- const vfloat * getFloatPtr (size_t idx)

  Get a pointer to a FLOAT value from the input row.
- const vfloat & getFloatRef (size_t idx)

  Get a reference to a FLOAT value from the input row.
- const Interval * getIntervalPtr (size_t idx)

  Get a pointer to an INTERVAL value from the input row.
- const Interval & getIntervalRef (size_t idx)

  Get a reference to an INTERVAL value from the input row.
- const IntervalYM * getIntervalYMPtr (size_t idx)

  Get a pointer to a INTERVAL YEAR TO MONTH value from the input row.
- const IntervalYM & getIntervalYMRef (size_t idx)

  Get a reference to an INTERVAL YEAR TO MONTH value from the input row.
- const vint * getIntPtr (size_t idx)

  Get a pointer to an INTEGER value from the input row.
- const vint & getIntRef (size_t idx)

  Get a reference to an INTEGER value from the input row.
- size_t getNumCols () const
- const VNumeric * getNumericPtr (size_t idx)

  Get a pointer to a VNumeric value from the input row.
- const VNumeric & getNumericRef (size_t idx)

  Get a reference to a VNumeric value from the input row.
- int getNumRows () const
- const VString * getStringPtr (size_t idx)

  Get a pointer to a VString value from the input row.
- const VString & getStringRef (size_t idx)

  Get a reference to an VString value from the input row.
- const TimeADT * getTimePtr (size_t idx)

  Get a pointer to a TIME value from the input row.
- const TimeADT & getTimeRef (size_t idx)

  Get a reference to a TIME value from the input row.

- const Timestamp * getTimestampPtr (size_t idx)

  Get a pointer to a TIMESTAMP value from the input row.
- const Timestamp & getTimestampRef (size_t idx)

  Get a reference to a TIMESTAMP value from the input row.
- const TimestampTz * getTimestampTzPtr (size_t idx)

  Get a pointer to a TIMESTAMP WITH TIMEZONE value from the input row.
- const TimestampTz & getTimestampTzRef (size_t idx)

  Get a reference to a TIMESTAMP WITH TIMEZONE value from the input row.
- const TimeTzADT * getTimeTzPtr (size_t idx)

  Get a pointer to a TIME WITH TIMEZONE value from the input row.
- const TimeTzADT & getTimeTzRef (size_t idx)

  Get a reference to a TIME WITH TIMEZONE value from the input row.
- const SizedColumnTypes & getTypeMetaData () const
- SizedColumnTypes & getTypeMetaData ()
- bool isNull (int col)

  Check if the idx'th argument is null.
- bool next ()

Protected Member Functions

- void addCol (char *arg, int colstride, const VerticaType &dt, const std::string fieldName="")
- void **addCol** (const char *arg, int colstride, const VerticaType &dt, const std::string fieldName="")
- void **validateStringColumn** (size_t idx, const VString &s, const VerticaType &t)

Protected Attributes

- std::vector< char * > **cols**
- std::vector< int > **colstrides**
- int **count**
- int **index**
- size_t **ncols**
- std::vector< VString > **svWrappers**
- SizedColumnTypes **typeMetaData**
- std::vector< VNumeric > **vnWrappers**

Friends

- class **EE::VEval**
- class **VerticaRInterface**

Detailed Description

Iterator interface for reading rows in a Vertica block.

This class provides the input to the ScalarFunction.processBlock() function. You extract values from the input row using data type specific functions to extract each column value. You can also determine the number of columns and their data types, if your processBlock function does not have hard-coded input expectations.

Member Function Documentation

void Vertica::VerticaBlock::addCol (char * *arg*, int *colstride*, const VerticaType & *dt*, const std::string *fieldName* = "") `[inline], [protected], [inherited]`

Add the location for reading a particular argument.

Parameters

arg	The base location to find data.
colstride	The stride between data instances.
dt	The type of input.
fieldname	the name of the field

const vbool* Vertica::BlockReader::getBoolPtr (size_t *idx*) `[inline]`

Get a pointer to a BOOLEAN value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as a BOOLEAN.

const vbool& Vertica::BlockReader::getBoolRef (size_t *idx*) `[inline]`

Get a reference to a BOOLEAN value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as an BOOLEAN.

template<class T > const T* Vertica::VerticaBlock::getColPtr (size_t *idx*) [inline],[inherited]

Returns

a pointer to the idx'th argument, cast appropriately.

Example:

```
const vint *a = arg_reader->getColPtr<vint>(0);
```

template<class T > const T& Vertica::VerticaBlock::getColRef (size_t *idx*) [inline],[inherited]

Returns

a pointer to the idx'th argument, cast appropriately.

Example: const vint a = arg_reader->getColRef<vint>(0);

const DateADT* Vertica::BlockReader::getDatePtr (size_t *idx*) [inline]

Get a pointer to a DATE value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as a DATE.

const DateADT& Vertica::BlockReader::getDateRef (size_t *idx*) [inline]

Get a reference to a DATE value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as an DATE.

const vfloat* Vertica::BlockReader::getFloatPtr (size_t *idx*) [inline]

Get a pointer to a FLOAT value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as a FLOAT.

const vfloat& Vertica::BlockReader::getFloatRef (size_t *idx*) [inline]

Get a reference to a FLOAT value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

A reference to the idx'th argument, cast as an FLOAT.

const Interval* Vertica::BlockReader::getIntervalPtr (size_t *idx*) [inline]

Get a pointer to an INTERVAL value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as an INTERVAL.

const Interval& Vertica::BlockReader::getIntervalRef (size_t *idx*) `[inline]`

Get a reference to an INTERVAL value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as an INTERVAL.

const IntervalYM* Vertica::BlockReader::getIntervalYMPtr (size_t *idx*) `[inline]`

Get a pointer to a INTERVAL YEAR TO MONTH value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A point to the retrieved value cast as a INTERVAL YEAR TO MONTH.

const IntervalYM& Vertica::BlockReader::getIntervalYMRef (size_t *idx*) `[inline]`

Get a reference to an INTERVAL YEAR TO MONTH value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as an INTERVAL YEAR TO MONTH.

const vint* Vertica::BlockReader::getIntPtr (size_t *idx*) `[inline]`

Get a pointer to an INTEGER value from the input row.

Returns

a pointer to the idx'th argument, cast appropriately.

Parameters

idx	The column number to retrieve from the input row.

Example:

```
const vint *a = arg_reader->getIntPtr(0);
```

const vint& Vertica::BlockReader::getIntRef (size_t *idx*) [inline]

Get a reference to an INTEGER value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as an INTEGER.

Example:

```
const vint a = arg_reader->getIntRef(0);
```

size_t Vertica::VerticaBlock::getNumCols ( ) const [inline], [inherited]

Returns

the number of columns held by this block.

const VNumeric* Vertica::BlockReader::getNumericPtr (size_t *idx*) [inline]

Get a pointer to a VNumeric value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

A pointer to the retrieved value cast as a Numeric.

const VNumeric& Vertica::BlockReader::getNumericRef (size_t *idx*) [inline]

Get a reference to a VNumeric value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as an VNumeric.

int Vertica::VerticaBlock::getNumRows ( ) const [inline], [inherited]

Returns

the number of rows held by this block.

const VString* Vertica::BlockReader::getStringPtr (size_t *idx*) [inline]

Get a pointer to a VString value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

A pointer to the retrieved value cast as a VString.

const VString& Vertica::BlockReader::getStringRef (size_t *idx*) [inline]

Get a reference to an VString value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as an VString.

const TimeADT* Vertica::BlockReader::getTimePtr (size_t *idx*) [inline]

Get a pointer to a TIME value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as a TIME.

const TimeADT& Vertica::BlockReader::getTimeRef (size_t *idx*) [inline]

Get a reference to a TIME value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as a TIME.

const Timestamp* Vertica::BlockReader::getTimestampPtr (size_t *idx*) [inline]

Get a pointer to a TIMESTAMP value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as a TIMESTAMP.

const Timestamp& Vertica::BlockReader::getTimestampRef (size_t *idx*) `[inline]`

Get a reference to a TIMESTAMP value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as a TIMESTAMP.

const TimestampTz* Vertica::BlockReader::getTimestampTzPtr (size_t *idx*) `[inline]`

Get a pointer to a TIMESTAMP WITH TIMEZONE value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as a TIMESTAMP WITH TIMEZONE .

const TimestampTz& Vertica::BlockReader::getTimestampTzRef (size_t *idx*) `[inline]`

Get a reference to a TIMESTAMP WITH TIMEZONE value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as a TIMESTAMP WITH TIMEZONE.

const TimeTzADT* Vertica::BlockReader::getTimeTzPtr (size_t *idx*) `[inline]`

Get a pointer to a TIME WITH TIMEZONE value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as a TIME WITH TIMEZONE.

const TimeTzADT& Vertica::BlockReader::getTimeTzRef (size_t *idx*) `[inline]`

Get a reference to a TIME WITH TIMEZONE value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as a TIME WITH TIMEZONE.

const SizedColumnTypes& Vertica::VerticaBlock::getTypeMetaData ( ) const `[inline]`, `[inherited]`

Returns

information about the types and numbers of arguments

SizedColumnTypes& Vertica::VerticaBlock::getTypeMetaData ( ) `[inline]`, `[inherited]`

Returns

information about the types and numbers of arguments

bool Vertica::BlockReader::isNull (int *col*) `[inline]`

Check if the idx'th argument is null.

Parameters

col	The column number in the row to check for null

Returns

true is the col value is null false otherwise

bool Vertica::BlockReader::next ( ) [inline]

Advance to the next record.

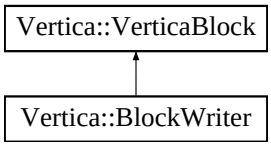
Returns

true if there are more rows to read, false otherwise.

Vertica::BlockWriter Class Reference

Iterator interface for writing rows to a Vertica block.

Inheritance diagram for Vertica::BlockWriter:



Public Member Functions

- **BlockWriter** (char *outArr, int stride, int rowcount, const VerticaType &dt)
- template<class T >
  const T * getColPtr (size_t idx)
- template<class T >
  T * **getColPtrForWrite** (size_t idx)
- template<class T >
  const T & getColRef (size_t idx)
- template<class T >
  T & **getColRefForWrite** (size_t idx)
- size_t getNumCols () const
- VNumeric & getNumericRef ()

  Allocate a new VNumeric object to use as output.
- int getNumRows () const
- VString & getStringRef ()

  Allocates a new VString object to use as output.
- const SizedColumnTypes & getTypeMetaData () const
- SizedColumnTypes & getTypeMetaData ()
- void * **getVoidPtr** ()

- void next ()

  Complete writing this row of output and move to the next row.
- void setBool (vbool r)

  Adds a BOOLEAN value to the output row.
- void setDate (DateADT r)

  Adds a BOOLEAN value to the output row.
- void setFloat (vfloat r)

  Adds a FLOAT value to the output row.
- void setInt (vint r)

  Adds an INTEGER value to the output row.
- void setInterval (Interval r)

  Adds an INTERVAL value to the output row.
- void setIntervalYM (IntervalYM r)

  Adds an INTERVAL YEAR TO MONTH value to the output row.
- void setTime (TimeADT r)

  Adds a TIMESTAMP value to the output row.
- void setTimestamp (Timestamp r)

  Adds a TIMESTAMP value to the output row.
- void setTimestampTz (TimestampTz r)

  Adds a TIMESTAMP WITH TIMEZONE value to the output row.
- void setTimeTz (TimeTzADT r)

  Adds a TIMESTAMP WITH TIMEZONE value to the output row.

Protected Member Functions

- void addCol (char *arg, int colstride, const VerticaType &dt, const std::string fieldName="")
- void **addCol** (const char *arg, int colstride, const VerticaType &dt, const std::string fieldName="")
- void **validateStringColumn** (size_t idx, const VString &s, const VerticaType &t)

Protected Attributes

- std::vector< char * > **cols**
- std::vector< int > **colstrides**
- int **count**
- int **index**
- size_t **ncols**
- std::vector< VString > **svWrappers**
- SizedColumnTypes **typeMetaData**
- std::vector< VNumeric > **vnWrappers**

Friends

- class **EE::VEval**

Detailed Description

Iterator interface for writing rows to a Vertica block.

This class provides the output rows that ScalarFunction.processBlock() writes to.

Member Function Documentation

void Vertica::VerticaBlock::addCol (char * *arg*, int *colstride*, const VerticaType & *dt*, const std::string *fieldName* = "") `[inline], [protected], [inherited]`

Add the location for reading a particular argument.

Parameters

arg	The base location to find data.
colstride	The stride between data instances.
dt	The type of input.
fieldname	the name of the field

template<class T > const T* Vertica::VerticaBlock::getColPtr (size_t *idx*) `[inline], [inherited]`

Returns

a pointer to the idx'th argument, cast appropriately.

Example:

```
const vint *a = arg_reader->getColPtr<vint>(0);
```

template<class T > const T& Vertica::VerticaBlock::getColRef (size_t *idx*) `[inline], [inherited]`

Returns

a pointer to the idx'th argument, cast appropriately.

Example: const vint a = arg_reader->getColRef<vint>(0);

size_t Vertica::VerticaBlock::getNumCols ( ) const [inline], [inherited]

Returns

the number of columns held by this block.

VNumeric& Vertica::BlockWriter::getNumericRef ( ) [inline]

Allocate a new VNumeric object to use as output.

Returns

A new VNumeric object to hold output. This object automatically added to the output row.

int Vertica::VerticaBlock::getNumRows ( ) const [inline], [inherited]

Returns

the number of rows held by this block.

VString& Vertica::BlockWriter::getStringRef ( ) [inline]

Allocates a new VString object to use as output.

Returns

A new VString object to hold output. This object automatically added to the output row.

const SizedColumnTypes& Vertica::VerticaBlock::getTypeMetaData ( ) const [inline], [inherited]

Returns

information about the types and numbers of arguments

SizedColumnTypes& Vertica::VerticaBlock::getTypeMetaData ( ) [inline], [inherited]

Returns

information about the types and numbers of arguments

void Vertica::BlockWriter::next ( ) [inline]

Complete writing this row of output and move to the next row.

void Vertica::BlockWriter::setBool (vbool *r*) [inline]

Adds a BOOLEAN value to the output row.

Parameters

r	The BOOLEAN value to insert into the output row.

void Vertica::BlockWriter::setDate (DateADT *r*) [inline]

Adds a BOOLEAN value to the output row.

Parameters

r	The BOOLEAN value to insert into the output row.

void Vertica::BlockWriter::setFloat (vfloat *r*) [inline]

Adds a FLOAT value to the output row.

Parameters

r	The FLOAT value to insert into the output row.

void Vertica::BlockWriter::setInt (vint *r*) [inline]

Adds an INTEGER value to the output row.

Setter methods

Parameters

r	The INTEGER value to insert into the output row.

void Vertica::BlockWriter::setInterval (Interval *r*) [inline]

Adds an INTERVAL value to the output row.

Parameters

r	The INTERVAL value to insert into the output row.

void Vertica::BlockWriter::setIntervalYM (IntervalYM *r*) [inline]

Adds an INTERVAL YEAR TO MONTH value to the output row.

Parameters

r	The INTERVAL YEAR TO MONTH value to insert into the output row.

void Vertica::BlockWriter::setTime (TimeADT *r*) [inline]

Adds a TIMESTAMP value to the output row.

Parameters

r	The TIMESTAMP value to insert into the output row.

void Vertica::BlockWriter::setTimestamp (Timestamp *r*) [inline]

Adds a TIMESTAMP value to the output row.

Parameters

r	The TIMESTAMP value to insert into the output row.

void Vertica::BlockWriter::setTimestampTz (TimestampTz *r*) [inline]

Adds a TIMESTAMP WITH TIMEZONE value to the output row.

Parameters

r	The TIMESTAMP WITH TIMEZONE value to insert into the output row.

void Vertica::BlockWriter::setTimeTz (TimeTzADT *r*) [inline]

Adds a TIMESTAMP WITH TIMEZONE value to the output row.

Parameters

r	The TIMESTAMP WITH TIMEZONE value to insert into the output row.

Vertica::ColumnTypes Class Reference

Represents (unsized) types of the columns used as input/output of a User Defined Function/Transform Function.

Public Member Functions

- void addAny ()

  Indicates that function can take any number and type of arguments.
- void addBinary ()

  Adds a column of type BINARY.
- void addBool ()

  Adds a column of type BOOLEAN.
- void addChar ()

  Adds a column of type CHAR.
- void addDate ()

  Adds a column of type DATE.
- void addFloat ()

  Adds a column of type FLOAT.
- void addInt ()

  Adds a column of type INTEGER.
- void addInterval ()

  Adds a column of type INTERVAL/INTERVAL DAY TO SECOND.
- void addIntervalYM ()

  Adds a column of type INTERVAL YEAR TO MONTH.
- void addLongVarbinary ()

  Adds a column of type VARBINARY.
- void addLongVarchar ()

  Adds a column of type VARBINARY.
- void addNumeric ()

  Adds a column of type NUMERIC.

- void addTime ()

  Adds a column of type TIME.
- void addTimestamp ()

  Adds a column of type TIMESTAMP.
- void addTimestampTz ()

  Adds a column of type TIMESTAMP WITH TIMEZONE.
- void addTimeTz ()

  Adds a column of type TIME WITH TIMEZONE.
- void **addUserDefinedType** (const char *typeName)
- void addVarbinary ()

  Adds a column of type VARBINARY.
- void addVarchar ()

  Adds a column of type VARCHAR.

Detailed Description

Represents (unsized) types of the columns used as input/output of a User Defined Function/Transform Function.

This class is used only for generating the function or transform function prototype, where the sizes and/or precisions of the data types are not known.

Vertica::DataBuffer Struct Reference

Public Attributes

- char * buf

  Pointer to the start of the buffer.
- size_t offset

  Number of bytes that have been processed by the UDL.
- size_t size

  Size of the buffer in bytes.

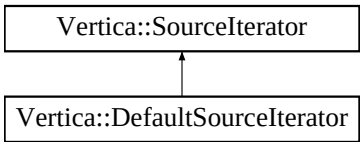
Detailed Description

DataBuffer

A contiguous in-memory buffer

Vertica::DefaultSourceIterator Class Reference

Inheritance diagram for Vertica::DefaultSourceIterator:



Public Member Functions

- **DefaultSourceIterator** (std::vector< UDSource * > sources)
- UnsizedUDSource * createNextSource (ServerInterface &srvInterface)

  Create the next UDSource to process.
- size_t getNumberOfSources ()
- size_t getSizeOfSource (size_t sourceNum)

Member Function Documentation

UnsizedUDSource* Vertica::DefaultSourceIterator::createNextSource (ServerInterface & *srvInterface*) `[inline], [virtual]`

Create the next UDSource to process.

Should return NULL if no further sources are available for processing.

Note that the previous Source may still be open and in use on a different thread when this function is called.

Returns

a new Source instance corresponding to a new input stream

Implements Vertica::SourceIterator.

size_t Vertica::DefaultSourceIterator::getNumberOfSources ( ) `[inline], [virtual]`

Returns

the total number of Sources that this factory will produce

Implements Vertica::SourceIterator.

size_t Vertica::DefaultSourceIterator::getSizeOfSource (size_t *sourceNum*)
`[inline], [virtual]`

Returns

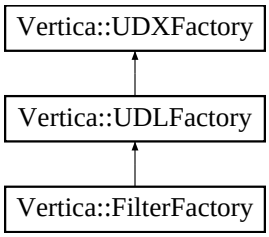
the raw-data size of the sourceNum'th source that will be produced by createNextSource(). Should return `vint_null` if the size is unknown.

This value is used as a hint, and is used by the "load_streams" table to display load progress. If incorrect or not set, "load_streams" may contain incorrect or unhelpful information.

Reimplemented from Vertica::SourceIterator.

Vertica::FilterFactory Class Reference

Inheritance diagram for Vertica::FilterFactory:



Public Types

- enum UDXType {
  FUNCTION, **TRANSFORM**, **ANALYTIC**, **MULTI_TRANSFORM**, **AGGREGATE**, **LOAD_SOURCE**, **LOAD_FILTER**, **LOAD_PARSER**, **TYPE** }

Public Member Functions

- virtual void getParameterType (ServerInterface &srvInterface, SizedColumnTypes &parameterTypes)
- virtual void getPerInstanceResources (ServerInterface &srvInterface, VResources &res)
- void getPrototype (ServerInterface &srvInterface, ColumnTypes &argTypes, ColumnTypes &returnType)
- virtual void getReturnType (ServerInterface &srvInterface, const SizedColumnTypes &argTypes, SizedColumnTypes &returnType)
- UDXFactory::UDXType getUDXFactoryType ()
- virtual void plan (ServerInterface &srvInterface, PlanContext &planCtxt)

- virtual UDFilter * prepare (ServerInterface &srvInterface, PlanContext &planCtxt)=0

Protected Attributes

- Oid **libOid**
- std::string **sqlName**

Detailed Description

Construct a single Filter.

Note that FilterFactories are singletons. Subclasses should be stateless, with no fields containing data, just methods. plan() and prepare() methods must never modify any global variables or state; they may only modify the variables that they are given as arguments. (If global state must be modified, use SourceIterator.)

Factories should be registered using the RegisterFactory() macro, defined in Vertica.h.

Member Enumeration Documentation

enum Vertica::UDXFactory::UDXType `[inherited]`

The type of UDX instance this factory produces

Member Function Documentation

virtual void Vertica::UDXFactory::getParameterType (ServerInterface & *srvInterface,* SizedColumnTypes & *parameterTypes*) `[inline], [virtual], [inherited]`

Function to tell Vertica the name and types of parameters that this function uses. Vertica will use this to warn function callers that certain parameters they provide are not affecting anything, or that certain parameters that are not being set are reverting to default values.

Reimplemented in Vertica::ParserFactory.

virtual void Vertica::UDXFactory::getPerInstanceResources (ServerInterface & *srvInterface,* VResources & *res*) `[inline], [virtual], [inherited]`

Set the resource required for each instance of the UDX Object subclass

Parameters

srvInterface	a ServerInterface object used to communicate with Vertica
res	a VResources object used to tell Vertica what resources are needed by the UDX

void Vertica::UDLFactory::getPrototype (ServerInterface & *srvInterface*, ColumnTypes & *argTypes*, ColumnTypes & *returnType*) [inline], [virtual], [inherited]

Provides the argument and return types of the UDL. UDL's take no input tuples; as such, their prototype is empty.

Implements Vertica::UDXFactory.

virtual void Vertica::UDLFactory::getReturnType (ServerInterface & *srvInterface*, const SizedColumnTypes & *argTypes*, SizedColumnTypes & *returnType*) [inline], [virtual], [inherited]

Not used in this form

Implements Vertica::UDXFactory.

UDXFactory::UDXType Vertica::FilterFactory::getUDXFactoryType ( ) [inline], [virtual]

Returns

the type of UDX Object instance this factory returns.

Note

User subclasses should use the appropriate subclass of UDXFactory and not override this method on their own.

Implements Vertica::UDXFactory.

virtual void Vertica::FilterFactory::plan (ServerInterface & *srvInterface*, PlanContext & *planCtxt*) [inline], [virtual]

Execute any planning logic required at query plan time. This method is run once per query, during query initialization. Its job is to perform parameter validation, and to modify the set of nodes that the COPY statement will run on (through srvInterface).

plan() runs exactly once per query, on the initiator node. If it throws an exception, the query will not proceed; it will be aborted prior to distributing the query to the other nodes and running prepare().

Parameters

srvInterface	Interface to server operations and functionality, including (not-per-column) parameter lookup
planCtxt	Context for storing and retrieving arbitrary data, for use just by this instance of this query. The same context is shared with plan().

virtual UDFilter* Vertica::FilterFactory::prepare (ServerInterface & *srvInterface*, PlanContext & *planCtxt*) `[pure virtual]`

Initialize a UDFilter. This function will be called on each node, prior to the Load operator starting to execute.

'planData' contains the same data that was placed there by the plan() static method.

Parameters

srvInterface	Interface to server operations and functionality, including (not-per-column) parameter lookup
planCtxt	Context for storing and retrieving arbitrary data, for use just by this instance of this query. The same context is shared with plan().

Returns

UDFilter instance to use for this query

Vertica::Flunion Union Reference

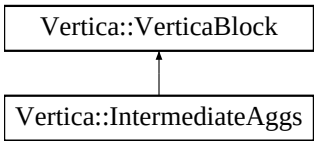
Public Attributes

- vfloat **vf**
- vint **vi**

Vertica::IntermediateAggs Class Reference

: A wrapper around a single intermediate aggregate value

Inheritance diagram for Vertica::IntermediateAggs:



Public Member Functions

- **IntermediateAggs** (size_t ninter)
- vbool * getBoolPtr (size_t idx)

  Get a pointer to a BOOLEAN value from the input row.
- vbool & getBoolRef (size_t idx)

  Get a reference to a BOOLEAN value from the input row.
- template<class T >
  const T * getColPtr (size_t idx)
- template<class T >
  T * **getColPtrForWrite** (size_t idx)
- template<class T >
  const T & getColRef (size_t idx)
- template<class T >
  T & **getColRefForWrite** (size_t idx)
- DateADT * getDatePtr (size_t idx)

  Get a pointer to a DATE value from the input row.
- DateADT & getDateRef (size_t idx)

  Get a reference to a DATE value from the input row.
- vfloat * getFloatPtr (size_t idx)

  Get a pointer to a FLOAT value from the input row.
- vfloat & getFloatRef (size_t idx)

  Get a reference to a FLOAT value from the input row.
- Interval * getIntervalPtr (size_t idx)

  Get a pointer to an INTERVAL value from the input row.
- Interval & getIntervalRef (size_t idx)

  Get a reference to an INTERVAL value from the input row.
- IntervalYM * getIntervalYMPtr (size_t idx)

  Get a pointer to a INTERVAL YEAR TO MONTH value from the input row.
- IntervalYM & getIntervalYMRef (size_t idx)

  Get a reference to an INTERVAL YEAR TO MONTH value from the input row.
- vint * getIntPtr (size_t idx)

  Get a pointer to an INTEGER value from the input row.
- vint & getIntRef (size_t idx)

  Get a reference to an INTEGER value from the input row.
- size_t getNumCols () const
- VNumeric * getNumericPtr (size_t idx)

  Get a pointer to a VNumeric value from the input row.
- VNumeric & getNumericRef (size_t idx)

  Get a reference to a VNumeric value from the input row.
- int getNumRows () const

- VString * getStringPtr (size_t idx)

  Get a pointer to a VString value from the input row.
- VString & getStringRef (size_t idx)

  Get a reference to an VString value from the input row.
- TimeADT * getTimePtr (size_t idx)

  Get a pointer to a TIME value from the input row.
- TimeADT & getTimeRef (size_t idx)

  Get a reference to a TIME value from the input row.
- Timestamp * getTimestampPtr (size_t idx)

  Get a pointer to a TIMESTAMP value from the input row.
- Timestamp & getTimestampRef (size_t idx)

  Get a reference to a TIMESTAMP value from the input row.
- TimestampTz * getTimestampTzPtr (size_t idx)

  Get a pointer to a TIMESTAMP WITH TIMEZONE value from the input row.
- TimestampTz & getTimestampTzRef (size_t idx)

  Get a reference to a TIMESTAMP WITH TIMEZONE value from the input row.
- TimeTzADT * getTimeTzPtr (size_t idx)

  Get a pointer to a TIME WITH TIMEZONE value from the input row.
- TimeTzADT & getTimeTzRef (size_t idx)

  Get a reference to a TIME WITH TIMEZONE value from the input row.
- const SizedColumnTypes & getTypeMetaData () const
- SizedColumnTypes & getTypeMetaData ()

Protected Member Functions

- void addCol (char *arg, int colstride, const VerticaType &dt, const std::string fieldName="")
- void **addCol** (const char *arg, int colstride, const VerticaType &dt, const std::string fieldName="")
- void **validateStringColumn** (size_t idx, const VString &s, const VerticaType &t)

Protected Attributes

- std::vector< char * > **cols**
- std::vector< int > **colstrides**
- int **count**
- int **index**
- size_t **ncols**
- std::vector< VString > **svWrappers**
- SizedColumnTypes **typeMetaData**
- std::vector< VNumeric > **vnWrappers**

Detailed Description

: A wrapper around a single intermediate aggregate value

Member Function Documentation

void Vertica::VerticaBlock::addCol (char * *arg,* int *colstride,* const VerticaType & *dt,* const std::string *fieldName* = "") `[inline],[protected],[inherited]`

Add the location for reading a particular argument.

Parameters

arg	The base location to find data.
colstride	The stride between data instances.
dt	The type of input.
fieldname	the name of the field

vbool* Vertica::IntermediateAggs::getBoolPtr (size_t *idx*) `[inline]`

Get a pointer to a BOOLEAN value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as a BOOLEAN.

vbool& Vertica::IntermediateAggs::getBoolRef (size_t *idx*) `[inline]`

Get a reference to a BOOLEAN value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as an BOOLEAN.

template<class T > const T* Vertica::VerticaBlock::getColPtr (size_t *idx*) `[inline],[inherited]`

Returns

a pointer to the idx'th argument, cast appropriately.

Example:

```
const vint *a = arg_reader->getColPtr<vint>(0);
```

template<class T > const T& Vertica::VerticaBlock::getColRef (size_t *idx*) `[inline],[inherited]`

Returns

a pointer to the idx'th argument, cast appropriately.

Example: const vint a = arg_reader->getColRef<vint>(0);

DateADT* Vertica::IntermediateAggs::getDatePtr (size_t *idx*) `[inline]`

Get a pointer to a DATE value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as a DATE.

DateADT& Vertica::IntermediateAggs::getDateRef (size_t *idx*) `[inline]`

Get a reference to a DATE value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as an DATE.

vfloat* Vertica::IntermediateAggs::getFloatPtr (size_t *idx*) `[inline]`

Get a pointer to a FLOAT value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as a FLOAT.

vfloat& Vertica::IntermediateAggs::getFloatRef (size_t *idx*) `[inline]`

Get a reference to a FLOAT value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

A reference to the idx'th argument, cast as an FLOAT.

Interval* Vertica::IntermediateAggs::getIntervalPtr (size_t *idx*) `[inline]`

Get a pointer to an INTERVAL value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as an INTERVAL.

Interval& Vertica::IntermediateAggs::getIntervalRef (size_t *idx*) `[inline]`

Get a reference to an INTERVAL value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as an INTERVAL.

IntervalYM* Vertica::IntermediateAggs::getIntervalYMPtr (size_t *idx*) [inline]

Get a pointer to a INTERVAL YEAR TO MONTH value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A point to the retrieved value cast as a INTERVAL YEAR TO MONTH.

IntervalYM& Vertica::IntermediateAggs::getIntervalYMRef (size_t *idx*) [inline]

Get a reference to an INTERVAL YEAR TO MONTH value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as an INTERVAL YEAR TO MONTH.

vint* Vertica::IntermediateAggs::getIntPtr (size_t *idx*) [inline]

Get a pointer to an INTEGER value from the input row.

Returns

a pointer to the idx'th argument, cast appropriately.

Parameters

idx	The column number to retrieve from the input row.

Example:

```
vint *a = arg_reader->getIntPtr(0);
```

vint& Vertica::IntermediateAggs::getIntRef (size_t *idx*) [inline]

Get a reference to an INTEGER value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as an INTEGER.

Example:

```
vint a = arg_reader->getIntRef(0);
```

size_t Vertica::VerticaBlock::getNumCols ( ) const [inline], [inherited]

Returns

the number of columns held by this block.

VNumeric* Vertica::IntermediateAggs::getNumericPtr (size_t *idx*) [inline]

Get a pointer to a VNumeric value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

A pointer to the retrieved value cast as a Numeric.

VNumeric& Vertica::IntermediateAggs::getNumericRef (size_t *idx*) [inline]

Get a reference to a VNumeric value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as an VNumeric.

int Vertica::VerticaBlock::getNumRows ( ) const `[inline], [inherited]`

Returns

the number of rows held by this block.

VString* Vertica::IntermediateAggs::getStringPtr (size_t *idx*) `[inline]`

Get a pointer to a VString value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

A pointer to the retrieved value cast as a VString.

VString& Vertica::IntermediateAggs::getStringRef (size_t *idx*) `[inline]`

Get a reference to an VString value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as an VString.

TimeADT* Vertica::IntermediateAggs::getTimePtr (size_t *idx*) `[inline]`

Get a pointer to a TIME value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as a TIME.

TimeADT& Vertica::IntermediateAggs::getTimeRef (size_t *idx*) `[inline]`

Get a reference to a TIME value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as a TIME.

Timestamp* Vertica::IntermediateAggs::getTimestampPtr (size_t *idx*) `[inline]`

Get a pointer to a TIMESTAMP value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as a TIMESTAMP.

Timestamp& Vertica::IntermediateAggs::getTimestampRef (size_t *idx*) `[inline]`

Get a reference to a TIMESTAMP value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as a TIMESTAMP.

TimestampTz* Vertica::IntermediateAggs::getTimestampTzPtr (size_t *idx*) [inline]

Get a pointer to a TIMESTAMP WITH TIMEZONE value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as a TIMESTAMP WITH TIMEZONE .

TimestampTz& Vertica::IntermediateAggs::getTimestampTzRef (size_t *idx*) [inline]

Get a reference to a TIMESTAMP WITH TIMEZONE value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as a TIMESTAMP WITH TIMEZONE.

TimeTzADT* Vertica::IntermediateAggs::getTimeTzPtr (size_t *idx*) [inline]

Get a pointer to a TIME WITH TIMEZONE value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as a TIME WITH TIMEZONE.

TimeTzADT& Vertica::IntermediateAggs::getTimeTzRef (size_t *idx*) `[inline]`

Get a reference to a TIME WITH TIMEZONE value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as a TIME WITH TIMEZONE.

const SizedColumnTypes& Vertica::VerticaBlock::getTypeMetaData ( ) const `[inline]`, `[inherited]`

Returns

information about the types and numbers of arguments

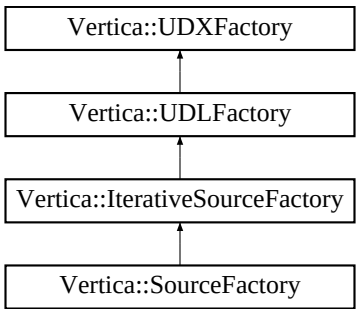
SizedColumnTypes& Vertica::VerticaBlock::getTypeMetaData ( ) `[inline]`, `[inherited]`

Returns

information about the types and numbers of arguments

Vertica::IterativeSourceFactory Class Reference

Inheritance diagram for Vertica::IterativeSourceFactory:



Public Types

- enum UDXType {
  FUNCTION, **TRANSFORM**, **ANALYTIC**, **MULTI_TRANSFORM**, **AGGREGATE**, **LOAD_SOURCE**, **LOAD_FILTER**, **LOAD_PARSER**, **TYPE** }

Public Member Functions

- virtual void getParameterType (ServerInterface &srvInterface, SizedColumnTypes &parameterTypes)
- virtual void getPerInstanceResources (ServerInterface &srvInterface, VResources &res)
- void getPrototype (ServerInterface &srvInterface, ColumnTypes &argTypes, ColumnTypes &returnType)
- virtual void getReturnType (ServerInterface &srvInterface, const SizedColumnTypes &argTypes, SizedColumnTypes &returnType)
- UDXFactory::UDXType getUDXFactoryType ()
- virtual void plan (ServerInterface &srvInterface, NodeSpecifyingPlanContext &planCtxt)
- virtual SourceIterator * prepare (ServerInterface &srvInterface, NodeSpecifyingPlanContext &planCtxt)=0

Protected Attributes

- Oid **libOid**
- std::string **sqlName**

Detailed Description

High-level initialization required by a UDSource.

Performs initial validation and planning of the query, before it is distributed over the network. Also instantiates objects to perform further initialization on each node, once the query has been distributed.

Note that SourceFactories are singletons. Subclasses should be stateless, with no fields containing data, just methods. plan() and prepare() methods must never modify any global variables or state; they may only modify the variables that they are given as arguments. (If global state must be modified, use SourceIterator.)

Factories should be registered using the RegisterFactory() macro, defined in Vertica.h.

Member Enumeration Documentation

enum Vertica::UDXFactory::UDXType `[inherited]`

The type of UDX instance this factory produces

Member Function Documentation

virtual void Vertica::UDXFactory::getParameterType (ServerInterface & *srvInterface,* SizedColumnTypes & *parameterTypes*) `[inline], [virtual], [inherited]`

Function to tell Vertica the name and types of parameters that this function uses. Vertica will use this to warn function callers that certain parameters they provide are not affecting anything, or that certain parameters that are not being set are reverting to default values.

Reimplemented in Vertica::ParserFactory.

virtual void Vertica::UDXFactory::getPerInstanceResources (ServerInterface & *srvInterface,* VResources & *res*) `[inline], [virtual], [inherited]`

Set the resource required for each instance of the UDX Object subclass

Parameters

srvInterface	a ServerInterface object used to communicate with Vertica
res	a VResources object used to tell Vertica what resources are needed by the UDX

void Vertica::UDLFactory::getPrototype (ServerInterface & *srvInterface,* ColumnTypes & *argTypes,* ColumnTypes & *returnType*) `[inline], [virtual], [inherited]`

Provides the argument and return types of the UDL. UDL's take no input tuples; as such, their prototype is empty.

Implements Vertica::UDXFactory.

virtual void Vertica::UDLFactory::getReturnType (ServerInterface & *srvInterface,* const SizedColumnTypes & *argTypes,* SizedColumnTypes & *returnType*) `[inline], [virtual], [inherited]`

Not used in this form

Implements Vertica::UDXFactory.

UDXFactory::UDXType Vertica::IterativeSourceFactory::getUDXFactoryType ( ) `[inline], [virtual]`

Returns

the type of UDX Object instance this factory returns.

Note

User subclasses should use the appropriate subclass of UDXFactory and not override this method on their own.

Implements Vertica::UDXFactory.

virtual void Vertica::IterativeSourceFactory::plan (ServerInterface & *srvInterface*, NodeSpecifyingPlanContext & *planCtxt*) [inline], [virtual]

Execute any planning logic required at query plan time. This method is run once per query, during query initialization. Its job is to perform parameter validation, and to modify the set of nodes that the COPY statement will run on.

plan() runs exactly once per query, on the initiator node. If it throws an exception, the query will not proceed; it will be aborted prior to distributing the query to the other nodes and running prepare().

Reimplemented in Vertica::SourceFactory.

virtual SourceIterator* Vertica::IterativeSourceFactory::prepare (ServerInterface & *srvInterface*, NodeSpecifyingPlanContext & *planCtxt*) [pure virtual]

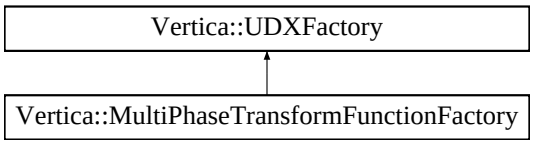
Prepare this SourceFactory to start creating sources. This function will be called on each node, prior to the Load operator starting to execute and prior to any other virtual functions on this class being called.

'planData' contains the same data that was placed there by the plan() static method.

If necessary, it is safe for this method to store any of the argument references as local fields on this instance. All will persist for the duration of the query.

Vertica::MultiPhaseTransformFunctionFactory Class Reference

Inheritance diagram for Vertica::MultiPhaseTransformFunctionFactory:



Public Types

- enum UDXType {
  FUNCTION, **TRANSFORM**, **ANALYTIC**, **MULTI_TRANSFORM**, **AGGREGATE**, **LOAD_SOURCE**, **LOAD_FILTER**, **LOAD_PARSER**, **TYPE** }

Public Member Functions

- virtual void getParameterType (ServerInterface &srvInterface, SizedColumnTypes &parameterTypes)
- virtual void getPerInstanceResources (ServerInterface &srvInterface, VResources &res)
- virtual void getPhases (ServerInterface &srvInterface, std::vector< TransformFunctionPhase * > &phases)=0
- virtual void getPrototype (ServerInterface &srvInterface, ColumnTypes &argTypes, ColumnTypes &returnType)
- virtual void getReturnType (ServerInterface &srvInterface, const SizedColumnTypes &argTypes, SizedColumnTypes &returnType)

Protected Member Functions

- virtual UDXType getUDXFactoryType ()

Protected Attributes

- Oid **libOid**
- std::string **sqlName**

Member Enumeration Documentation

enum Vertica::UDXFactory::UDXType `[inherited]`

The type of UDX instance this factory produces

Member Function Documentation

virtual void Vertica::UDXFactory::getParameterType (ServerInterface & *srvInterface,* SizedColumnTypes & *parameterTypes*) `[inline], [virtual], [inherited]`

Function to tell Vertica the name and types of parameters that this function uses. Vertica will use this to warn function callers that certain parameters they provide are not affecting anything, or that certain parameters that are not being set are reverting to default values.

Reimplemented in Vertica::ParserFactory.

virtual void Vertica::UDXFactory::getPerInstanceResources (ServerInterface & *srvInterface,* VResources & *res*) `[inline], [virtual], [inherited]`

Set the resource required for each instance of the UDX Object subclass

Parameters

srvInterface	a ServerInterface object used to communicate with Vertica
res	a VResources object used to tell Vertica what resources are needed by the UDX

virtual void Vertica::MultiPhaseTransformFunctionFactory::getPhases (ServerInterface & *srvInterface*, std::vector< TransformFunctionPhase * > & *phases*) `[pure virtual]`

Returns a vector of pointers to TransformFunctionPhase objects that represent the various phases of computation

virtual void Vertica::MultiPhaseTransformFunctionFactory::getPrototype (ServerInterface & *srvInterface*, ColumnTypes & *argTypes*, ColumnTypes & *returnType*) `[inline], [virtual]`

Provides the argument and return types of the UDX

Implements Vertica::UDXFactory.

virtual void Vertica::MultiPhaseTransformFunctionFactory::getReturnType (ServerInterface & *srvInterface*, const SizedColumnTypes & *argTypes*, SizedColumnTypes & *returnType*) `[inline], [virtual]`

Function to tell Vertica what the return types (and length/precision if necessary) of this UDX are.

For CHAR/VARCHAR types, specify the max length,

For NUMERIC types, specify the precision and scale.

For Time/Timestamp types (with or without time zone), specify the precision, -1 means unspecified/don't care

For IntervalYM/IntervalDS types, specify the precision and range

For all other types, no length/precision specification needed

Parameters

argTypes	Provides the data types of arguments that this UDT was called with. This may be used to modify the return types accordingly.
returnType	User code must fill in the names and data types returned by the UDT.

Implements Vertica::UDXFactory.

virtual UDXType Vertica::MultiPhaseTransformFunctionFactory::getUDXFactoryType ( ) [inline],[protected],[virtual]

Returns

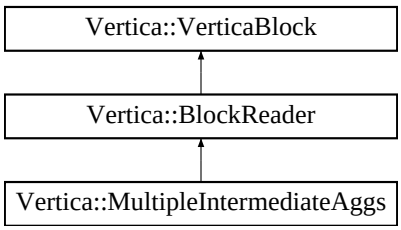
the object type internally used by Vertica

Implements Vertica::UDXFactory.

Vertica::MultipleIntermediateAggs Class Reference

: A wrapper around multiple intermediate aggregates

Inheritance diagram for Vertica::MultipleIntermediateAggs:



Public Member Functions

- **MultipleIntermediateAggs** (size_t narg)
- const vbool * getBoolPtr (size_t idx)

  Get a pointer to a BOOLEAN value from the input row.
- const vbool & getBoolRef (size_t idx)

  Get a reference to a BOOLEAN value from the input row.
- template<class T >
  const T * getColPtr (size_t idx)
- template<class T >
  T * **getColPtrForWrite** (size_t idx)
- template<class T >
  const T & getColRef (size_t idx)
- template<class T >
  T & **getColRefForWrite** (size_t idx)
- const DateADT * getDatePtr (size_t idx)

  Get a pointer to a DATE value from the input row.
- const DateADT & getDateRef (size_t idx)

  Get a reference to a DATE value from the input row.
- const vfloat * getFloatPtr (size_t idx)

  Get a pointer to a FLOAT value from the input row.

- const vfloat & getFloatRef (size_t idx)

  Get a reference to a FLOAT value from the input row.
- const Interval * getIntervalPtr (size_t idx)

  Get a pointer to an INTERVAL value from the input row.
- const Interval & getIntervalRef (size_t idx)

  Get a reference to an INTERVAL value from the input row.
- const IntervalYM * getIntervalYMPtr (size_t idx)

  Get a pointer to a INTERVAL YEAR TO MONTH value from the input row.
- const IntervalYM & getIntervalYMRef (size_t idx)

  Get a reference to an INTERVAL YEAR TO MONTH value from the input row.
- const vint * getIntPtr (size_t idx)

  Get a pointer to an INTEGER value from the input row.
- const vint & getIntRef (size_t idx)

  Get a reference to an INTEGER value from the input row.
- size_t getNumCols () const
- const VNumeric * getNumericPtr (size_t idx)

  Get a pointer to a VNumeric value from the input row.
- const VNumeric & getNumericRef (size_t idx)

  Get a reference to a VNumeric value from the input row.
- int getNumRows () const
- const VString * getStringPtr (size_t idx)

  Get a pointer to a VString value from the input row.
- const VString & getStringRef (size_t idx)

  Get a reference to an VString value from the input row.
- const TimeADT * getTimePtr (size_t idx)

  Get a pointer to a TIME value from the input row.
- const TimeADT & getTimeRef (size_t idx)

  Get a reference to a TIME value from the input row.
- const Timestamp * getTimestampPtr (size_t idx)

  Get a pointer to a TIMESTAMP value from the input row.
- const Timestamp & getTimestampRef (size_t idx)

  Get a reference to a TIMESTAMP value from the input row.
- const TimestampTz * getTimestampTzPtr (size_t idx)

  Get a pointer to a TIMESTAMP WITH TIMEZONE value from the input row.
- const TimestampTz & getTimestampTzRef (size_t idx)

  Get a reference to a TIMESTAMP WITH TIMEZONE value from the input row.
- const TimeTzADT * getTimeTzPtr (size_t idx)

  Get a pointer to a TIME WITH TIMEZONE value from the input row.
- const TimeTzADT & getTimeTzRef (size_t idx)

Get a reference to a TIME WITH TIMEZONE value from the input row.

- const SizedColumnTypes & getTypeMetaData () const
- SizedColumnTypes & getTypeMetaData ()
- bool isNull (int col)

  Check if the idx'th argument is null.
- bool next ()

Protected Member Functions

- void addCol (char *arg, int colstride, const VerticaType &dt, const std::string fieldName="")
- void **addCol** (const char *arg, int colstride, const VerticaType &dt, const std::string fieldName="")
- void **validateStringColumn** (size_t idx, const VString &s, const VerticaType &t)

Protected Attributes

- std::vector< char * > **cols**
- std::vector< int > **colstrides**
- int **count**
- int **index**
- size_t **ncols**
- std::vector< VString > **svWrappers**
- SizedColumnTypes **typeMetaData**
- std::vector< VNumeric > **vnWrappers**

Detailed Description

: A wrapper around multiple intermediate aggregates

Member Function Documentation

void Vertica::VerticaBlock::addCol (char * *arg*, int *colstride*, const VerticaType & *dt*, const std::string *fieldName* = "") `[inline],[protected],[inherited]`

Add the location for reading a particular argument.

Parameters

arg	The base location to find data.
colstride	The stride between data instances.
dt	The type of input.
fieldname	the name of the field

const vbool* Vertica::BlockReader::getBoolPtr (size_t *idx*) `[inline], [inherited]`

Get a pointer to a BOOLEAN value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as a BOOLEAN.

const vbool& Vertica::BlockReader::getBoolRef (size_t *idx*) `[inline], [inherited]`

Get a reference to a BOOLEAN value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as an BOOLEAN.

template<class T > const T* Vertica::VerticaBlock::getColPtr (size_t *idx*) `[inline], [inherited]`

Returns

a pointer to the idx'th argument, cast appropriately.

Example:

```
const vint *a = arg_reader->getColPtr<vint>(0);
```

template<class T > const T& Vertica::VerticaBlock::getColRef (size_t *idx*) `[inline], [inherited]`

Returns

a pointer to the idx'th argument, cast appropriately.

Example: const vint a = arg_reader->getColRef<vint>(0);

const DateADT* Vertica::BlockReader::getDatePtr (size_t *idx*) `[inline], [inherited]`

Get a pointer to a DATE value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as a DATE.

const DateADT& Vertica::BlockReader::getDateRef (size_t *idx*) `[inline], [inherited]`

Get a reference to a DATE value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as an DATE.

const vfloat* Vertica::BlockReader::getFloatPtr (size_t *idx*) `[inline], [inherited]`

Get a pointer to a FLOAT value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as a FLOAT.

const vfloat& Vertica::BlockReader::getFloatRef (size_t *idx*) `[inline], [inherited]`

Get a reference to a FLOAT value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

A reference to the idx'th argument, cast as an FLOAT.

const Interval* Vertica::BlockReader::getIntervalPtr (size_t *idx*) `[inline], [inherited]`

Get a pointer to an INTERVAL value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as an INTERVAL.

const Interval& Vertica::BlockReader::getIntervalRef (size_t *idx*) `[inline], [inherited]`

Get a reference to an INTERVAL value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as an INTERVAL.

const IntervalYM* Vertica::BlockReader::getIntervalYMPtr (size_t *idx*) `[inline], [inherited]`

Get a pointer to a INTERVAL YEAR TO MONTH value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A point to the retrieved value cast as a INTERVAL YEAR TO MONTH.

const IntervalYM& Vertica::BlockReader::getIntervalYMRef (size_t *idx*) [inline], [inherited]

Get a reference to an INTERVAL YEAR TO MONTH value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as an INTERVAL YEAR TO MONTH.

const vint* Vertica::BlockReader::getIntPtr (size_t *idx*) [inline], [inherited]

Get a pointer to an INTEGER value from the input row.

Returns

a pointer to the idx'th argument, cast appropriately.

Parameters

idx	The column number to retrieve from the input row.

Example:

```
const vint *a = arg_reader->getIntPtr(0);
```

const vint& Vertica::BlockReader::getIntRef (size_t *idx*) [inline], [inherited]

Get a reference to an INTEGER value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as an INTEGER.

Example:

```
const vint a = arg_reader->getIntRef(0);
```

size_t Vertica::VerticaBlock::getNumCols ( ) const [inline], [inherited]

Returns

the number of columns held by this block.

const VNumeric* Vertica::BlockReader::getNumericPtr (size_t *idx*) [inline], [inherited]

Get a pointer to a VNumeric value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

A pointer to the retrieved value cast as a Numeric.

const VNumeric& Vertica::BlockReader::getNumericRef (size_t *idx*) [inline], [inherited]

Get a reference to a VNumeric value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as an VNumeric.

int Vertica::VerticaBlock::getNumRows ( ) const [inline], [inherited]

Returns

the number of rows held by this block.

const VString* Vertica::BlockReader::getStringPtr (size_t *idx*) `[inline], [inherited]`

Get a pointer to a VString value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

A pointer to the retrieved value cast as a VString.

const VString& Vertica::BlockReader::getStringRef (size_t *idx*) `[inline], [inherited]`

Get a reference to an VString value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as an VString.

const TimeADT* Vertica::BlockReader::getTimePtr (size_t *idx*) `[inline], [inherited]`

Get a pointer to a TIME value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as a TIME.

const TimeADT& Vertica::BlockReader::getTimeRef (size_t *idx*) `[inline], [inherited]`

Get a reference to a TIME value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as a TIME.

const Timestamp* Vertica::BlockReader::getTimestampPtr (size_t *idx*) `[inline], [inherited]`

Get a pointer to a TIMESTAMP value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as a TIMESTAMP.

const Timestamp& Vertica::BlockReader::getTimestampRef (size_t *idx*) `[inline], [inherited]`

Get a reference to a TIMESTAMP value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as a TIMESTAMP.

const TimestampTz* Vertica::BlockReader::getTimestampTzPtr (size_t *idx*) `[inline], [inherited]`

Get a pointer to a TIMESTAMP WITH TIMEZONE value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as a TIMESTAMP WITH TIMEZONE .

const TimestampTz& Vertica::BlockReader::getTimestampTzRef (size_t *idx*) `[inline], [inherited]`

Get a reference to a TIMESTAMP WITH TIMEZONE value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as a TIMESTAMP WITH TIMEZONE.

const TimeTzADT* Vertica::BlockReader::getTimeTzPtr (size_t *idx*) `[inline], [inherited]`

Get a pointer to a TIME WITH TIMEZONE value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as a TIME WITH TIMEZONE.

const TimeTzADT& Vertica::BlockReader::getTimeTzRef (size_t *idx*) `[inline], [inherited]`

Get a reference to a TIME WITH TIMEZONE value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as a TIME WITH TIMEZONE.

const SizedColumnTypes& Vertica::VerticaBlock::getTypeMetaData ( ) const `[inline],[inherited]`

Returns

information about the types and numbers of arguments

SizedColumnTypes& Vertica::VerticaBlock::getTypeMetaData ( ) `[inline], [inherited]`

Returns

information about the types and numbers of arguments

bool Vertica::BlockReader::isNull (int *col*) `[inline],[inherited]`

Check if the idx'th argument is null.

Parameters

col	The column number in the row to check for null

Returns

true is the col value is null false otherwise

bool Vertica::BlockReader::next ( ) `[inline],[inherited]`

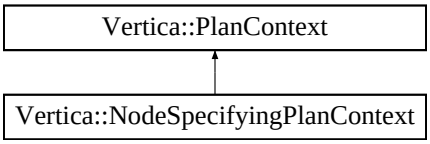
Advance to the next record.

Returns

true if there are more rows to read, false otherwise.

Vertica::NodeSpecifyingPlanContext Class Reference

Inheritance diagram for Vertica::NodeSpecifyingPlanContext:



Public Member Functions

- **NodeSpecifyingPlanContext** (ParamWriter &writer, std::vector< std::string > clusterNodes, std::vector< std::string > targetNodes)
- **NodeSpecifyingPlanContext** (ParamWriter &writer, std::vector< std::string > clusterNodes)
- **NodeSpecifyingPlanContext** (ParamWriter &writer)
- const std::vector< std::string > & getClusterNodes ()
- ParamReader & getReader ()
- const std::vector< std::string > & getTargetNodes () const
- ParamWriter & getWriter ()
- void setTargetNodes (const std::vector< std::string > &nodes)

Detailed Description

Interface that allows storage of query-plan state, when different parts of query planning take place on different computers. For example, if some work is done on the query initiator node and some is done on each node executing the query.

In addition to the functionality provided by PlanContext, NodeSpecifyingPlanContext allows you to specify which nodes the query should run on.

Member Function Documentation

const std::vector<std::string>& Vertica::PlanContext::getClusterNodes ( ) `[inline], [inherited]`

Get a list of all of the nodes in the current cluster, by node name

ParamReader& Vertica::PlanContext::getReader ( ) `[inline], [inherited]`

Get a read-only instance of the current context

const std::vector<std::string>& Vertica::NodeSpecifyingPlanContext::getTargetNodes ( ) const `[inline]`

Return the set of nodes that this query is currently set to run on

ParamWriter& Vertica::PlanContext::getWriter ( ) [inline],[inherited]

Get the current context for writing

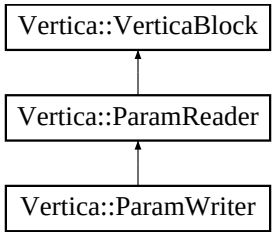
void Vertica::NodeSpecifyingPlanContext::setTargetNodes (const std::vector< std::string > & *nodes*) [inline]

Change the set of nodes that the query is intended to run on. Throws UnknownNodeException if any of the specified node names is not actually the name of any node in the cluster.

Vertica::ParamReader Class Reference

: A wrapper around Parameters that have a name->value correspondence

Inheritance diagram for Vertica::ParamReader:



Public Member Functions

- **ParamReader** (size_t nparams)
- void addParameter (std::string paramName, const char *arg, const VerticaType &dt)
- bool containsParameter (std::string paramName)

  Function to see if the ParamReader has a value for the parameter.
- const vbool * getBoolPtr (std::string paramName)

  Get a pointer to a BOOLEAN value from the input row.
- const vbool & getBoolRef (std::string paramName)

  Get a reference to a BOOLEAN value from the input row.
- template<class T >
  const T * getColPtr (size_t idx)
- template<class T >
  T * **getColPtrForWrite** (size_t idx)
- template<class T >
  const T & getColRef (size_t idx)
- template<class T >
  T & **getColRefForWrite** (size_t idx)

- const DateADT * getDatePtr (std::string paramName)

  Get a pointer to a DATE value from the input row.
- const DateADT & getDateRef (std::string paramName)

  Get a reference to a DATE value from the input row.
- const vfloat * getFloatPtr (std::string paramName)

  Get a pointer to a FLOAT value from the input row.
- const vfloat & getFloatRef (std::string paramName)

  Get a reference to a FLOAT value from the input row.
- size_t **getIndex** (std::string paramName)
- const Interval * getIntervalPtr (std::string paramName)

  Get a pointer to an INTERVAL value from the input row.
- const Interval & getIntervalRef (std::string paramName)

  Get a reference to an INTERVAL value from the input row.
- const IntervalYM * getIntervalYMPtr (std::string paramName)

  Get a pointer to a INTERVAL YEAR TO MONTH value from the input row.
- const IntervalYM & getIntervalYMRef (std::string paramName)

  Get a reference to an INTERVAL YEAR TO MONTH value from the input row.
- const vint * getIntPtr (std::string paramName)

  Get a pointer to an INTEGER value from the input row.
- const vint & getIntRef (std::string paramName)

  Get a reference to an INTEGER value from the input row.
- size_t getNumCols () const
- const VNumeric * getNumericPtr (std::string paramName)

  Get a pointer to a VNumeric value from the input row.
- const VNumeric & getNumericRef (std::string paramName)

  Get a reference to a VNumeric value from the input row.
- int getNumRows () const
- std::vector< std::string > getParamNames ()

  Return all names of parameters stored in this ParamReader.
- const VString * getStringPtr (std::string paramName)

  Get a pointer to a VString value from the input row.
- const VString & getStringRef (std::string paramName)

  Get a reference to an VString value from the input row.
- const TimeADT * getTimePtr (std::string paramName)

  Get a pointer to a TIME value from the input row.
- const TimeADT & getTimeRef (std::string paramName)

  Get a reference to a TIME value from the input row.
- const Timestamp * getTimestampPtr (std::string paramName)

  Get a pointer to a TIMESTAMP value from the input row.

- const Timestamp & getTimestampRef (std::string paramName)

  Get a reference to a TIMESTAMP value from the input row.
- const TimestampTz * getTimestampTzPtr (std::string paramName)

  Get a pointer to a TIMESTAMP WITH TIMEZONE value from the input row.
- const TimestampTz & getTimestampTzRef (std::string paramName)

  Get a reference to a TIMESTAMP WITH TIMEZONE value from the input row.
- const TimeTzADT * getTimeTzPtr (std::string paramName)

  Get a pointer to a TIME WITH TIMEZONE value from the input row.
- const TimeTzADT & getTimeTzRef (std::string paramName)

  Get a reference to a TIME WITH TIMEZONE value from the input row.
- VerticaType getType (std::string paramName)

  Return the type of the given parameter.
- const SizedColumnTypes & getTypeMetaData () const
- SizedColumnTypes & getTypeMetaData ()
- bool isEmpty () const

  Returns true if there are no parameters.

Public Attributes

- std::map< std::string, size_t > paramNameToIndex

Protected Member Functions

- void addCol (char *arg, int colstride, const VerticaType &dt, const std::string fieldName="")
- void **addCol** (const char *arg, int colstride, const VerticaType &dt, const std::string fieldName="")
- void **validateStringColumn** (size_t idx, const VString &s, const VerticaType &t)

Protected Attributes

- std::vector< char * > **cols**
- std::vector< int > **colstrides**
- int **count**
- int **index**
- size_t **ncols**
- std::vector< VString > **svWrappers**
- SizedColumnTypes **typeMetaData**
- std::vector< VNumeric > **vnWrappers**

Friends

- class **VerticaBlockSerializer**

Detailed Description

: A wrapper around Parameters that have a name->value correspondence

Member Function Documentation

void Vertica::VerticaBlock::addCol (char * *arg*, int *colstride*, const VerticaType & *dt*, const std::string *fieldName* = "") `[inline],[protected],[inherited]`

Add the location for reading a particular argument.

Parameters

arg	The base location to find data.
colstride	The stride between data instances.
dt	The type of input.
fieldname	the name of the field

void Vertica::ParamReader::addParameter (std::string *paramName*, const char * *arg*, const VerticaType & *dt*) `[inline]`

Add a parameter to the block and stores it name and corresponding index in the paramNameToIndex map

const vbool* Vertica::ParamReader::getBoolPtr (std::string *paramName*) `[inline]`

Get a pointer to a BOOLEAN value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

A pointer to the retrieved value cast as a BOOLEAN.

const vbool& Vertica::ParamReader::getBoolRef (std::string *paramName*)
`[inline]`

Get a reference to a BOOLEAN value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

a reference to the parameter value, cast as an BOOLEAN.

template<class T > const T* Vertica::VerticaBlock::getColPtr (size_t *idx*)
`[inline], [inherited]`

Returns

a pointer to the idx'th argument, cast appropriately.

Example:

```
const vint *a = arg_reader->getColPtr<vint>(0);
```

template<class T > const T& Vertica::VerticaBlock::getColRef (size_t *idx*)
`[inline], [inherited]`

Returns

a pointer to the idx'th argument, cast appropriately.

Example: const vint a = arg_reader->getColRef<vint>(0);

const DateADT* Vertica::ParamReader::getDatePtr (std::string *paramName*)
`[inline]`

Get a pointer to a DATE value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

A pointer to the retrieved value cast as a DATE.

const DateADT& Vertica::ParamReader::getDateRef (std::string *paramName*) [inline]

Get a reference to a DATE value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

a reference to the parameter value, cast as an DATE.

const vfloat* Vertica::ParamReader::getFloatPtr (std::string *paramName*) [inline]

Get a pointer to a FLOAT value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

A pointer to the retrieved value cast as a FLOAT.

const vfloat& Vertica::ParamReader::getFloatRef (std::string *paramName*) [inline]

Get a reference to a FLOAT value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

A reference to the parameter value, cast as an FLOAT.

const Interval* Vertica::ParamReader::getIntervalPtr (std::string *paramName*) `[inline]`

Get a pointer to an INTERVAL value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

A pointer to the retrieved value cast as an INTERVAL.

const Interval& Vertica::ParamReader::getIntervalRef (std::string *paramName*) `[inline]`

Get a reference to an INTERVAL value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

a reference to the parameter value, cast as an INTERVAL.

const IntervalYM* Vertica::ParamReader::getIntervalYMPtr (std::string *paramName*) `[inline]`

Get a pointer to a INTERVAL YEAR TO MONTH value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

A point to the retrieved value cast as a INTERVAL YEAR TO MONTH.

const IntervalYM& Vertica::ParamReader::getIntervalYMRef (std::string *paramName*) `[inline]`

Get a reference to an INTERVAL YEAR TO MONTH value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

a reference to the parameter value, cast as an INTERVAL YEAR TO MONTH.

const vint* Vertica::ParamReader::getIntPtr (std::string *paramName*) `[inline]`

Get a pointer to an INTEGER value from the input row.

Returns

a pointer to the idx'th argument, cast appropriately.

Parameters

paramName	The name of the parameter to retrieve

Example:

```
vint *a = arg_reader->getIntPtr("max");
```

const vint& Vertica::ParamReader::getIntRef (std::string *paramName*) `[inline]`

Get a reference to an INTEGER value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

a reference to the parameter value, cast as an INTEGER.

Example:

```
vint a = arg_reader->getIntRef("max");
```

size_t Vertica::VerticaBlock::getNumCols ( ) const `[inline], [inherited]`

Returns

the number of columns held by this block.

const VNumeric* Vertica::ParamReader::getNumericPtr (std::string *paramName*) [inline]

Get a pointer to a VNumeric value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

A pointer to the retrieved value cast as a Numeric.

const VNumeric& Vertica::ParamReader::getNumericRef (std::string *paramName*) [inline]

Get a reference to a VNumeric value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

a reference to the parameter value, cast as an VNumeric.

int Vertica::VerticaBlock::getNumRows ( ) const [inline], [inherited]

Returns

the number of rows held by this block.

const VString* Vertica::ParamReader::getStringPtr (std::string *paramName*) [inline]

Get a pointer to a VString value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

A pointer to the retrieved value cast as a VString.

const VString& Vertica::ParamReader::getStringRef (std::string *paramName*) [inline]

Get a reference to an VString value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

a reference to the parameter value, cast as an VString.

const TimeADT* Vertica::ParamReader::getTimePtr (std::string *paramName*) [inline]

Get a pointer to a TIME value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

A pointer to the retrieved value cast as a TIME.

const TimeADT& Vertica::ParamReader::getTimeRef (std::string *paramName*) [inline]

Get a reference to a TIME value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

a reference to the parameter value, cast as a TIME.

const Timestamp* Vertica::ParamReader::getTimestampPtr (std::string *paramName*) `[inline]`

Get a pointer to a TIMESTAMP value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

A pointer to the retrieved value cast as a TIMESTAMP.

const Timestamp& Vertica::ParamReader::getTimestampRef (std::string *paramName*) `[inline]`

Get a reference to a TIMESTAMP value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

a reference to the parameter value, cast as a TIMESTAMP.

const TimestampTz* Vertica::ParamReader::getTimestampTzPtr (std::string *paramName*) `[inline]`

Get a pointer to a TIMESTAMP WITH TIMEZONE value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

A pointer to the retrieved value cast as a TIMESTAMP WITH TIMEZONE .

const TimestampTz& Vertica::ParamReader::getTimestampTzRef (std::string *paramName*) `[inline]`

Get a reference to a TIMESTAMP WITH TIMEZONE value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

a reference to the parameter value, cast as a TIMESTAMP WITH TIMEZONE.

const TimeTzADT* Vertica::ParamReader::getTimeTzPtr (std::string *paramName*) `[inline]`

Get a pointer to a TIME WITH TIMEZONE value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

A pointer to the retrieved value cast as a TIME WITH TIMEZONE.

const TimeTzADT& Vertica::ParamReader::getTimeTzRef (std::string *paramName*) `[inline]`

Get a reference to a TIME WITH TIMEZONE value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

a reference to the parameter value, cast as a TIME WITH TIMEZONE.

const SizedColumnTypes& Vertica::VerticaBlock::getTypeMetaData ( ) const `[inline],[inherited]`

Returns

information about the types and numbers of arguments

SizedColumnTypes& Vertica::VerticaBlock::getTypeMetaData ( ) [inline], [inherited]

Returns

information about the types and numbers of arguments

Member Data Documentation

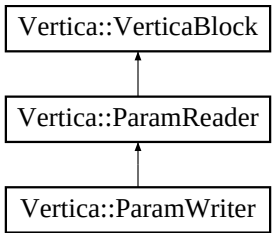
std::map<std::string, size_t> Vertica::ParamReader::paramNameToIndex

Bookkeepinp to make a parameter and its position in the block

Vertica::ParamWriter Class Reference

Iterator interface for writing rows to a Vertica block.

Inheritance diagram for Vertica::ParamWriter:



Public Member Functions

- **ParamWriter** (VTAllocator *allocator=NULL)
- void addParameter (std::string paramName, const char *arg, const VerticaType &dt)
- bool containsParameter (std::string paramName)

  Function to see if the ParamReader has a value for the parameter.
- const vbool * getBoolPtr (std::string paramName)

  Get a pointer to a BOOLEAN value from the input row.
- const vbool & getBoolRef (std::string paramName)

  Get a reference to a BOOLEAN value from the input row.
- template<class T >
  const T * getColPtr (size_t idx)

- template<class T >
  T * **getColPtrForWrite** (size_t idx)
- template<class T >
  const T & getColRef (size_t idx)
- template<class T >
  T & **getColRefForWrite** (size_t idx)
- const DateADT * getDatePtr (std::string paramName)

  Get a pointer to a DATE value from the input row.
- const DateADT & getDateRef (std::string paramName)

  Get a reference to a DATE value from the input row.
- const vfloat * getFloatPtr (std::string paramName)

  Get a pointer to a FLOAT value from the input row.
- const vfloat & getFloatRef (std::string paramName)

  Get a reference to a FLOAT value from the input row.
- size_t **getIndex** (std::string paramName)
- const Interval * getIntervalPtr (std::string paramName)

  Get a pointer to an INTERVAL value from the input row.
- const Interval & getIntervalRef (std::string paramName)

  Get a reference to an INTERVAL value from the input row.
- const IntervalYM * getIntervalYMPtr (std::string paramName)

  Get a pointer to a INTERVAL YEAR TO MONTH value from the input row.
- const IntervalYM & getIntervalYMRef (std::string paramName)

  Get a reference to an INTERVAL YEAR TO MONTH value from the input row.
- const vint * getIntPtr (std::string paramName)

  Get a pointer to an INTEGER value from the input row.
- const vint & getIntRef (std::string paramName)

  Get a reference to an INTEGER value from the input row.
- VString & getLongStringRef (std::string fieldName)

  Allocates a new VString object to use as output. Sets it to be a 32mb LONG type by default.
- size_t getNumCols () const
- const VNumeric * getNumericPtr (std::string paramName)

  Get a pointer to a VNumeric value from the input row.
- VNumeric & getNumericRef (std::string fieldName)

  Allocate a new VNumeric object to use as output.
- int getNumRows () const
- std::vector< std::string > getParamNames ()

  Return all names of parameters stored in this ParamReader.
- const VString * getStringPtr (std::string paramName)

  Get a pointer to a VString value from the input row.

- VString & getStringRef (std::string fieldName)

  Allocates a new VString object to use as output.
- const TimeADT * getTimePtr (std::string paramName)

  Get a pointer to a TIME value from the input row.
- const TimeADT & getTimeRef (std::string paramName)

  Get a reference to a TIME value from the input row.
- const Timestamp * getTimestampPtr (std::string paramName)

  Get a pointer to a TIMESTAMP value from the input row.
- const Timestamp & getTimestampRef (std::string paramName)

  Get a reference to a TIMESTAMP value from the input row.
- const TimestampTz * getTimestampTzPtr (std::string paramName)

  Get a pointer to a TIMESTAMP WITH TIMEZONE value from the input row.
- const TimestampTz & getTimestampTzRef (std::string paramName)

  Get a reference to a TIMESTAMP WITH TIMEZONE value from the input row.
- const TimeTzADT * getTimeTzPtr (std::string paramName)

  Get a pointer to a TIME WITH TIMEZONE value from the input row.
- const TimeTzADT & getTimeTzRef (std::string paramName)

  Get a reference to a TIME WITH TIMEZONE value from the input row.
- VerticaType getType (std::string paramName)

  Return the type of the given parameter.
- const SizedColumnTypes & getTypeMetaData () const
- SizedColumnTypes & getTypeMetaData ()
- bool isEmpty () const

  Returns true if there are no parameters.
- void setAllocator (VTAllocator *allocator)

  Sets the allocator to be used to allocate variable size param values such as VString. Given allocator live longer than this ParamWriter.
- void setBool (std::string fieldName, vbool r)

  Adds a BOOLEAN value to the output row.
- void setDate (std::string fieldName, DateADT r)

  Adds a BOOLEAN value to the output row.
- void setFloat (std::string fieldName, vfloat r)

  Adds a FLOAT value to the output row.
- void setInt (std::string fieldName, vint r)

  Adds an INTEGER value to the output row.
- void setInterval (std::string fieldName, Interval r, int32 precision, int32 range)

  Adds an INTERVAL value to the output row.
- void setIntervalYM (std::string fieldName, IntervalYM r, int32 range)

  Adds an INTERVAL YEAR TO MONTH value to the output row.
- void setTime (std::string fieldName, TimeADT r, int32 precision)

Adds a TIME value to the output row.

- void setTimestamp (std::string fieldName, Timestamp r, int32 precision)

  Adds a TIMESTAMP value to the output row.
- void setTimestampTz (std::string fieldName, TimestampTz r, int32 precision)

  Adds a TIMESTAMP WITH TIMEZONE value to the output row.
- void setTimeTz (std::string fieldName, TimeTzADT r, int32 precision)

  Adds a TIME WITH TIMEZONE value to the output row.

Public Attributes

- std::map< std::string, size_t > paramNameToIndex

Protected Member Functions

- void addCol (char *arg, int colstride, const VerticaType &dt, const std::string fieldName="")
- void **addCol** (const char *arg, int colstride, const VerticaType &dt, const std::string fieldName="")
- void **validateStringColumn** (size_t idx, const VString &s, const VerticaType &t)

Protected Attributes

- std::vector< char * > **cols**
- std::vector< int > **colstrides**
- int **count**
- int **index**
- size_t **ncols**
- std::vector< VString > **svWrappers**
- SizedColumnTypes **typeMetaData**
- std::vector< VNumeric > **vnWrappers**

Friends

- class **VerticaBlockSerializer**

Detailed Description

Iterator interface for writing rows to a Vertica block.

This class provides the output rows that ScalarFunction.processBlock() writes to.

Member Function Documentation

void Vertica::VerticaBlock::addCol (char * *arg,* int *colstride,* const VerticaType & *dt,* const std::string *fieldName* = "") `[inline],[protected],[inherited]`

Add the location for reading a particular argument.

Parameters

arg	The base location to find data.
colstride	The stride between data instances.
dt	The type of input.
fieldname	the name of the field

void Vertica::ParamReader::addParameter (std::string *paramName,* const char * *arg,* const VerticaType & *dt*) `[inline],[inherited]`

Add a parameter to the block and stores it name and corresponding index in the paramNameToIndex map

const vbool* Vertica::ParamReader::getBoolPtr (std::string *paramName*) `[inline],[inherited]`

Get a pointer to a BOOLEAN value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

A pointer to the retrieved value cast as a BOOLEAN.

const vbool& Vertica::ParamReader::getBoolRef (std::string *paramName*) `[inline],[inherited]`

Get a reference to a BOOLEAN value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

a reference to the parameter value, cast as an BOOLEAN.

template<class T > const T* Vertica::VerticaBlock::getColPtr (size_t *idx*)
`[inline], [inherited]`

Returns

a pointer to the idx'th argument, cast appropriately.

Example:

```
const vint *a = arg_reader->getColPtr<vint>(0);
```

template<class T > const T& Vertica::VerticaBlock::getColRef (size_t *idx*)
`[inline], [inherited]`

Returns

a pointer to the idx'th argument, cast appropriately.

Example: const vint a = arg_reader->getColRef<vint>(0);

const DateADT* Vertica::ParamReader::getDatePtr (std::string *paramName*)
`[inline], [inherited]`

Get a pointer to a DATE value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

A pointer to the retrieved value cast as a DATE.

const DateADT& Vertica::ParamReader::getDateRef (std::string *paramName*)
`[inline], [inherited]`

Get a reference to a DATE value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

a reference to the parameter value, cast as an DATE.

const vfloat* Vertica::ParamReader::getFloatPtr (std::string *paramName*) [inline],[inherited]

Get a pointer to a FLOAT value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

A pointer to the retrieved value cast as a FLOAT.

const vfloat& Vertica::ParamReader::getFloatRef (std::string *paramName*) [inline],[inherited]

Get a reference to a FLOAT value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

A reference to the parameter value, cast as an FLOAT.

const Interval* Vertica::ParamReader::getIntervalPtr (std::string *paramName*) [inline],[inherited]

Get a pointer to an INTERVAL value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

A pointer to the retrieved value cast as an INTERVAL.

const Interval& Vertica::ParamReader::getIntervalRef (std::string *paramName*) [inline],[inherited]

Get a reference to an INTERVAL value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

a reference to the parameter value, cast as an INTERVAL.

const IntervalYM* Vertica::ParamReader::getIntervalYMPtr (std::string *paramName*) [inline],[inherited]

Get a pointer to a INTERVAL YEAR TO MONTH value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

A point to the retrieved value cast as a INTERVAL YEAR TO MONTH.

const IntervalYM& Vertica::ParamReader::getIntervalYMRef (std::string *paramName*) [inline],[inherited]

Get a reference to an INTERVAL YEAR TO MONTH value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

a reference to the parameter value, cast as an INTERVAL YEAR TO MONTH.

const vint* Vertica::ParamReader::getIntPtr (std::string *paramName*) [inline], [inherited]

Get a pointer to an INTEGER value from the input row.

Returns

a pointer to the idx'th argument, cast appropriately.

Parameters

paramName	The name of the parameter to retrieve

Example:

```
vint *a = arg_reader->getIntPtr("max");
```

const vint& Vertica::ParamReader::getIntRef (std::string *paramName*) [inline], [inherited]

Get a reference to an INTEGER value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

a reference to the parameter value, cast as an INTEGER.

Example:

```
vint a = arg_reader->getIntRef("max");
```

VString& Vertica::ParamWriter::getLongStringRef (std::string *fieldName*) [inline]

Allocates a new VString object to use as output. Sets it to be a 32mb LONG type by default.

Returns

A new VString object to hold output. This object automatically added to the output row.

size_t Vertica::VerticaBlock::getNumCols ( ) const `[inline], [inherited]`

Returns

the number of columns held by this block.

const VNumeric* Vertica::ParamReader::getNumericPtr (std::string *paramName*) `[inline], [inherited]`

Get a pointer to a VNumeric value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

A pointer to the retrieved value cast as a Numeric.

VNumeric& Vertica::ParamWriter::getNumericRef (std::string *fieldName*) `[inline]`

Allocate a new VNumeric object to use as output.

Returns

A new VNumeric object to hold output. This object automatically added to the output row.

int Vertica::VerticaBlock::getNumRows ( ) const `[inline], [inherited]`

Returns

the number of rows held by this block.

const VString* Vertica::ParamReader::getStringPtr (std::string *paramName*) `[inline], [inherited]`

Get a pointer to a VString value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

A pointer to the retrieved value cast as a VString.

VString& Vertica::ParamWriter::getStringRef (std::string *fieldName*) [inline]

Allocates a new VString object to use as output.

Returns

A new VString object to hold output. This object automatically added to the output row.

const TimeADT* Vertica::ParamReader::getTimePtr (std::string *paramName*) [inline], [inherited]

Get a pointer to a TIME value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

A pointer to the retrieved value cast as a TIME.

const TimeADT& Vertica::ParamReader::getTimeRef (std::string *paramName*) [inline], [inherited]

Get a reference to a TIME value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

a reference to the parameter value, cast as a TIME.

const Timestamp* Vertica::ParamReader::getTimestampPtr (std::string *paramName*) [inline], [inherited]

Get a pointer to a TIMESTAMP value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

A pointer to the retrieved value cast as a TIMESTAMP.

const Timestamp& Vertica::ParamReader::getTimestampRef (std::string *paramName*) `[inline],[inherited]`

Get a reference to a TIMESTAMP value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

a reference to the parameter value, cast as a TIMESTAMP.

const TimestampTz* Vertica::ParamReader::getTimestampTzPtr (std::string *paramName*) `[inline],[inherited]`

Get a pointer to a TIMESTAMP WITH TIMEZONE value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

A pointer to the retrieved value cast as a TIMESTAMP WITH TIMEZONE .

const TimestampTz& Vertica::ParamReader::getTimestampTzRef (std::string *paramName*) `[inline],[inherited]`

Get a reference to a TIMESTAMP WITH TIMEZONE value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

a reference to the parameter value, cast as a TIMESTAMP WITH TIMEZONE.

const TimeTzADT* Vertica::ParamReader::getTimeTzPtr (std::string *paramName*) `[inline],[inherited]`

Get a pointer to a TIME WITH TIMEZONE value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

A pointer to the retrieved value cast as a TIME WITH TIMEZONE.

const TimeTzADT& Vertica::ParamReader::getTimeTzRef (std::string *paramName*) `[inline],[inherited]`

Get a reference to a TIME WITH TIMEZONE value from the input row.

Parameters

paramName	The name of the parameter to retrieve

Returns

a reference to the parameter value, cast as a TIME WITH TIMEZONE.

const SizedColumnTypes& Vertica::VerticaBlock::getTypeMetaData ( ) const `[inline],[inherited]`

Returns

information about the types and numbers of arguments

SizedColumnTypes& Vertica::VerticaBlock::getTypeMetaData ( ) `[inline], [inherited]`

Returns

information about the types and numbers of arguments

void Vertica::ParamWriter::setAllocator (VTAllocator * *allocator*) [inline]

Sets the allocator to be used to allocate variable size param values such as VString. Given allocator live longer than this ParamWriter.

Parameters

allocator	Used to allocate param values.

void Vertica::ParamWriter::setBool (std::string *fieldName*, vbool *r*) [inline]

Adds a BOOLEAN value to the output row.

Parameters

r	The BOOLEAN value to insert into the output row.

void Vertica::ParamWriter::setDate (std::string *fieldName*, DateADT *r*) [inline]

Adds a BOOLEAN value to the output row.

Parameters

r	The BOOLEAN value to insert into the output row.

void Vertica::ParamWriter::setFloat (std::string *fieldName*, vfloat *r*) [inline]

Adds a FLOAT value to the output row.

Parameters

r	The FLOAT value to insert into the output row.

void Vertica::ParamWriter::setInt (std::string *fieldName*, vint *r*) [inline]

Adds an INTEGER value to the output row.

Setter methods

Parameters

r	The INTEGER value to insert into the output row.

void Vertica::ParamWriter::setInterval (std::string *fieldName,* Interval *r,* int32 *precision,* int32 *range*) `[inline]`

Adds an INTERVAL value to the output row.

Parameters

r	The INTERVAL value to insert into the output row.

void Vertica::ParamWriter::setIntervalYM (std::string *fieldName,* IntervalYM *r,* int32 *range*) `[inline]`

Adds an INTERVAL YEAR TO MONTH value to the output row.

Parameters

r	The INTERVAL YEAR TO MONTH value to insert into the output row.

void Vertica::ParamWriter::setTime (std::string *fieldName,* TimeADT *r,* int32 *precision*) `[inline]`

Adds a TIME value to the output row.

Parameters

r	The TIME value to insert into the output row.

void Vertica::ParamWriter::setTimestamp (std::string *fieldName,* Timestamp *r,* int32 *precision*) `[inline]`

Adds a TIMESTAMP value to the output row.

Parameters

r	The TIMESTAMP value to insert into the output row.

void Vertica::ParamWriter::setTimestampTz (std::string *fieldName,* TimestampTz *r,* int32 *precision*) `[inline]`

Adds a TIMESTAMP WITH TIMEZONE value to the output row.

Parameters

r	The TIMESTAMP WITH TIMEZONE value to insert into the output row.

void Vertica::ParamWriter::setTimeTz (std::string *fieldName*, TimeTzADT *r*, int32 *precision*) `[inline]`

Adds a TIME WITH TIMEZONE value to the output row.

Parameters

r	The TIME WITH TIMEZONE value to insert into the output row.

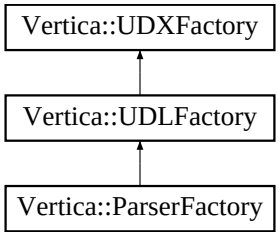
Member Data Documentation

std::map<std::string, size_t> Vertica::ParamReader::paramNameToIndex `[inherited]`

Bookkeepinp to make a parameter and its position in the block

Vertica::ParserFactory Class Reference

Inheritance diagram for Vertica::ParserFactory:



Public Types

- enum UDXType {
  FUNCTION, **TRANSFORM**, **ANALYTIC**, **MULTI_TRANSFORM**, **AGGREGATE**, **LOAD_SOURCE**, **LOAD_FILTER**, **LOAD_PARSER**, **TYPE** }

Public Member Functions

- virtual void getParameterType (ServerInterface &srvInterface, SizedColumnTypes &parameterTypes)
- virtual void getParserReturnType (ServerInterface &srvInterface, PerColumnParamReader &perColumnParamReader, PlanContext &planCtxt, const SizedColumnTypes &argTypes, SizedColumnTypes &returnType)
- virtual void getPerInstanceResources (ServerInterface &srvInterface, VResources &res)
- void getPrototype (ServerInterface &srvInterface, ColumnTypes &argTypes, ColumnTypes &returnType)
- virtual void getReturnType (ServerInterface &srvInterface, const SizedColumnTypes &argTypes, SizedColumnTypes &returnType)
- UDXFactory::UDXType getUDXFactoryType ()
- virtual void plan (ServerInterface &srvInterface, PerColumnParamReader &perColumnParamReader, PlanContext &planCtxt)
- virtual UDParser * prepare (ServerInterface &srvInterface, PerColumnParamReader &perColumnParamReader, PlanContext &planCtxt, const SizedColumnTypes &returnType)=0

Protected Attributes

- Oid **libOid**
- std::string **sqlName**

Detailed Description

Construct a single Parser.

Note that ParserFactories are singletons. Subclasses should be stateless, with no fields containing data, just methods. plan() and prepare() methods must never modify any global variables or state; they may only modify the variables that they are given as arguments. (If global state must be modified, use SourceIterator.)

Factories should be registered using the RegisterFactory() macro, defined in Vertica.h.

Member Enumeration Documentation

enum Vertica::UDXFactory::UDXType `[inherited]`

The type of UDX instance this factory produces

Member Function Documentation

virtual void Vertica::ParserFactory::getParameterType (ServerInterface & *srvInterface,* SizedColumnTypes & *parameterTypes*) `[inline],[virtual]`

Inherited from the parent "UDXFactory" class in VerticaUDx.h

Reimplemented from Vertica::UDXFactory.

virtual void Vertica::ParserFactory::getParserReturnType (ServerInterface & *srvInterface,* PerColumnParamReader & *perColumnParamReader,* PlanContext & *planCtxt,* const SizedColumnTypes & *argTypes,* SizedColumnTypes & *returnType*) `[inline],[virtual]`

Function to tell Vertica what the return types (and length/precision if necessary) of this UDX are. Called, possibly multiple times, on each node executing the query.

The default provided implementation configures Vertica to use the same output column types as the destination table. This requires that the UDParser validate the expected output column types and emit appropriate tuples. Note that the default provided implementation of this function should be sufficient for most Parsers, so this method should not be overridden by most Parser implementations. If a COPY statement has a return type that doesn't match the destination table, Vertica will emit an appropriate error. Users can use COPY expressions to perform typecasting and conversion if necessary.

For CHAR/VARCHAR types, specify the max length,

For NUMERIC types, specify the precision and scale.

For Time/Timestamp types (with or without time zone), specify the precision, -1 means unspecified/don't care

For IntervalYM/IntervalDS types, specify the precision and range

For all other types, no length/precision specification needed

Parameters

srvInterface	Interface to server operations and functionality, including (not-per-column) parameter lookup
perColumnParamReader	Per-column parameters passed into the query
planCtxt	Context for storing and retrieving arbitrary data, for use just by this instance of this query. The same context is shared with plan().
argTypes	Provides the data types of arguments that this UDT was called with. This may be used to modify the return types accordingly.
returnType	User code must fill in the names and data types returned by the UDT.

virtual void Vertica::UDXFactory::getPerInstanceResources (ServerInterface & ***srvInterface,*** **VResources &** ***res*** **)** `[inline], [virtual], [inherited]`

Set the resource required for each instance of the UDX Object subclass

Parameters

srvInterface	a ServerInterface object used to communicate with Vertica
res	a VResources object used to tell Vertica what resources are needed by the UDX

void Vertica::UDLFactory::getPrototype (ServerInterface & ***srvInterface,*** **ColumnTypes &** ***argTypes,*** **ColumnTypes &** ***returnType*** **)** `[inline], [virtual], [inherited]`

Provides the argument and return types of the UDL. UDL's take no input tuples; as such, their prototype is empty.

Implements Vertica::UDXFactory.

virtual void Vertica::UDLFactory::getReturnType (ServerInterface & ***srvInterface,*** **const SizedColumnTypes &** ***argTypes,*** **SizedColumnTypes &** ***returnType*** **)** `[inline], [virtual], [inherited]`

Not used in this form

Implements Vertica::UDXFactory.

UDXFactory::UDXType Vertica::ParserFactory::getUDXFactoryType ( ) `[inline], [virtual]`

Returns

the type of UDX Object instance this factory returns.

Note

User subclasses should use the appropriate subclass of UDXFactory and not override this method on their own.

Implements Vertica::UDXFactory.

virtual void Vertica::ParserFactory::plan (ServerInterface & *srvInterface,* PerColumnParamReader & *perColumnParamReader,* PlanContext & *planCtxt*) `[inline],[virtual]`

Execute any planning logic required at query plan time. This method is run once per query, during query initialization. Its job is to perform parameter validation, and to modify the set of nodes that the COPY statement will run on (through srvInterface).

plan() runs exactly once per query, on the initiator node. If it throws an exception, the query will not proceed; it will be aborted prior to distributing the query to the other nodes and running prepare().

Parameters

srvInterface	Interface to server operations and functionality, including (not-per-column) parameter lookup
perColumn-Param-Reader	Per-column parameters passed into the query
planCtxt	Context for storing and retrieving arbitrary data, for use just by this instance of this query. The same context is shared with plan().

virtual UDParser* Vertica::ParserFactory::prepare (ServerInterface & *srvInterface,* PerColumnParamReader & *perColumnParamReader,* PlanContext & *planCtxt,* const SizedColumnTypes & *returnType*) `[pure virtual]`

Instantiate a UDParser instance. This function will be called on each node, prior to the Load operator starting to execute.

'planData' contains the same data that was placed there by the plan() static method.

Parameters

srvInterface	Interface to server operations and functionality, including (not-per-column) parameter lookup
perColumn-Param-Reader	Per-column parameters passed into the query
planCtxt	Context for storing and retrieving arbitrary data, for use just by this instance of this query. The same context is shared with plan().
returnType	The data types of the columns that this Parser must produce

Returns

The UDParser instance to be used by this query

Vertica::PartitionOrderColumnInfo Struct Reference

Represents the partition by and order by column information for each phase in a multi-phase transform function.

Public Attributes

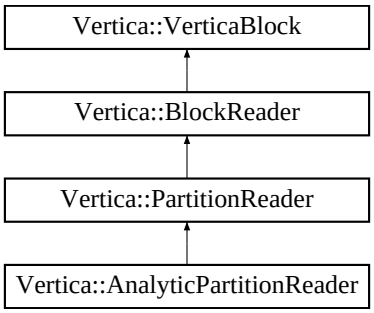
- int **last_order_col**
- int **last_partition_col**

Detailed Description

Represents the partition by and order by column information for each phase in a multi-phase transform function.

Vertica::PartitionReader Class Reference

Inheritance diagram for Vertica::PartitionReader:



Public Member Functions

- **PartitionReader** (size_t narg, EE::UserDefinedProcess ∗udx_object)
- const vbool ∗ getBoolPtr (size_t idx)

  Get a pointer to a BOOLEAN value from the input row.
- const vbool & getBoolRef (size_t idx)

  Get a reference to a BOOLEAN value from the input row.
- template<class T >
  const T ∗ getColPtr (size_t idx)
- template<class T >
  T ∗ **getColPtrForWrite** (size_t idx)
- template<class T >
  const T & getColRef (size_t idx)

- template<class T >
  T & **getColRefForWrite** (size_t idx)
- const DateADT * getDatePtr (size_t idx)

  Get a pointer to a DATE value from the input row.
- const DateADT & getDateRef (size_t idx)

  Get a reference to a DATE value from the input row.
- const vfloat * getFloatPtr (size_t idx)

  Get a pointer to a FLOAT value from the input row.
- const vfloat & getFloatRef (size_t idx)

  Get a reference to a FLOAT value from the input row.
- const Interval * getIntervalPtr (size_t idx)

  Get a pointer to an INTERVAL value from the input row.
- const Interval & getIntervalRef (size_t idx)

  Get a reference to an INTERVAL value from the input row.
- const IntervalYM * getIntervalYMPtr (size_t idx)

  Get a pointer to a INTERVAL YEAR TO MONTH value from the input row.
- const IntervalYM & getIntervalYMRef (size_t idx)

  Get a reference to an INTERVAL YEAR TO MONTH value from the input row.
- const vint * getIntPtr (size_t idx)

  Get a pointer to an INTEGER value from the input row.
- const vint & getIntRef (size_t idx)

  Get a reference to an INTEGER value from the input row.
- size_t getNumCols () const
- const VNumeric * getNumericPtr (size_t idx)

  Get a pointer to a VNumeric value from the input row.
- const VNumeric & getNumericRef (size_t idx)

  Get a reference to a VNumeric value from the input row.
- int getNumRows () const
- const VString * getStringPtr (size_t idx)

  Get a pointer to a VString value from the input row.
- const VString & getStringRef (size_t idx)

  Get a reference to an VString value from the input row.
- const TimeADT * getTimePtr (size_t idx)

  Get a pointer to a TIME value from the input row.
- const TimeADT & getTimeRef (size_t idx)

  Get a reference to a TIME value from the input row.
- const Timestamp * getTimestampPtr (size_t idx)

  Get a pointer to a TIMESTAMP value from the input row.
- const Timestamp & getTimestampRef (size_t idx)

  Get a reference to a TIMESTAMP value from the input row.

- const TimestampTz * getTimestampTzPtr (size_t idx)

  Get a pointer to a TIMESTAMP WITH TIMEZONE value from the input row.
- const TimestampTz & getTimestampTzRef (size_t idx)

  Get a reference to a TIMESTAMP WITH TIMEZONE value from the input row.
- const TimeTzADT * getTimeTzPtr (size_t idx)

  Get a pointer to a TIME WITH TIMEZONE value from the input row.
- const TimeTzADT & getTimeTzRef (size_t idx)

  Get a reference to a TIME WITH TIMEZONE value from the input row.
- const SizedColumnTypes & getTypeMetaData () const
- SizedColumnTypes & getTypeMetaData ()
- bool isNull (int col)

  Check if the idx'th argument is null.
- bool **next** ()
- virtual bool readNextBlock ()

Protected Member Functions

- void addCol (char *arg, int colstride, const VerticaType &dt, const std::string fieldName="")
- void **addCol** (const char *arg, int colstride, const VerticaType &dt, const std::string fieldName="")
- virtual void **setupColsAndStrides** ()
- void **validateStringColumn** (size_t idx, const VString &s, const VerticaType &t)

Protected Attributes

- union {
  EE::UserDefinedAnalytic * **udan**
  EE::UserDefinedTransform * **udt**
  EE::UserDefinedProcess * **udx_object**
  };

- std::vector< char * > **cols**
- std::vector< int > **colstrides**
- int **count**
- EE::DataHolder * **dh**
- int **index**
- size_t **ncols**
- vpos **pstart**
- std::vector< VString > **svWrappers**
- SizedColumnTypes **typeMetaData**
- std::vector< VNumeric > **vnWrappers**

Friends

- class **EE::UserDefinedAnalytic**
- class **EE::UserDefinedProcess**
- class **EE::UserDefinedTransform**
- class **FullPartition**
- class **VerticaBlockSerializer**

Detailed Description

PartitionReader provides an iterator-based read interface over all input data in a single partition. Automatically fetches data a block-at-a-time, as needed.

Member Function Documentation

void Vertica::VerticaBlock::addCol (char * *arg,* int *colstride,* const VerticaType & *dt,* const std::string *fieldName* = "") `[inline], [protected], [inherited]`

Add the location for reading a particular argument.

Parameters

arg	The base location to find data.
colstride	The stride between data instances.
dt	The type of input.
fieldname	the name of the field

const vbool* Vertica::BlockReader::getBoolPtr (size_t *idx*) `[inline], [inherited]`

Get a pointer to a BOOLEAN value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as a BOOLEAN.

const vbool& Vertica::BlockReader::getBoolRef (size_t *idx*) `[inline], [inherited]`

Get a reference to a BOOLEAN value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as an BOOLEAN.

template<class T > const T* Vertica::VerticaBlock::getColPtr (size_t *idx*) `[inline], [inherited]`

Returns

a pointer to the idx'th argument, cast appropriately.

Example:

```
const vint *a = arg_reader->getColPtr<vint>(0);
```

template<class T > const T& Vertica::VerticaBlock::getColRef (size_t *idx*) `[inline], [inherited]`

Returns

a pointer to the idx'th argument, cast appropriately.

Example: const vint a = arg_reader->getColRef<vint>(0);

const DateADT* Vertica::BlockReader::getDatePtr (size_t *idx*) `[inline], [inherited]`

Get a pointer to a DATE value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as a DATE.

const DateADT& Vertica::BlockReader::getDateRef (size_t *idx*) `[inline], [inherited]`

Get a reference to a DATE value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as an DATE.

const vfloat* Vertica::BlockReader::getFloatPtr (size_t *idx*) `[inline], [inherited]`

Get a pointer to a FLOAT value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as a FLOAT.

const vfloat& Vertica::BlockReader::getFloatRef (size_t *idx*) `[inline], [inherited]`

Get a reference to a FLOAT value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

A reference to the idx'th argument, cast as an FLOAT.

const Interval* Vertica::BlockReader::getIntervalPtr (size_t *idx*) `[inline], [inherited]`

Get a pointer to an INTERVAL value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as an INTERVAL.

const Interval& Vertica::BlockReader::getIntervalRef (size_t *idx*) `[inline], [inherited]`

Get a reference to an INTERVAL value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as an INTERVAL.

const IntervalYM* Vertica::BlockReader::getIntervalYMPtr (size_t *idx*) `[inline], [inherited]`

Get a pointer to a INTERVAL YEAR TO MONTH value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A point to the retrieved value cast as a INTERVAL YEAR TO MONTH.

const IntervalYM& Vertica::BlockReader::getIntervalYMRef (size_t *idx*) `[inline], [inherited]`

Get a reference to an INTERVAL YEAR TO MONTH value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as an INTERVAL YEAR TO MONTH.

const vint* Vertica::BlockReader::getIntPtr (size_t *idx*) `[inline], [inherited]`

Get a pointer to an INTEGER value from the input row.

Returns

a pointer to the idx'th argument, cast appropriately.

Parameters

idx	The column number to retrieve from the input row.

Example:

```
const vint *a = arg_reader->getIntPtr(0);
```

const vint& Vertica::BlockReader::getIntRef (size_t *idx*) `[inline], [inherited]`

Get a reference to an INTEGER value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as an INTEGER.

Example:

```
const vint a = arg_reader->getIntRef(0);
```

size_t Vertica::VerticaBlock::getNumCols ( ) const `[inline], [inherited]`

Returns

the number of columns held by this block.

const VNumeric* Vertica::BlockReader::getNumericPtr (size_t *idx*) `[inline], [inherited]`

Get a pointer to a VNumeric value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

A pointer to the retrieved value cast as a Numeric.

const VNumeric& Vertica::BlockReader::getNumericRef (size_t *idx*) `[inline], [inherited]`

Get a reference to a VNumeric value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as an VNumeric.

int Vertica::VerticaBlock::getNumRows ( ) const `[inline], [inherited]`

Returns

the number of rows held by this block.

const VString* Vertica::BlockReader::getStringPtr (size_t *idx*) `[inline], [inherited]`

Get a pointer to a VString value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

A pointer to the retrieved value cast as a VString.

const VString& Vertica::BlockReader::getStringRef (size_t *idx*) `[inline], [inherited]`

Get a reference to an VString value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as an VString.

const TimeADT* Vertica::BlockReader::getTimePtr (size_t *idx*) `[inline], [inherited]`

Get a pointer to a TIME value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as a TIME.

const TimeADT& Vertica::BlockReader::getTimeRef (size_t *idx*) `[inline], [inherited]`

Get a reference to a TIME value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as a TIME.

const Timestamp* Vertica::BlockReader::getTimestampPtr (size_t *idx*) `[inline], [inherited]`

Get a pointer to a TIMESTAMP value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as a TIMESTAMP.

const Timestamp& Vertica::BlockReader::getTimestampRef (size_t *idx*) `[inline], [inherited]`

Get a reference to a TIMESTAMP value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as a TIMESTAMP.

const TimestampTz* Vertica::BlockReader::getTimestampTzPtr (size_t *idx*) `[inline], [inherited]`

Get a pointer to a TIMESTAMP WITH TIMEZONE value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as a TIMESTAMP WITH TIMEZONE .

const TimestampTz& Vertica::BlockReader::getTimestampTzRef (size_t *idx*) `[inline],[inherited]`

Get a reference to a TIMESTAMP WITH TIMEZONE value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as a TIMESTAMP WITH TIMEZONE.

const TimeTzADT* Vertica::BlockReader::getTimeTzPtr (size_t *idx*) `[inline], [inherited]`

Get a pointer to a TIME WITH TIMEZONE value from the input row.

Parameters

idx	The column number in the input row to retrieve.

Returns

A pointer to the retrieved value cast as a TIME WITH TIMEZONE.

const TimeTzADT& Vertica::BlockReader::getTimeTzRef (size_t *idx*) `[inline], [inherited]`

Get a reference to a TIME WITH TIMEZONE value from the input row.

Parameters

idx	The column number to retrieve from the input row.

Returns

a reference to the idx'th argument, cast as a TIME WITH TIMEZONE.

const SizedColumnTypes& Vertica::VerticaBlock::getTypeMetaData ( ) const `[inline],[inherited]`

Returns

information about the types and numbers of arguments

SizedColumnTypes& Vertica::VerticaBlock::getTypeMetaData ( ) [inline], [inherited]

Returns

information about the types and numbers of arguments

bool Vertica::BlockReader::isNull (int *col*) [inline], [inherited]

Check if the idx'th argument is null.

Parameters

col	The column number in the row to check for null

Returns

true is the col value is null false otherwise

virtual bool Vertica::PartitionReader::readNextBlock ( ) [virtual]

Reads in the next block of data and positions cursor at the beginning.

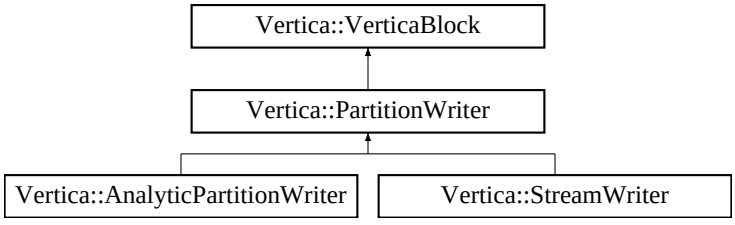
Returns

false if there's no more input data

Reimplemented in Vertica::AnalyticPartitionReader.

Vertica::PartitionWriter Class Reference

Inheritance diagram for Vertica::PartitionWriter:



Public Member Functions

- **PartitionWriter** (size_t narg, EE::UserDefinedProcess *udx_object)
- void copyFromInput (size_t dstIdx, PartitionReader &input_reader, size_t srcIdx)
- template<class T >
  const T * getColPtr (size_t idx)
- template<class T >
  T * **getColPtrForWrite** (size_t idx)
- template<class T >
  const T & getColRef (size_t idx)
- template<class T >
  T & **getColRefForWrite** (size_t idx)
- size_t getNumCols () const
- VNumeric & **getNumericRef** (size_t idx)
- int getNumRows () const
- VString & **getStringRef** (size_t idx)
- VString & **getStringRefNoClear** (size_t idx)
- const SizedColumnTypes & getTypeMetaData () const
- SizedColumnTypes & getTypeMetaData ()
- void * **getVoidPtr** (size_t idx)
- virtual bool getWriteableBlock ()
- bool **next** ()
- void **setBool** (size_t idx, vbool r)
- void **setDate** (size_t idx, DateADT r)
- void **setFloat** (size_t idx, vfloat r)
- void setInt (size_t idx, vint r)
- void **setInterval** (size_t idx, Interval r)
- void setNull (size_t idx)

  Set the idx'th argument to null.
- void **setTime** (size_t idx, TimeADT r)
- void **setTimestamp** (size_t idx, Timestamp r)
- void **setTimestampTz** (size_t idx, TimestampTz r)
- void **setTimeTz** (size_t idx, TimeTzADT r)
- void **validateColumn** (size_t idx)

Protected Member Functions

- void addCol (char *arg, int colstride, const VerticaType &dt, const std::string fieldName="")
- void **addCol** (const char *arg, int colstride, const VerticaType &dt, const std::string fieldName="")
- void **validateStringColumn** (size_t idx, const VString &s, const VerticaType &t)

Protected Attributes

- union {
  EE::UserDefinedAnalytic * **udan**
  EE::UserDefinedTransform * **udt**
  EE::UserDefinedProcess * **udx_object**
  };

- std::vector< char * > **cols**
- std::vector< int > **colstrides**
- int **count**
- int **index**
- size_t **ncols**
- int **pstart**
- std::vector< VString > **svWrappers**
- SizedColumnTypes **typeMetaData**
- std::vector< VNumeric > **vnWrappers**

Friends

- class **EE::Loader::UserDefinedLoad**
- class **EE::UserDefinedAnalytic**
- class **EE::UserDefinedProcess**
- class **EE::UserDefinedTransform**

Detailed Description

PartitionWriter provides an iterator-based write interface over output data for a single partition. Automatically makes space a block-at-a-time, as needed.

Member Function Documentation

void Vertica::VerticaBlock::addCol (char * *arg,* int *colstride,* const VerticaType & *dt,* const std::string *fieldName* = "") `[inline],[protected],[inherited]`

Add the location for reading a particular argument.

Parameters

arg	The base location to find data.
colstride	The stride between data instances.
dt	The type of input.
fieldname	the name of the field

void Vertica::PartitionWriter::copyFromInput (size_t *dstIdx,* PartitionReader & *input_reader,* size_t *srcIdx*) `[inline]`

Copies a column from the input reader to the output writer. The data types and sizes of the source and destination columns must match exactly.

Parameters

dstIdx	The destination column index (in the output writer)
input_reader	The input reader from which to copy a column
srcIdx	The source column index (in the input reader)

template<class T > const T* Vertica::VerticaBlock::getColPtr (size_t *idx*) `[inline], [inherited]`

Returns

a pointer to the idx'th argument, cast appropriately.

Example:

```
const vint *a = arg_reader->getColPtr<vint>(0);
```

template<class T > const T& Vertica::VerticaBlock::getColRef (size_t *idx*) `[inline], [inherited]`

Returns

a pointer to the idx'th argument, cast appropriately.

Example: const vint a = arg_reader->getColRef<vint>(0);

size_t Vertica::VerticaBlock::getNumCols ( ) const `[inline], [inherited]`

Returns

the number of columns held by this block.

int Vertica::VerticaBlock::getNumRows ( ) const `[inline], [inherited]`

Returns

the number of rows held by this block.

const SizedColumnTypes& Vertica::VerticaBlock::getTypeMetaData ( ) const `[inline], [inherited]`

Returns

information about the types and numbers of arguments

SizedColumnTypes& Vertica::VerticaBlock::getTypeMetaData ( ) `[inline], [inherited]`

Returns

information about the types and numbers of arguments

virtual bool Vertica::PartitionWriter::getWriteableBlock ( ) `[virtual]`

Gets a writeable block of data and positions cursor at the beginning.

Reimplemented in Vertica::AnalyticPartitionWriter, and Vertica::StreamWriter.

void Vertica::PartitionWriter::setInt (size_t *idx*, vint *r*) `[inline]`

Setter methods

void Vertica::PartitionWriter::setNull (size_t *idx*) `[inline]`

Set the idx'th argument to null.

Parameters

idx	The column number in the row to set to null

Vertica::PerColumnParamReader Class Reference

: A wrapper around a map from column to ParamReader.

Public Member Functions

- bool containsColumn (std::string columnName) const

  Returns true if a ParamReader exists for the given column.
- std::vector< std::string > getColumnNames () const

  Gets the names of all columns with column specific arguments.

- ParamReader & getColumnParamReader (const std::string &column)

  Gets the parameters of the given column.

Detailed Description

: A wrapper around a map from column to ParamReader.

Member Function Documentation

std::vector<std::string> Vertica::PerColumnParamReader::getColumnNames ( ) const `[inline]`

Gets the names of all columns with column specific arguments.

Returns

a vector of column names

ParamReader& Vertica::PerColumnParamReader::getColumnParamReader (const std::string & *column*) `[inline]`

Gets the parameters of the given column.

Parameters

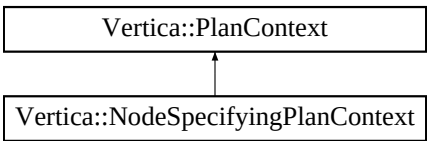
the	name of the column of interest

Returns

the parameters of the given column

Vertica::PlanContext Class Reference

Inheritance diagram for Vertica::PlanContext:



Public Member Functions

- **PlanContext** (ParamWriter &writer, std::vector< std::string > clusterNodes)
- **PlanContext** (ParamWriter &writer)
- const std::vector< std::string > & getClusterNodes ()
- ParamReader & getReader ()
- ParamWriter & getWriter ()

Detailed Description

Interface that allows storage of query-plan state, when different parts of query planning take place on different computers. For example, if some work is done on the query initiator node and some is done on each node executing the query.

Member Function Documentation

const std::vector<std::string>& Vertica::PlanContext::getClusterNodes ( ) [inline]

Get a list of all of the nodes in the current cluster, by node name

ParamReader& Vertica::PlanContext::getReader ( ) [inline]

Get a read-only instance of the current context

ParamWriter& Vertica::PlanContext::getWriter ( ) [inline]

Get the current context for writing

Vertica::RejectedRecord Struct Reference

Public Member Functions

- **RejectedRecord** (const std::string &reason, const char *data=NULL, size_t length=0, const std::string &terminator="\n")

Public Attributes

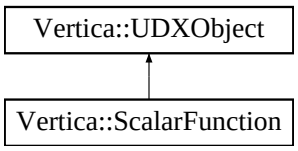
- const char * **data**
- size_t **length**
- std::string **reason**
- std::string **terminator**

Detailed Description

Information about a rejected record.

Vertica::ScalarFunction Class Reference

Inheritance diagram for Vertica::ScalarFunction:



Public Member Functions

- virtual void destroy (ServerInterface &srvInterface, const SizedColumnTypes &arg-Types)
- virtual void processBlock (ServerInterface &srvInterface, BlockReader &arg_-reader, BlockWriter &res_writer)=0
- virtual void setup (ServerInterface &srvInterface, const SizedColumnTypes &arg-Types)

Protected Types

- enum **InterfaceType** { **FunctionT**, **IndexListFunctionT** }

Protected Member Functions

- virtual InterfaceType **getInterfaceType** ()

Friends

- class **UdfSupport**

Detailed Description

Interface for User Defined Scalar Function, the actual code to process a block of data.

Member Function Documentation

virtual void Vertica::UDXObject::destroy (ServerInterface & *srvInterface,* const SizedColumnTypes & *argTypes*) `[inline], [virtual], [inherited]`

Perform per instance destruction. This function may throw errors

virtual void Vertica::ScalarFunction::processBlock (ServerInterface & *srvInterface,* BlockReader & *arg_reader,* BlockWriter & *res_writer*) `[pure virtual]`

Invoke a user defined function on a set of rows. As the name suggests, a batch of rows are passed in for every invocation to amortize performance.

Parameters

srvInterface	a ServerInterface object used to communicate with Vertica
arg_reader	input rows
res_writer	output location

Note

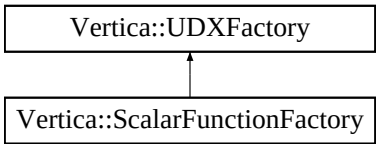
- This methods may be invoked by different threads at different times, and by a different thread than the constructor.
- The order in which the function sees rows is not guaranteed.
- C++ exceptions may NOT be thrown out of this method. Use the vertica specific vt_throw_exception() function or vt_report_error() macro instead

virtual void Vertica::UDXObject::setup (ServerInterface & *srvInterface,* const SizedColumnTypes & *argTypes*) `[inline], [virtual], [inherited]`

Perform per instance initialization. This function may throw errors.

Vertica::ScalarFunctionFactory Class Reference

Inheritance diagram for Vertica::ScalarFunctionFactory:



Public Types

- enum UDXType {
  FUNCTION, **TRANSFORM**, **ANALYTIC**, **MULTI_TRANSFORM**,
  AGGREGATE, **LOAD_SOURCE**, **LOAD_FILTER**, **LOAD_PARSER**,
  TYPE }

Public Member Functions

- virtual ScalarFunction * createScalarFunction (ServerInterface &srvInterface)=0
- virtual void getParameterType (ServerInterface &srvInterface, SizedColumnTypes &parameterTypes)
- virtual void getPerInstanceResources (ServerInterface &srvInterface, VResources &res)
- virtual void getPrototype (ServerInterface &srvInterface, ColumnTypes &argTypes, ColumnTypes &returnType)=0
- virtual void getReturnType (ServerInterface &srvInterface, const SizedColumnTypes &argTypes, SizedColumnTypes &returnType)

Public Attributes

- strictness **strict**
- volatility vol

Protected Member Functions

- virtual UDXType getUDXFactoryType ()

Protected Attributes

- Oid **libOid**
- std::string **sqlName**

Detailed Description

MetaData interface for Vertica User Defined Scalar Functions.

A ScalarFunctionFactory is responsible for providing type and type modifier information.

Member Enumeration Documentation

enum Vertica::UDXFactory::UDXType `[inherited]`

The type of UDX instance this factory produces

Member Function Documentation

virtual ScalarFunction* Vertica::ScalarFunctionFactory::createScalarFunction (ServerInterface & *srvInterface*) `[pure virtual]`

Returns

an ScalarFunction object which implements the UDx API described by this meta-data.

Parameters

srvInterface	a ServerInterface object used to communicate with Vertica

Note

More than one object may be instantiated per query.

virtual void Vertica::UDXFactory::getParameterType (ServerInterface & *srvInterface*, SizedColumnTypes & *parameterTypes*) `[inline], [virtual], [inherited]`

Function to tell Vertica the name and types of parameters that this function uses. Vertica will use this to warn function callers that certain parameters they provide are not affecting anything, or that certain parameters that are not being set are reverting to default values.

Reimplemented in Vertica::ParserFactory.

virtual void Vertica::UDXFactory::getPerInstanceResources (ServerInterface & *srvInterface*, VResources & *res*) `[inline], [virtual], [inherited]`

Set the resource required for each instance of the UDX Object subclass

Parameters

srvInterface	a ServerInterface object used to communicate with Vertica
res	a VResources object used to tell Vertica what resources are needed by the UDX

virtual void Vertica::UDXFactory::getPrototype (ServerInterface & *srvInterface*, ColumnTypes & *argTypes*, ColumnTypes & *returnType*) `[pure virtual], [inherited]`

Provides the argument and return types of the UDX

Implemented in Vertica::UDLFactory, and Vertica::MultiPhaseTransformFunctionFactory.

virtual void Vertica::ScalarFunctionFactory::getReturnType (ServerInterface & *srvInterface*, const SizedColumnTypes & *argTypes*, SizedColumnTypes & *returnType*) `[inline], [virtual]`

For scalar functions, this function needs to be overridden only if the return type needs length/precision specification.

Parameters

srvInterface	a ServerInterface object used to communicate with Vertica
argTypes	The data type of the return value defined by processBlock()
returnType	The size of the data returned by processBlock()

Implements Vertica::UDXFactory.

virtual UDXType Vertica::ScalarFunctionFactory::getUDXFactoryType ( ) `[inline], [protected], [virtual]`

Returns

the object type internally used by Vertica

Implements Vertica::UDXFactory.

Member Data Documentation

volatility Vertica::ScalarFunctionFactory::vol

Strictness and Volatility settings that the UDSF programmer can set Defaults are VOLATILE and CALLED_ON_NULL_INPUT

Vertica::ServerInterface Class Reference

Public Types

- typedef void(∗ **LoggingFunc**)(ServerInterface ∗, const char ∗fmt, va_list ap)

Public Member Functions

- ServerInterface (VTAllocator *allocator, LoggingFunc func, const std::string &sqlName, const ParamReader &paramReader)
- **ServerInterface** (VTAllocator *allocator, LoggingFunc func, const std::string &sqlName)
- formatWithName **append** (" - ")
- formatWithName **append** (format)
- void std::string **formatWithName** (sqlName)
- const std::string & getCurrentNodeName () const
- const std::string & getLocale () const
- ParamReader getParamReader ()
- void log (const char *format,...) __attribute__((format(printf
- void setParamReader (const ParamReader &paramReader)
- **va_end** (ap)
- **va_start** (ap, format)
- **vlog** (format, ap)
- void vlog (const char *format, va_list ap)

Public Attributes

- VTAllocator * allocator
- va_list **ap**
- **format** = formatWithName.c_str()

Protected Attributes

- std::string locale
- std::string nodeName
- ParamReader paramReader
- std::string sqlName
- LoggingFunc vlogPtr

Friends

- class **EE::UserDefinedAggregate**
- class **EE::UserDefinedAnalytic**
- class **EE::UserDefinedTransform**
- class **UdfSupport**

Detailed Description

Interface that UDX writers can use to interact with the Vertica Server

Constructor & Destructor Documentation

Vertica::ServerInterface::ServerInterface (VTAllocator * *allocator,* LoggingFunc *func,* const std::string & *sqlName,* const ParamReader & *paramReader*) `[inline]`

Create a new ServerInterface.

Note

that Only Vertica Server should use this method. It is not guaranteed to stay the same between releases.

Member Function Documentation

const std::string& Vertica::ServerInterface::getCurrentNodeName ( ) const `[inline]`

Returns

the name of the vertica node on which this code is executed.

const std::string& Vertica::ServerInterface::getLocale ( ) const `[inline]`

Returns

the locale of the current session.

ParamReader Vertica::ServerInterface::getParamReader ( ) `[inline]`

Returns the ParamReader that allows accessing parameter values using their names

void Vertica::ServerInterface::log (const char * *format,* ...)

Write a message to the vertica.log system log. The message will contain the SQL name of the user defined function or transform being called

Parameters

format	a printf style format string specifying the log message format.

void Vertica::ServerInterface::setParamReader (const ParamReader & *paramReader*) [inline]

Set the paramReader of this ServerInterface when delayed creation is required Used by the code when delayed creation of the parameters is needed Users should not call this function

void Vertica::ServerInterface::vlog (const char * *format,* va_list *ap*) [inline]

Write a message to the vertica.log system log.

Parameters

format	a printf style format string specifying the log message format.
ap	va_list for variable arguments

Member Data Documentation

VTAllocator* Vertica::ServerInterface::allocator

Memory source which is managed and freed by the server.

std::string Vertica::ServerInterface::locale [protected]

The locale of the current session

std::string Vertica::ServerInterface::nodeName [protected]

Store the name of the current node

ParamReader Vertica::ServerInterface::paramReader [protected]

A reader for paremeters that have been toknized using the following format: key1=val1,key2=val2,key3=val3. Has accessor methods like BlockReader to be able to access parameters of different data types

std::string Vertica::ServerInterface::sqlName [protected]

Store the name for error logging

LoggingFunc Vertica::ServerInterface::vlogPtr [protected]

Callback for logging, set by the server

Vertica::SizedColumnTypes Class Reference

Represents types and information to determine the size of the columns as input/output of a User Defined Function/Transform.

Public Member Functions

- void addBinary (int32 len, const std::string &fieldName="")

  Adds a column of type BINARY.
- void addBinaryOrderColumn (int32 len, const std::string &fieldName="")

  Adds an order column of type BINARY.
- void addBinaryPartitionColumn (int32 len, const std::string &fieldName="")

  Adds a partition column of type BINARY.
- void addBool (const std::string &fieldName="")

  Adds a column of type BOOLEAN.
- void addBoolOrderColumn (const std::string &fieldName="")

  Adds an order column of type BOOLEAN.
- void addBoolPartitionColumn (const std::string &fieldName="")

  Adds a partition column of type BOOLEAN.
- void addChar (int32 len, const std::string &fieldName="")

  Adds a column of type CHAR.
- void addCharOrderColumn (int32 len, const std::string &fieldName="")

  Adds an order column of type CHAR.
- void addCharPartitionColumn (int32 len, const std::string &fieldName="")

  Adds a partition column of type CHAR.
- void addDate (const std::string &fieldName="")

  Adds a column of type DATE.
- void addDateOrderColumn (const std::string &fieldName="")

  Adds an order column of type DATE.
- void addDatePartitionColumn (const std::string &fieldName="")

  Adds a partition column of type DATE.
- void addFloat (const std::string &fieldName="")

  Adds a column of type FLOAT.
- void addFloatOrderColumn (const std::string &fieldName="")

  Adds an order column of type FLOAT.
- void addFloatPartitionColumn (const std::string &fieldName="")

  Adds a partition column of type FLOAT.
- void addInt (const std::string &fieldName="")

  Adds a column of type INTEGER.

- void addInterval (int32 precision, int32 range, const std::string &fieldName="")

  Adds a column of type INTERVAL/INTERVAL DAY TO SECOND.
- void addIntervalOrderColumn (int32 precision, int32 range, const std::string &fieldName="")

  Adds an order column of type INTERVAL/INTERVAL DAY TO SECOND.
- void addIntervalPartitionColumn (int32 precision, int32 range, const std::string &fieldName="")

  Adds a partition column of type INTERVAL/INTERVAL DAY TO SECOND.
- void addIntervalYM (int32 range, const std::string &fieldName="")

  Adds a column of type INTERVAL YEAR TO MONTH.
- void addIntervalYMOrderColumn (int32 range, const std::string &fieldName="")

  Adds an order column of type INTERVAL YEAR TO MONTH.
- void addIntervalYMPartitionColumn (int32 range, const std::string &fieldName="")

  Adds a partition column of type INTERVAL YEAR TO MONTH.
- void addIntOrderColumn (const std::string &fieldName="")

  Adds an order column of type INTEGER.
- void addIntPartitionColumn (const std::string &fieldName="")

  Adds a partition column of type INTEGER.
- void addLongVarbinary (int32 len, const std::string &fieldName="")

  Adds a column of type LONG VARBINARY.
- void addLongVarbinaryOrderColumn (int32 len, const std::string &fieldName="")

  Adds an order column of type VARBINARY.
- void addLongVarbinaryPartitionColumn (int32 len, const std::string &fieldName="")

  Adds a partition column of type VARBINARY.
- void addLongVarchar (int32 len, const std::string &fieldName="")

  Adds a column of type LONG VARCHAR.
- void addLongVarcharOrderColumn (int32 len, const std::string &fieldName="")

  Adds an order column of type VARCHAR.
- void addLongVarcharPartitionColumn (int32 len, const std::string &fieldName="")

  Adds a partition column of type VARCHAR.
- void addNumeric (int32 precision, int32 scale, const std::string &fieldName="")

  Adds a column of type NUMERIC.

- void addNumericOrderColumn (int32 precision, int32 scale, const std::string &fieldName="")

  Adds an order column of type NUMERIC.
- void addNumericPartitionColumn (int32 precision, int32 scale, const std::string &fieldName="")

  Adds a partition column of type NUMERIC.
- void addOrderColumn (const VerticaType &dt, const std::string &fieldName="")

  Adds an order column of the specified type. (only relevant to multiphase UDTs.)
- void addPartitionColumn (const VerticaType &dt, const std::string &fieldName="")

  Adds a partition column of the specified type (only relevant to multiphase UDTs.)
- void addTime (int32 precision, const std::string &fieldName="")

  Adds a column of type TIME.
- void addTimeOrderColumn (int32 precision, const std::string &fieldName="")

  Adds an order column of type TIME.
- void addTimePartitionColumn (int32 precision, const std::string &fieldName="")

  Adds a partition column of type TIME.
- void addTimestamp (int32 precision, const std::string &fieldName="")

  Adds a column of type TIMESTAMP.
- void addTimestampOrderColumn (int32 precision, const std::string &fieldName="")

  Adds an order column of type TIMESTAMP.
- void addTimestampPartitionColumn (int32 precision, const std::string &fieldName="")

  Adds a partition column of type TIMESTAMP.
- void addTimestampTz (int32 precision, const std::string &fieldName="")

  Adds a column of type TIMESTAMP WITH TIMEZONE.
- void addTimestampTzOrderColumn (int32 precision, const std::string &fieldName="")

  Adds an order column of type TIMESTAMP WITH TIMEZONE.
- void addTimestampTzPartitionColumn (int32 precision, const std::string &fieldName="")

  Adds a partition column of type TIMESTAMP WITH TIMEZONE.
- void addTimeTz (int32 precision, const std::string &fieldName="")

  Adds a column of type TIME WITH TIMEZONE.
- void addTimeTzOrderColumn (int32 precision, const std::string &fieldName="")

  Adds an order column of type TIME WITH TIMEZONE.
- void addTimeTzPartitionColumn (int32 precision, const std::string &fieldName="")

  Adds a partition column of type TIME WITH TIMEZONE.

- void addUserDefinedType (const char *typeName, int32 len, const std::string &fieldName="")

  Adds a column of a user-defined type.
- void addVarbinary (int32 len, const std::string &fieldName="")

  Adds a column of type VARBINARY.
- void addVarbinaryOrderColumn (int32 len, const std::string &fieldName="")

  Adds an order column of type VARBINARY.
- void addVarbinaryPartitionColumn (int32 len, const std::string &fieldName="")

  Adds a partition column of type VARBINARY.
- void addVarchar (int32 len, const std::string &fieldName="")

  Adds a column of type VARCHAR.
- void addVarcharOrderColumn (int32 len, const std::string &fieldName="")

  Adds an order column of type VARCHAR.
- void addVarcharPartitionColumn (int32 len, const std::string &fieldName="")

  Adds a partition column of type VARCHAR.
- void getArgumentColumns (std::vector< size_t > &cols) const

  Retrieves indexes of argument columns. Indexes in cols can be used in conjunction with other functions, e.g. getColumnType(size_t) and getColumnName(size_t).
- size_t getColumnCount () const

  Returns the number of columns.
- const std::string & getColumnName (size_t idx) const

  Returns the name of the column at the specified index.
- const VerticaType & getColumnType (size_t idx) const

  Returns the type of the column at the specified index.
- VerticaType & getColumnType (size_t idx)

  Returns the type of the column at the specified index.
- int getLastOrderColumnIdx () const

  Gets the last ORDER BY column index.
- int getLastPartitionColumnIdx () const

  Gets the last PARTITION BY column index.
- void getOrderByColumns (std::vector< size_t > &cols) const

  Retrieves indexes of ORDER BY columns in the OVER() clause. Indexes in cols can be used in conjunction with other functions, e.g. getColumnType(size_t) and getColumnName(size_t).
- void getPartitionByColumns (std::vector< size_t > &cols) const

  Retrieves indexes of PARTITION BY columns in the OVER() clause. Indexes in cols can be used in conjunction with other functions, e.g. getColumnType(size_t) and getColumnName(size_t).
- bool isOrderByColumn (int idx) const

  Indicates whether the column at the specified index is an ORDER BY column.

- bool isPartitionByColumn (int idx) const

  Indicates whether the column at the specified index is a PARTITION BY column.
- void setPartitionOrderColumnIdx (int partition_idx, int order_idx)

  Sets the PARTITION BY and ORDER BY column indexes.
- void setPartitionOrderColumnIdx (const SizedColumnTypes &other)

  Sets the PARTITION BY and ORDER BY column indexes from another SizedColumnTypes object.

Detailed Description

Represents types and information to determine the size of the columns as input/output of a User Defined Function/Transform.

This class is used to exchange size and precision information between Vertica and the user defined function/transform function. Vertica provides the user code with size/precision information about the particular data types that it has been called with, and expects the user code to provide size/precision information about what it will return.

Member Function Documentation

void Vertica::SizedColumnTypes::addBinary (int32 *len*, const std::string & *fieldName* = "") [inline]

Adds a column of type BINARY.

Parameters

len	The length of the binary string.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addBinaryOrderColumn (int32 *len*, const std::string & *fieldName* = "") [inline]

Adds an order column of type BINARY.

Parameters

len	The length of the binary string.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addBinaryPartitionColumn (int32 *len,* const std::string & *fieldName* = "") [inline]

Adds a partition column of type BINARY.

Parameters

len	The length of the binary string.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addBool (const std::string & *fieldName* = "") [inline]

Adds a column of type BOOLEAN.

Parameters

fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addBoolOrderColumn (const std::string & *fieldName* = "") [inline]

Adds an order column of type BOOLEAN.

Parameters

fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addBoolPartitionColumn (const std::string & *fieldName* = "") [inline]

Adds a partition column of type BOOLEAN.

Parameters

fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addChar (int32 *len,* const std::string & *fieldName* = "") [inline]

Adds a column of type CHAR.

Parameters

len	The length of the string.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addCharOrderColumn (int32 *len,* const std::string & *fieldName =* "") [inline]

Adds an order column of type CHAR.

Parameters

len	The length of the string.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addCharPartitionColumn (int32 *len,* const std::string & *fieldName =* "") [inline]

Adds a partition column of type CHAR.

Parameters

len	The length of the string.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addDate (const std::string & *fieldName =* "") [inline]

Adds a column of type DATE.

Parameters

fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addDateOrderColumn (const std::string & *fieldName =* "") [inline]

Adds an order column of type DATE.

Parameters

fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addDatePartitionColumn (const std::string & *fieldName* = "") [inline]

Adds a partition column of type DATE.

Parameters

fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addFloat (const std::string & *fieldName* = "") [inline]

Adds a column of type FLOAT.

Parameters

fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addFloatOrderColumn (const std::string & *fieldName* = "") [inline]

Adds an order column of type FLOAT.

Parameters

fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addFloatPartitionColumn (const std::string & *fieldName* = "") [inline]

Adds a partition column of type FLOAT.

Parameters

fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addInt (const std::string & *fieldName* = "") [inline]

Adds a column of type INTEGER.

Parameters

fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addInterval (int32 *precision*, int32 *range*, const std::string & *fieldName* = "") [inline]

Adds a column of type INTERVAL/INTERVAL DAY TO SECOND.

Parameters

precision	The precision for the interval.
range	The range for the interval.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addIntervalOrderColumn (int32 *precision*, int32 *range*, const std::string & *fieldName* = "") [inline]

Adds an order column of type INTERVAL/INTERVAL DAY TO SECOND.

Parameters

precision	The precision for the interval.
range	The range for the interval.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addIntervalPartitionColumn (int32 *precision*, int32 *range*, const std::string & *fieldName* = "") [inline]

Adds a partition column of type INTERVAL/INTERVAL DAY TO SECOND.

Parameters

precision	The precision for the interval.
range	The range for the interval.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addIntervalYM (int32 *range*, const std::string & *fieldName* = "") [inline]

Adds a column of type INTERVAL YEAR TO MONTH.

Parameters

range	The range for the interval.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addIntervalYMOrderColumn (int32 *range*, const std::string & *fieldName* = "") [inline]

Adds an order column of type INTERVAL YEAR TO MONTH.

Parameters

range	The range for the interval.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addIntervalYMPartitionColumn (int32 *range*, const std::string & *fieldName* = "") [inline]

Adds a partition column of type INTERVAL YEAR TO MONTH.

Parameters

range	The range for the interval.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addIntOrderColumn (const std::string & *fieldName* = "") [inline]

Adds an order column of type INTEGER.

Parameters

fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addIntPartitionColumn (const std::string & *fieldName* = "") [inline]

Adds a partition column of type INTEGER.

Parameters

fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addLongVarbinary (int32 *len*, const std::string & *fieldName* = "") [inline]

Adds a column of type LONG VARBINARY.

Parameters

len	The length of the binary string.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addLongVarbinaryOrderColumn (int32 *len*, const std::string & *fieldName* = "") [inline]

Adds an order column of type VARBINARY.

Parameters

len	The length of the binary string.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addLongVarbinaryPartitionColumn (int32 *len*, const std::string & *fieldName* = "") [inline]

Adds a partition column of type VARBINARY.

Parameters

len	The length of the binary string.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addLongVarchar (int32 *len*, const std::string & *fieldName* = "") [inline]

Adds a column of type LONG VARCHAR.

Parameters

len	The length of the string.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addLongVarcharOrderColumn (int32 *len*, const std::string & *fieldName* = "") `[inline]`

Adds an order column of type VARCHAR.

Parameters

len	The length of the string.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addLongVarcharPartitionColumn (int32 *len*, const std::string & *fieldName* = "") `[inline]`

Adds a partition column of type VARCHAR.

Parameters

len	The length of the string.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addNumeric (int32 *precision*, int32 *scale*, const std::string & *fieldName* = "") `[inline]`

Adds a column of type NUMERIC.

Parameters

precision	The precision for the numeric value.
scale	The scale for the numeric value.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addNumericOrderColumn (int32 *precision*, int32 *scale*, const std::string & *fieldName* = "") `[inline]`

Adds an order column of type NUMERIC.

Parameters

precision	The precision for the numeric value.
scale	The scale for the numeric value.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addNumericPartitionColumn (int32 *precision*, int32 *scale*, const std::string & *fieldName* = "") [inline]

Adds a partition column of type NUMERIC.

Parameters

precision	The precision for the numeric value.
scale	The scale for the numeric value.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addTime (int32 *precision*, const std::string & *fieldName* = "") [inline]

Adds a column of type TIME.

Parameters

precision	The precision for the time.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addTimeOrderColumn (int32 *precision*, const std::string & *fieldName* = "") [inline]

Adds an order column of type TIME.

Parameters

precision	The precision for the time.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addTimePartitionColumn (int32 *precision*, const std::string & *fieldName* = "") [inline]

Adds a partition column of type TIME.

Parameters

precision	The precision for the time.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addTimestamp (int32 *precision,* const std::string & *fieldName* = "") [inline]

Adds a column of type TIMESTAMP.

Parameters

precision	The precision for the timestamp.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addTimestampOrderColumn (int32 *precision,* const std::string & *fieldName* = "") [inline]

Adds an order column of type TIMESTAMP.

Parameters

precision	The precision for the timestamp.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addTimestampPartitionColumn (int32 *precision,* const std::string & *fieldName* = "") [inline]

Adds a partition column of type TIMESTAMP.

Parameters

precision	The precision for the timestamp.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addTimestampTz (int32 *precision,* const std::string & *fieldName* = "") [inline]

Adds a column of type TIMESTAMP WITH TIMEZONE.

Parameters

precision	The precision for the timestamp.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addTimestampTzOrderColumn (int32 *precision,* const std::string & *fieldName* = "") [inline]

Adds an order column of type TIMESTAMP WITH TIMEZONE.

Parameters

precision	The precision for the timestamp.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addTimestampTzPartitionColumn (int32 *precision,* const std::string & *fieldName* = "") [inline]

Adds a partition column of type TIMESTAMP WITH TIMEZONE.

Parameters

precision	The precision for the timestamp.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addTimeTz (int32 *precision,* const std::string & *fieldName* = "") [inline]

Adds a column of type TIME WITH TIMEZONE.

Parameters

precision	The precision for the time.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addTimeTzOrderColumn (int32 *precision,* const std::string & *fieldName* = "") [inline]

Adds an order column of type TIME WITH TIMEZONE.

Parameters

precision	The precision for the time.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addTimeTzPartitionColumn (int32 *precision,* const std::string & *fieldName* = "") [inline]

Adds a partition column of type TIME WITH TIMEZONE.

Parameters

precision	The precision for the time.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addUserDefinedType (const char * *typeName,* int32 *len,* const std::string & *fieldName* = "") [inline]

Adds a column of a user-defined type.

Parameters

typeName	the name of the type
len	the length of the type field, in bytes
fieldName	the name of the field
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addVarbinary (int32 *len,* const std::string & *fieldName* = "") [inline]

Adds a column of type VARBINARY.

Parameters

len	The length of the binary string.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addVarbinaryOrderColumn (int32 *len,* const std::string & *fieldName* = "") [inline]

Adds an order column of type VARBINARY.

Parameters

len	The length of the binary string.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addVarbinaryPartitionColumn (int32 *len,* const std::string & *fieldName* = "") [inline]

Adds a partition column of type VARBINARY.

Parameters

len	The length of the binary string.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addVarchar (int32 *len,* const std::string & *fieldName* = "") [inline]

Adds a column of type VARCHAR.

Parameters

len	The length of the string.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addVarcharOrderColumn (int32 *len,* const std::string & *fieldName* = "") [inline]

Adds an order column of type VARCHAR.

Parameters

len	The length of the string.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::addVarcharPartitionColumn (int32 *len,* const std::string & *fieldName* = "") [inline]

Adds a partition column of type VARCHAR.

Parameters

len	The length of the string.
fieldName	The name for the output column.

void Vertica::SizedColumnTypes::getArgumentColumns (std::vector< size_t > & *cols*) const `[inline]`

Retrieves indexes of argument columns. Indexes in cols can be used in conjunction with other functions, e.g. getColumnType(size_t) and getColumnName(size_t).

Parameters

cols	A vector to store the retrieved column indexes.

const std::string& Vertica::SizedColumnTypes::getColumnName (size_t *idx*) const `[inline]`

Returns the name of the column at the specified index.

Parameters

idx	The index of the column

const VerticaType& Vertica::SizedColumnTypes::getColumnType (size_t *idx*) const `[inline]`

Returns the type of the column at the specified index.

Parameters

idx	The index of the column

Returns

a VerticaType object describing the column's data type.

VerticaType& Vertica::SizedColumnTypes::getColumnType (size_t *idx*) `[inline]`

Returns the type of the column at the specified index.

Parameters

idx	The index of the column

void Vertica::SizedColumnTypes::getOrderByColumns (std::vector< size_t > & *cols*) const `[inline]`

Retrieves indexes of ORDER BY columns in the OVER() clause. Indexes in cols can be used in conjunction with other functions, e.g. getColumnType(size_t) and getColumnName(size_t).

Parameters

cols	A vector to store the retrieved column indexes.

void Vertica::SizedColumnTypes::getPartitionByColumns (std::vector< size_t > & *cols*) const `[inline]`

Retrieves indexes of PARTITION BY columns in the OVER() clause. Indexes in cols can be used in conjunction with other functions, e.g. getColumnType(size_t) and getColumnName(size_t).

Parameters

cols	A vector to store the retrieved column indexes.

bool Vertica::SizedColumnTypes::isOrderByColumn (int *idx*) const `[inline]`

Indicates whether the column at the specified index is an ORDER BY column.

Parameters

idx	The index of the column

bool Vertica::SizedColumnTypes::isPartitionByColumn (int *idx*) const `[inline]`

Indicates whether the column at the specified index is a PARTITION BY column.

Parameters

idx	The index of the column

void Vertica::SizedColumnTypes::setPartitionOrderColumnIdx (int *partition_idx*, int *order_idx*) `[inline]`

Sets the PARTITION BY and ORDER BY column indexes.

Parameters

partition_-idx	Index of the last partition-by column
order_idx	Index of the last order-by column

void Vertica::SizedColumnTypes::setPartitionOrderColumnIdx (const SizedColumnTypes & *other*) [inline]

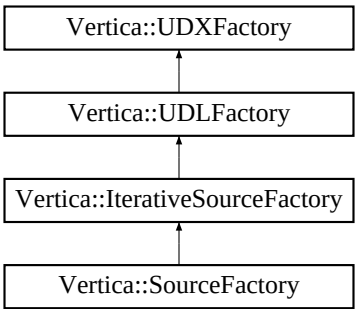
Sets the PARTITION BY and ORDER BY column indexes from another SizedColumn-Types object.

Parameters

other	The SizedColumnTypes object to set the indexes from

Vertica::SourceFactory Class Reference

Inheritance diagram for Vertica::SourceFactory:



Public Types

- enum UDXType {
  FUNCTION, **TRANSFORM**, **ANALYTIC**, **MULTI_TRANSFORM**, **AGGREGATE**, **LOAD_SOURCE**, **LOAD_FILTER**, **LOAD_PARSER**, **TYPE** }

Public Member Functions

- virtual void getParameterType (ServerInterface &srvInterface, SizedColumnTypes &parameterTypes)
- virtual void getPerInstanceResources (ServerInterface &srvInterface, VResources &res)

- void getPrototype (ServerInterface &srvInterface, ColumnTypes &argTypes, ColumnTypes &returnType)
- virtual void getReturnType (ServerInterface &srvInterface, const SizedColumnTypes &argTypes, SizedColumnTypes &returnType)
- UDXFactory::UDXType getUDXFactoryType ()
- virtual void plan (ServerInterface &srvInterface, NodeSpecifyingPlanContext &planCtxt)
- virtual SourceIterator * prepare (ServerInterface &srvInterface, NodeSpecifyingPlanContext &planCtxt)=0
- virtual std::vector< UDSource * > prepareUDSources (ServerInterface &srvInterface, NodeSpecifyingPlanContext &planCtxt)=0

Protected Attributes

- Oid **libOid**
- std::string **sqlName**

Detailed Description

A SourceFactory whose prepare() method constructs UDSources directly.

When implementing the factories for a UDSource, you have two options:

- Implement both an IterativeSourceFactory and a SourceIterator
- Implement just a SourceFactory (and no custom SourceIterator)

Factories should be registered using the RegisterFactory() macro, defined in Vertica.h.

Member Enumeration Documentation

enum Vertica::UDXFactory::UDXType `[inherited]`

The type of UDX instance this factory produces

Member Function Documentation

virtual void Vertica::UDXFactory::getParameterType (ServerInterface & *srvInterface*, SizedColumnTypes & *parameterTypes*) `[inline], [virtual], [inherited]`

Function to tell Vertica the name and types of parameters that this function uses. Vertica will use this to warn function callers that certain parameters they provide are not affecting anything, or that certain parameters that are not being set are reverting to default values.

Reimplemented in Vertica::ParserFactory.

virtual void Vertica::UDXFactory::getPerInstanceResources (ServerInterface & *srvInterface*, VResources & *res*) `[inline], [virtual], [inherited]`

Set the resource required for each instance of the UDX Object subclass

Parameters

srvInterface	a ServerInterface object used to communicate with Vertica
res	a VResources object used to tell Vertica what resources are needed by the UDX

void Vertica::UDLFactory::getPrototype (ServerInterface & *srvInterface*, ColumnTypes & *argTypes*, ColumnTypes & *returnType*) `[inline], [virtual], [inherited]`

Provides the argument and return types of the UDL. UDL's take no input tuples; as such, their prototype is empty.

Implements Vertica::UDXFactory.

virtual void Vertica::UDLFactory::getReturnType (ServerInterface & *srvInterface*, const SizedColumnTypes & *argTypes*, SizedColumnTypes & *returnType*) `[inline], [virtual], [inherited]`

Not used in this form

Implements Vertica::UDXFactory.

UDXFactory::UDXType Vertica::IterativeSourceFactory::getUDXFactoryType ( ) `[inline], [virtual], [inherited]`

Returns

the type of UDX Object instance this factory returns.

Note

User subclasses should use the appropriate subclass of UDXFactory and not override this method on their own.

Implements Vertica::UDXFactory.

virtual void Vertica::SourceFactory::plan (ServerInterface & *srvInterface*, NodeSpecifyingPlanContext & *planCtxt*) `[inline], [virtual]`

Execute any planning logic required at query plan time. This method is run once per query, during query initialization. Its job is to perform parameter validation, and to modify the set of nodes that the COPY statement will run on.

plan() runs exactly once per query, on the initiator node. If it throws an exception, the query will not proceed; it will be aborted prior to distributing the query to the other nodes and running prepare().

Parameters

srvInterface	Interface to server operations and functionality, including (not-per-column) parameter lookup
planCtxt	Context for storing and retrieving arbitrary data, for use just by this instance of this query. The same context is shared with plan(). Also provides functionality for specifying which nodes this query will run on.

Reimplemented from Vertica::IterativeSourceFactory.

virtual SourceIterator* Vertica::IterativeSourceFactory::prepare (ServerInterface & *srvInterface*, NodeSpecifyingPlanContext & *planCtxt*) `[pure virtual], [inherited]`

Prepare this SourceFactory to start creating sources. This function will be called on each node, prior to the Load operator starting to execute and prior to any other virtual functions on this class being called.

'planData' contains the same data that was placed there by the plan() static method.

If necessary, it is safe for this method to store any of the argument references as local fields on this instance. All will persist for the duration of the query.

virtual std::vector<UDSource*> Vertica::SourceFactory::prepareUDSources (ServerInterface & *srvInterface*, NodeSpecifyingPlanContext & *planCtxt*) `[pure virtual]`

Create UDSources. This function will be called on each node, prior to the Load operator starting to execute and prior to any other virtual functions on this class being called.

'planData' contains the same data that was placed there by the plan() static method.

If necessary, it is safe for this method to store any of the argument references as local fields on this instance. All will persist for the duration of the query.

Unlike the standard SourceFactory, this method directly instantiates all of its UDSources, and returns a vector of them. This requires that all UDSources be resident

in memory for the duration of the query, which is fine in the common case but which may not be acceptable for some resource-intensive UDSources.

Parameters

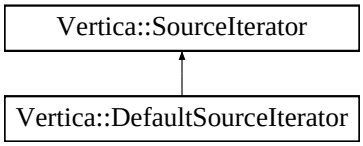
srvInterface	Interface to server operations and functionality, including (not-per-column) parameter lookup
planCtxt	Context for storing and retrieving arbitrary data, for use just by this instance of this query. The same context is shared with plan(). Also provides functionality for determining which nodes this query is running on.

Returns

A vector of UDSources to use for this query. Sources will be loaded in a pooled manner, several at a time.

Vertica::SourceIterator Class Reference

Inheritance diagram for Vertica::SourceIterator:



Public Member Functions

- virtual UnsizedUDSource * createNextSource (ServerInterface &srvInterface)=0

  Create the next UDSource to process.
- virtual size_t getNumberOfSources ()=0
- virtual size_t getSizeOfSource (size_t sourceNum)

Detailed Description

Wrappers to help construct and manage UDLs Construct a set of Sources. createNextSource() will be called repeatedly until it returns NULL. Each resulting Source will be read to completion, and the contained data passed to the Filter and Parser.

Member Function Documentation

virtual UnsizedUDSource* Vertica::SourceIterator::createNextSource (ServerInterface & *srvInterface*) `[pure virtual]`

Create the next UDSource to process.

Should return NULL if no further sources are available for processing.

Note that the previous Source may still be open and in use on a different thread when this function is called.

Returns

a new Source instance corresponding to a new input stream

Implemented in Vertica::DefaultSourceIterator.

virtual size_t Vertica::SourceIterator::getNumberOfSources ( ) `[pure virtual]`

Returns

the total number of Sources that this factory will produce

Implemented in Vertica::DefaultSourceIterator.

virtual size_t Vertica::SourceIterator::getSizeOfSource (size_t *sourceNum*) `[inline], [virtual]`

Returns

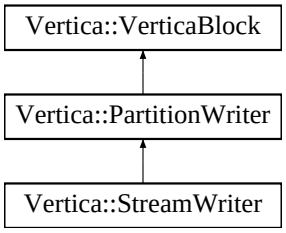
the raw-data size of the sourceNum'th source that will be produced by createNextSource(). Should return `vint_null` if the size is unknown.

This value is used as a hint, and is used by the "load_streams" table to display load progress. If incorrect or not set, "load_streams" may contain incorrect or unhelpful information.

Reimplemented in Vertica::DefaultSourceIterator.

Vertica::StreamWriter Class Reference

Inheritance diagram for Vertica::StreamWriter:



Public Member Functions

- **StreamWriter** (size_t narg, EE::Loader::UserDefinedLoad ∗udl)
- void copyFromInput (size_t dstIdx, PartitionReader &input_reader, size_t srcIdx)
- template<class T >
  const T ∗ getColPtr (size_t idx)
- template<class T >
  T ∗ **getColPtrForWrite** (size_t idx)
- template<class T >
  const T & getColRef (size_t idx)
- template<class T >
  T & **getColRefForWrite** (size_t idx)
- size_t getNumCols () const
- VNumeric & **getNumericRef** (size_t idx)
- int getNumRows () const
- VString & **getStringRef** (size_t idx)
- VString & **getStringRefNoClear** (size_t idx)
- virtual uint64 **getTotalRowCount** ()
- const SizedColumnTypes & getTypeMetaData () const
- SizedColumnTypes & getTypeMetaData ()
- void ∗ **getVoidPtr** (size_t idx)
- virtual bool getWriteableBlock ()
- bool **next** ()
- void **setBool** (size_t idx, vbool r)
- void **setDate** (size_t idx, DateADT r)
- void **setFloat** (size_t idx, vfloat r)
- void setInt (size_t idx, vint r)
- void **setInterval** (size_t idx, Interval r)
- void setNull (size_t idx)

  Set the idx'th argument to null.
- void **setTime** (size_t idx, TimeADT r)
- void **setTimestamp** (size_t idx, Timestamp r)
- void **setTimestampTz** (size_t idx, TimestampTz r)
- void **setTimeTz** (size_t idx, TimeTzADT r)
- void **validateColumn** (size_t idx)

Protected Member Functions

- void addCol (char *arg, int colstride, const VerticaType &dt, const std::string fieldName="")
- void **addCol** (const char *arg, int colstride, const VerticaType &dt, const std::string fieldName="")
- void **validateStringColumn** (size_t idx, const VString &s, const VerticaType &t)

Protected Attributes

- union {
  EE::UserDefinedAnalytic * **udan**
  EE::UserDefinedTransform * **udt**
  EE::UserDefinedProcess * **udx_object**
  };

- std::vector< char * > **cols**
- std::vector< int > **colstrides**
- int **count**
- int **index**
- size_t **ncols**
- EE::Loader::UserDefinedLoad * **parent**
- int **pstart**
- std::vector< VString > **svWrappers**
- SizedColumnTypes **typeMetaData**
- std::vector< VNumeric > **vnWrappers**

Friends

- class **EE::Loader::UserDefinedLoad**

Detailed Description

StreamWriter provides an iterator-based write interface over output data for a stream of blocks. Automatically makes space a block-at-a-time, as needed.

Member Function Documentation

void Vertica::VerticaBlock::addCol (char * *arg*, int *colstride*, const VerticaType & *dt*, const std::string *fieldName* = "") `[inline],[protected],[inherited]`

Add the location for reading a particular argument.

Parameters

arg	The base location to find data.
colstride	The stride between data instances.
dt	The type of input.
fieldname	the name of the field

void Vertica::PartitionWriter::copyFromInput (size_t *dstIdx,* PartitionReader & *input_reader,* size_t *srcIdx*) `[inline],[inherited]`

Copies a column from the input reader to the output writer. The data types and sizes of the source and destination columns must match exactly.

Parameters

dstIdx	The destination column index (in the output writer)
input_reader	The input reader from which to copy a column
srcIdx	The source column index (in the input reader)

template<class T > const T* Vertica::VerticaBlock::getColPtr (size_t *idx*) `[inline],[inherited]`

Returns

a pointer to the idx'th argument, cast appropriately.

Example:

```
const vint *a = arg_reader->getColPtr<vint>(0);
```

template<class T > const T& Vertica::VerticaBlock::getColRef (size_t *idx*) `[inline],[inherited]`

Returns

a pointer to the idx'th argument, cast appropriately.

Example: const vint a = arg_reader->getColRef<vint>(0);

size_t Vertica::VerticaBlock::getNumCols ( ) const `[inline],[inherited]`

Returns

the number of columns held by this block.

int Vertica::VerticaBlock::getNumRows ( ) const `[inline], [inherited]`

Returns

the number of rows held by this block.

const SizedColumnTypes& Vertica::VerticaBlock::getTypeMetaData ( ) const `[inline], [inherited]`

Returns

information about the types and numbers of arguments

SizedColumnTypes& Vertica::VerticaBlock::getTypeMetaData ( ) `[inline], [inherited]`

Returns

information about the types and numbers of arguments

virtual bool Vertica::StreamWriter::getWriteableBlock ( ) `[virtual]`

Gets a writeable block of data and positions cursor at the beginning.

Reimplemented from Vertica::PartitionWriter.

void Vertica::PartitionWriter::setInt (size_t *idx*, vint *r*) `[inline], [inherited]`

Setter methods

void Vertica::PartitionWriter::setNull (size_t *idx*) `[inline], [inherited]`

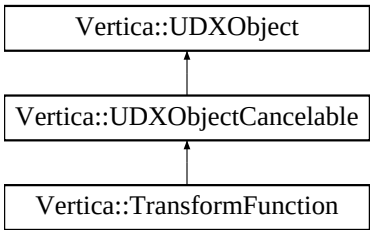
Set the idx'th argument to null.

Parameters

idx	The column number in the row to set to null

Vertica::TransformFunction Class Reference

Inheritance diagram for Vertica::TransformFunction:



Public Member Functions

- virtual void cancel (ServerInterface &srvInterface)
- virtual void destroy (ServerInterface &srvInterface, const SizedColumnTypes &arg-Types)
- bool isCanceled ()
- virtual void processPartition (ServerInterface &srvInterface, PartitionReader &input-_reader, PartitionWriter &output_writer)=0
- virtual void setup (ServerInterface &srvInterface, const SizedColumnTypes &arg-Types)

Protected Attributes

- volatile bool **canceled**

Detailed Description

Interface for User Defined Transform, the actual code to process a partition of data coming in as a stream

Member Function Documentation

virtual void Vertica::UDXObjectCancelable::cancel (ServerInterface & *srvInterface*) `[inline], [virtual], [inherited]`

This function is invoked from a different thread when the execution is canceled This baseclass cancel should be called in any override.

virtual void Vertica::UDXObject::destroy (ServerInterface & *srvInterface,* const SizedColumnTypes & *argTypes*) `[inline], [virtual], [inherited]`

Perform per instance destruction. This function may throw errors

bool Vertica::UDXObjectCancelable::isCanceled ( ) [inline], [inherited]

Returns true if execution was canceled.

virtual void Vertica::TransformFunction::processPartition (ServerInterface & *srvInterface,* PartitionReader & *input_reader,* PartitionWriter & *output_writer*) [pure virtual]

Invoke a user defined transform on a set of rows. As the name suggests, a batch of rows are passed in for every invocation to amortize performance.

Parameters

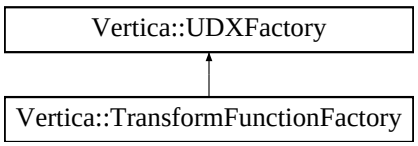
srvInterface	a ServerInterface object used to communicate with Vertica
input_reader	input rows
output_writer	output location

virtual void Vertica::UDXObject::setup (ServerInterface & *srvInterface,* const SizedColumnTypes & *argTypes*) [inline], [virtual], [inherited]

Perform per instance initialization. This function may throw errors.

Vertica::TransformFunctionFactory Class Reference

Inheritance diagram for Vertica::TransformFunctionFactory:



Public Types

- enum UDXType {
  FUNCTION, **TRANSFORM**, **ANALYTIC**, **MULTI_TRANSFORM**, **AGGREGATE**, **LOAD_SOURCE**, **LOAD_FILTER**, **LOAD_PARSER**, **TYPE** }

Public Member Functions

- virtual TransformFunction * createTransformFunction (ServerInterface &srvInterface)=0

- virtual void getParameterType (ServerInterface &srvInterface, SizedColumnTypes &parameterTypes)
- virtual void getPerInstanceResources (ServerInterface &srvInterface, VResources &res)
- virtual void getPrototype (ServerInterface &srvInterface, ColumnTypes &argTypes, ColumnTypes &returnType)=0
- virtual void getReturnType (ServerInterface &srvInterface, const SizedColumnTypes &argTypes, SizedColumnTypes &returnType)=0

Protected Member Functions

- virtual UDXType getUDXFactoryType ()

Protected Attributes

- Oid **libOid**
- std::string **sqlName**

Detailed Description

∗Interface to provide User Defined Transform compile time information

Member Enumeration Documentation

enum Vertica::UDXFactory::UDXType `[inherited]`

The type of UDX instance this factory produces

Member Function Documentation

virtual TransformFunction∗ Vertica::TransformFunctionFactory::createTransformFunction (ServerInterface & *srvInterface*) `[pure virtual]`

Called when Vertica needs a new TransformFunction object to process a UDTF function call.

Returns

an TransformFunction object which implements the UDx API described by this metadata.

Parameters

srvInterface	a ServerInterface object used to communicate with Vertica

Note

More than one object may be instantiated per query.

virtual void Vertica::UDXFactory::getParameterType (ServerInterface & *srvInterface*, SizedColumnTypes & *parameterTypes*) `[inline], [virtual], [inherited]`

Function to tell Vertica the name and types of parameters that this function uses. Vertica will use this to warn function callers that certain parameters they provide are not affecting anything, or that certain parameters that are not being set are reverting to default values.

Reimplemented in Vertica::ParserFactory.

virtual void Vertica::UDXFactory::getPerInstanceResources (ServerInterface & *srvInterface*, VResources & *res*) `[inline], [virtual], [inherited]`

Set the resource required for each instance of the UDX Object subclass

Parameters

srvInterface	a ServerInterface object used to communicate with Vertica
res	a VResources object used to tell Vertica what resources are needed by the UDX

virtual void Vertica::UDXFactory::getPrototype (ServerInterface & *srvInterface*, ColumnTypes & *argTypes*, ColumnTypes & *returnType*) `[pure virtual], [inherited]`

Provides the argument and return types of the UDX

Implemented in Vertica::UDLFactory, and Vertica::MultiPhaseTransformFunctionFactory.

virtual void Vertica::UDXFactory::getReturnType (ServerInterface & *srvInterface*, const SizedColumnTypes & *argTypes*, SizedColumnTypes & *returnType*) `[pure virtual], [inherited]`

Function to tell Vertica what the return types (and length/precision if necessary) of this UDX are.

For CHAR/VARCHAR types, specify the max length,

For NUMERIC types, specify the precision and scale.

For Time/Timestamp types (with or without time zone), specify the precision, -1 means unspecified/don't care

For IntervalYM/IntervalDS types, specify the precision and range

For all other types, no length/precision specification needed

Parameters

argTypes	Provides the data types of arguments that this UDT was called with. This may be used to modify the return types accordingly.
returnType	User code must fill in the names and data types returned by the UDT.

Implemented in Vertica::UDLFactory, Vertica::MultiPhaseTransformFunctionFactory, and Vertica::ScalarFunctionFactory.

virtual UDXType Vertica::TransformFunctionFactory::getUDXFactoryType ( )

`[inline], [protected], [virtual]`

Returns

the object type internally used by Vertica

Implements Vertica::UDXFactory.

Vertica::TransformFunctionPhase Class Reference

Public Member Functions

- virtual TransformFunction * createTransformFunction (ServerInterface &srvInterface)=0
- virtual void getReturnType (ServerInterface &srvInterface, const SizedColumnTypes &argTypes, SizedColumnTypes &returnTypes)=0
- bool **isPrepass** ()
- void setPrepass ()

  Indicates that this phase is a pre-pass (i.e., runs before any data movement)

Detailed Description

Interface to provide compile time information for a single phase of a multi-phase user-defined transform function. Note that even though this class shares some methods with TransformFunctionFactory, it is explicitly not a sub-class (since it is incorrect to implement a getPrototype() method for this class)

Member Function Documentation

virtual TransformFunction* Vertica::TransformFunctionPhase::create-TransformFunction (ServerInterface & *srvInterface*) `[pure virtual]`

Called when Vertica needs a new TransformFunction object to process this phase of a multi-phase UDT.

Returns

a TransformFunction object which implements the UDx API described by this metadata.

Parameters

srvInterface	a ServerInterface object used to communicate with Vertica

Note

More than one object may be instantiated per query.

virtual void Vertica::TransformFunctionPhase::getReturnType (ServerInterface & *srvInterface*, const SizedColumnTypes & *argTypes*, SizedColumnTypes & *returnTypes*) `[pure virtual]`

Function to tell Vertica what the return types (and length/precision if necessary) and partition-by, order-by of this phase are.

For CHAR/VARCHAR types, specify the max length,

For NUMERIC types, specify the precision and scale.

For Time/Timestamp types (with or without time zone), specify the precision, -1 means unspecified/don't care

For IntervalYM/IntervalDS types, specify the precision and range

For all other types, no length/precision specification needed

Parameters

argTypes	Provides the data types of arguments that this phase was called with, along with partition and order information. This may be used to modify the return types accordingly.
returnTypes	User code must fill in the names and data types returned by this phase, along with the partition-by and order-by column information (if any).

Vertica::UDFilter Class Reference

Public Member Functions

- virtual void destroy (ServerInterface &srvInterface)
- virtual StreamState process (ServerInterface &srvInterface, DataBuffer &input, InputState input_state, DataBuffer &output)=0
- virtual void setup (ServerInterface &srvInterface)

Detailed Description

UDFilter

Responsible for reading raw input data from a file and preparing it to be processed by a parser. This preparation may involve decompression, re-encoding, or any other sort of binary manipulation.

Member Function Documentation

virtual void Vertica::UDFilter::destroy (ServerInterface & *srvInterface*) `[inline], [virtual]`

UDFilter::destroy()

Will be invoked during query execution, after the last time that process() is called on this UDFilter instance for a particular input file.

May optionally be overridden to perform tear-down/destruction.

See UDFilter::setup() for a note about the restartability of UDFilters.

virtual StreamState Vertica::UDFilter::process (ServerInterface & *srvInterface*, DataBuffer & *input*, InputState *input_state*, DataBuffer & *output*) `[pure virtual]`

UDFilter::process()

Will be invoked repeatedly during query execution, until it returns DONE or until the query is canceled by the user.

On each invocation, process() is handed some input data and a buffer to write output data into. It is expected to read and process some amount of the input data, write some amount of output data, and return a value that informs Vertica what needs to happen next.

process() must set `input.offset` to the number of bytes that were successfully read from the `input` buffer, and that will not need to be re-consumed by a subsequent invocation of process(). This may not be larger than `input.size`. (`input.size` is the size of the buffer.) If it is set to 0, this indicates that process() cannot process

any part of an input buffer of this size, and requires more data per invocation. (For example, a block-based decompression algorithm might return 0 if the input buffer does not contain a complete block.)

Note that `input` may contain null bytes, if the source file contains null bytes. Note also that `input` is NOT automatically null-terminated.

If `input_state` == END_OF_FILE, then the last byte in `input` is the last byte in the input stream. Returning INPUT_NEEDED will not result in any new input appearing. process() should return DONE in this case as soon as this operator has finished producing all output that it is going to produce.

process() must set `output.offset` to the number of bytes that were written to the `output` buffer. This may not be larger than `output.size`. If it is set to 0, this indicates that process() requires a larger output buffer.

process() must not block indefinitely. If it cannot proceed for an extended period of time, it should return KEEP_GOING. It will be called again shortly. Failure to do this will, among other things, prevent the query from being canceled by the user.

Returns

OUTPUT_NEEDED if this UDFilter has more data to produce; INPUT_NEEDED if it needs more data to continue working; DONE if has no more data to produce.

Note that it is UNSAFE to maintain pointers or references to any of these arguments (or any other argument passed by reference into any other function in this API) beyond the scope of the function call in question. For example, do not store a reference to the server interface or the input block on an instance variable. Vertica may free and replace these objects.

virtual void Vertica::UDFilter::setup (ServerInterface & *srvInterface*) `[inline], [virtual]`

UDFilter::setup()

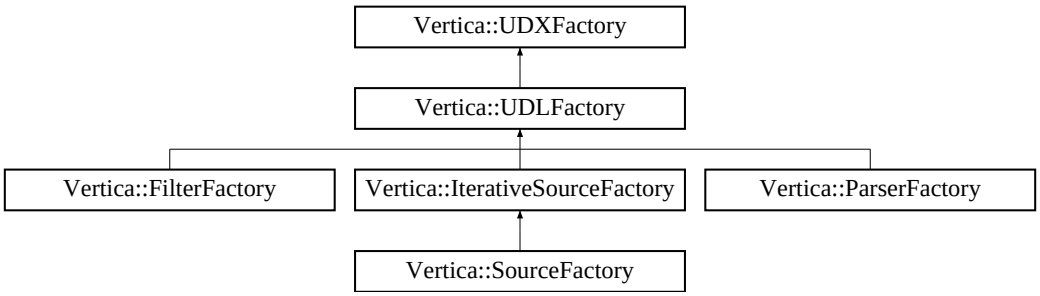
Will be invoked during query execution, prior to the first time that process() is called on this UDFilter instance for a particular input file.

May optionally be overridden to perform setup/initialzation.

Note that UDFilters MUST BE RESTARTABLE! If loading large numbers of files, a given UDFilter may be re-used for multiple files. Vertica follows the worker-pool design pattern: At the start of COPY execution, several Parsers and several Filters are instantiated per node, by calling the corresponding prepare() method multiple times. Each Filter/Parser pair is then internally assigned to an initial Source (UDSource or internal). At that point, setup() is called; then process() is called until it is finished; then destroy() is called. If there are still sources in the pool waiting to be processed, then the UDFilter/UDSource pair will be given a second Source; setup() will be called a second time, then process() until it is finished, then destroy(). This repeats until all sources have been read.

Vertica::UDLFactory Class Reference

Inheritance diagram for Vertica::UDLFactory:



Public Types

- enum UDXType {
  FUNCTION, **TRANSFORM**, **ANALYTIC**, **MULTI_TRANSFORM**,
  AGGREGATE, **LOAD_SOURCE**, **LOAD_FILTER**, **LOAD_PARSER**,
  TYPE }

Public Member Functions

- virtual void getParameterType (ServerInterface &srvInterface, SizedColumnTypes &parameterTypes)
- virtual void getPerInstanceResources (ServerInterface &srvInterface, VResources &res)
- void getPrototype (ServerInterface &srvInterface, ColumnTypes &argTypes, ColumnTypes &returnType)
- virtual void getReturnType (ServerInterface &srvInterface, const SizedColumnTypes &argTypes, SizedColumnTypes &returnType)
- virtual UDXType getUDXFactoryType ()=0

Protected Attributes

- Oid **libOid**
- std::string **sqlName**

Member Enumeration Documentation

enum Vertica::UDXFactory::UDXType `[inherited]`

The type of UDX instance this factory produces

Member Function Documentation

virtual void Vertica::UDXFactory::getParameterType (ServerInterface & *srvInterface*, SizedColumnTypes & *parameterTypes*) `[inline], [virtual], [inherited]`

Function to tell Vertica the name and types of parameters that this function uses. Vertica will use this to warn function callers that certain parameters they provide are not affecting anything, or that certain parameters that are not being set are reverting to default values.

Reimplemented in Vertica::ParserFactory.

virtual void Vertica::UDXFactory::getPerInstanceResources (ServerInterface & *srvInterface*, VResources & *res*) `[inline], [virtual], [inherited]`

Set the resource required for each instance of the UDX Object subclass

Parameters

srvInterface	a ServerInterface object used to communicate with Vertica
res	a VResources object used to tell Vertica what resources are needed by the UDX

void Vertica::UDLFactory::getPrototype (ServerInterface & *srvInterface*, ColumnTypes & *argTypes*, ColumnTypes & *returnType*) `[inline], [virtual]`

Provides the argument and return types of the UDL. UDL's take no input tuples; as such, their prototype is empty.

Implements Vertica::UDXFactory.

virtual void Vertica::UDLFactory::getReturnType (ServerInterface & *srvInterface*, const SizedColumnTypes & *argTypes*, SizedColumnTypes & *returnType*) `[inline], [virtual]`

Not used in this form

Implements Vertica::UDXFactory.

virtual UDXType Vertica::UDXFactory::getUDXFactoryType ( ) `[pure virtual], [inherited]`

Returns

the type of UDX Object instance this factory returns.

Note

User subclasses should use the appropriate subclass of UDXFactory and not override this method on their own.

Implemented in Vertica::MultiPhaseTransformFunctionFactory, Vertica::AggregateFunctionFactory, Vertica::AnalyticFunctionFactory, Vertica::TransformFunctionFactory, Vertica::ScalarFunctionFactory, Vertica::ParserFactory, Vertica::FilterFactory, and Vertica::IterativeSourceFactory.

Vertica::UDParser Class Reference

Public Member Functions

- virtual void destroy (ServerInterface &srvInterface, SizedColumnTypes &returnType)
- virtual RejectedRecord getRejectedRecord ()
- virtual StreamState process (ServerInterface &srvInterface, DataBuffer &input, InputState input_state)=0
- void **setStreamWriter** (StreamWriter *writer)
- virtual void setup (ServerInterface &srvInterface, SizedColumnTypes &returnType)

Protected Attributes

- StreamWriter * writer

Detailed Description

UDParser

Responsible for parsing an input stream into Vertica tuples, rows to be inserted into a table.

Member Function Documentation

virtual void Vertica::UDParser::destroy (ServerInterface & *srvInterface*, SizedColumnTypes & *returnType*) `[inline], [virtual]`

UDParser::destroy()

Will be invoked during query execution, after the last time that process() is called on this UDParser instance for a particular input file.

May optionally be overridden to perform tear-down/destruction.

See UDParser::setup() for a note about the restartability of UDParsers.

virtual RejectedRecord Vertica::UDParser::getRejectedRecord ( ) [inline], [virtual]

Returns information about the rejected record

virtual StreamState Vertica::UDParser::process (ServerInterface & *srvInterface*, DataBuffer & *input*, InputState *input_state*) [pure virtual]

UDParser::process()

Will be invoked repeatedly during query execution, until it returns DONE or until the query is canceled by the user.

On each invocation, process() will be given an input buffer. It should read data from that buffer, converting it to fields and tuples and writing those tuples via `writer`. Once it has consumed as much as it reasonably can (for example, once it has consumed the last complete row in the input buffer), it should return INPUT_NEEDED to indicate that more data is needed, or DONE to indicate that it has completed parsing this input stream and will not be reading more bytes from it.

If `input_state` == END_OF_FILE, then the last byte in `input` is the last byte in the input stream. Returning INPUT_NEEDED will not result in any new input appearing. process() should return DONE in this case as soon as this operator has finished producing all output that it is going to produce.

Note that `input` may contain null bytes, if the source file contains null bytes. Note also that `input` is NOT automatically null-terminated.

process() must not block indefinitely. If it cannot proceed for an extended period of time, it should return KEEP_GOING. It will be called again shortly. Failure to do this will, among other things, prevent the query from being canceled by the user.

Row Rejection

process() can "reject" a row, causing it to be logged by Vertica's rejected-rows mechanism. Rejected rows should not be emitted as tuples. A rejected row must start at the first byte of `input` (meaning all previous input must have been consumed by a previous call to process()). To reject a row, set `input.offset` to the size of the row, and return REJECT.

Returns

INPUT_NEEDED if this UDParser has more data to produce; DONE if has no more data to produce; REJECT to reject a row

Note that it is UNSAFE to maintain pointers or references to any of these arguments (or any other argument passed by reference into any other function in this API) beyond the scope of the function call in question. For example, do not store a reference to the server interface or the input block on an instance variable. Vertica may free and replace these objects.

**virtual void Vertica::UDParser::setup (ServerInterface & *srvInterface,*
SizedColumnTypes & *returnType*) `[inline], [virtual]`

UDParser::setup()

Will be invoked during query execution, prior to the first time that process() is called on this UDParser instance for a particular input source.

May optionally be overridden to perform setup/initialzation.

Note that UDParsers MUST BE RESTARTABLE! If loading large numbers of files, a given UDParsers may be re-used for multiple files. Vertica follows the worker-pool design pattern: At the start of COPY execution, several Parsers and several Filters are instantiated per node, by calling the corresponding prepare() method multiple times. Each Filter/Parser pair is then internally assigned to an initial Source (UDSource or internal). At that point, setup() is called; then process() is called until it is finished; then destroy() is called. If there are still sources in the pool waiting to be processed, then the UDFilter/UDSource pair will be given a second Source; setup() will be called a second time, then process() until it is finished, then destroy(). This repeats until all sources have been read.

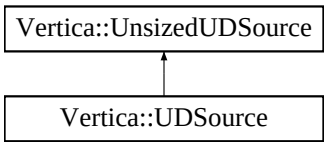
Member Data Documentation

StreamWriter* Vertica::UDParser::writer `[protected]`

Writer to write parsed tuples to. Has the same API as PartitionWriter, from the UDT framework.

Vertica::UDSource Class Reference

Inheritance diagram for Vertica::UDSource:



Public Member Functions

- virtual void destroy (ServerInterface &srvInterface)
- virtual vint getSize ()
- virtual std::string getUri ()
- virtual StreamState process (ServerInterface &srvInterface, DataBuffer &output)=0
- virtual void setup (ServerInterface &srvInterface)

Detailed Description

UDSource

Responsible for acquiring data from an external source (such as a file, a URL, etc) and producing that data in a streaming manner.

Member Function Documentation

virtual void Vertica::UDSource::destroy (ServerInterface & *srvInterface*) `[inline], [virtual]`

UDSource::destroy()

Will be invoked during query execution, after the last time that process() is called on this UDSource instance.

May optionally be overridden to perform tear-down/destruction.

Reimplemented from Vertica::UnsizedUDSource.

virtual vint Vertica::UDSource::getSize ( ) `[inline], [virtual]`

UDSource::getSize()

Returns the estimated number of bytes that process() will return.

This value is treated as advisory only. It is used to indicate the file size in the LOAD_-STREAMS table.

IMPORTANT: getSize() can be called at any time, even before setup() is called! (Though not before or during the constructor.)

In the case of Sources whose factories can potentially produce many UDSource instances, getSize() should avoid acquiring resources that last for the life of the object. Doing otherwise can defeat Vertica's attempts to limit the maximum number of Sources that are consuming system resources at any given time. For example, if it opens a file handle and leaves that file handle open for use by process(), and if a large number of UDSources are loaded in a single statement, the query may exceed the operating system limit on file handles and crash, even though Vertica only operates on a small number of files at once. This doesn't apply to singleton Sources, Sources whose factory will only ever produce one UDSource instance.

virtual std::string Vertica::UnsizedUDSource::getUri ( ) `[inline], [virtual], [inherited]`

UnsizedUDSource::getUri()

Return the URI of the current source of data.

This function will be invoked during execution to fill in monitoring information.

virtual StreamState Vertica::UDSource::process (ServerInterface & *srvInterface*, DataBuffer & *output*) `[pure virtual]`

UDSource::process()

Will be invoked repeatedly during query execution, until it returns DONE or until the query is canceled by the user.

On each invocation, process() should acquire more data and write that data to the buffer specified by `output`.

process() must set `output.offset` to the number of bytes that were written to the `output` buffer. It is common, though not necessary, for this to be the same as `output.size`. `output.offset` is initially uninitialized. If it is set to 0, this indicates that the `output` buffer is too small for process() to be able to write a unit of input to it.

process() must not block indefinitely. If it cannot proceed for an extended period of time, it should return KEEP_GOING. It will be called again shortly. Failure to do this will, among other things, prevent the query from being canceled by the user.

Returns

OUTPUT_NEEDED if this UDSource has more data to produce; DONE if has no more data to produce.

Note that it is UNSAFE to maintain pointers or references to any of these arguments (or any other argument passed by reference into any other function in this API) beyond the scope of the function call in question. For example, do not store a reference to the server interface or the input block on an instance variable. Vertica may free and replace these objects.

Implements Vertica::UnsizedUDSource.

virtual void Vertica::UDSource::setup (ServerInterface & *srvInterface*) `[inline], [virtual]`

UDSource::setup()

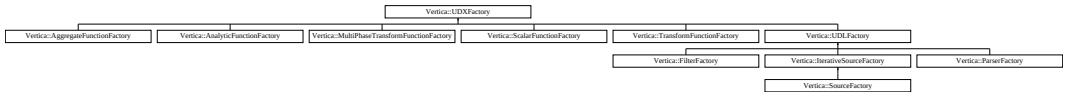
Will be invoked during query execution, prior to the first time that process() is called on this UDSource instance.

May optionally be overridden to perform setup/initialzation.

Reimplemented from Vertica::UnsizedUDSource.

Vertica::UDXFactory Class Reference

Inheritance diagram for Vertica::UDXFactory:



Public Types

- enum UDXType {
  FUNCTION, **TRANSFORM**, **ANALYTIC**, **MULTI_TRANSFORM**, **AGGREGATE**, **LOAD_SOURCE**, **LOAD_FILTER**, **LOAD_PARSER**, **TYPE** }

Public Member Functions

- virtual void getParameterType (ServerInterface &srvInterface, SizedColumnTypes &parameterTypes)
- virtual void getPerInstanceResources (ServerInterface &srvInterface, VResources &res)
- virtual void getPrototype (ServerInterface &srvInterface, ColumnTypes &argTypes, ColumnTypes &returnType)=0
- virtual void getReturnType (ServerInterface &srvInterface, const SizedColumnTypes &argTypes, SizedColumnTypes &returnType)=0
- virtual UDXType getUDXFactoryType ()=0

Protected Attributes

- Oid **libOid**
- std::string **sqlName**

Friends

- class **UdfSupport**

Detailed Description

MetaData interface for Vertica User Defined extensions

Member Enumeration Documentation

enum Vertica::UDXFactory::UDXType

The type of UDX instance this factory produces

Member Function Documentation

virtual void Vertica::UDXFactory::getParameterType (ServerInterface & *srvInterface*, SizedColumnTypes & *parameterTypes*) [inline], [virtual]

Function to tell Vertica the name and types of parameters that this function uses. Vertica will use this to warn function callers that certain parameters they provide are not affecting anything, or that certain parameters that are not being set are reverting to default values.

Reimplemented in Vertica::ParserFactory.

virtual void Vertica::UDXFactory::getPerInstanceResources (ServerInterface & *srvInterface*, VResources & *res*) [inline], [virtual]

Set the resource required for each instance of the UDX Object subclass

Parameters

srvInterface	a ServerInterface object used to communicate with Vertica
res	a VResources object used to tell Vertica what resources are needed by the UDX

virtual void Vertica::UDXFactory::getPrototype (ServerInterface & *srvInterface*, ColumnTypes & *argTypes*, ColumnTypes & *returnType*) [pure virtual]

Provides the argument and return types of the UDX

Implemented in Vertica::UDLFactory, and Vertica::MultiPhaseTransformFunctionFactory.

virtual void Vertica::UDXFactory::getReturnType (ServerInterface & *srvInterface*, const SizedColumnTypes & *argTypes*, SizedColumnTypes & *returnType*) [pure virtual]

Function to tell Vertica what the return types (and length/precision if necessary) of this UDX are.

For CHAR/VARCHAR types, specify the max length,

For NUMERIC types, specify the precision and scale.

For Time/Timestamp types (with or without time zone), specify the precision, -1 means unspecified/don't care

For IntervalYM/IntervalDS types, specify the precision and range

For all other types, no length/precision specification needed

Parameters

argTypes	Provides the data types of arguments that this UDT was called with. This may be used to modify the return types accordingly.
returnType	User code must fill in the names and data types returned by the UDT.

Implemented in Vertica::UDLFactory, Vertica::MultiPhaseTransformFunctionFactory, and Vertica::ScalarFunctionFactory.

virtual UDXType Vertica::UDXFactory::getUDXFactoryType ( ) [pure virtual]

Returns

the type of UDX Object instance this factory returns.

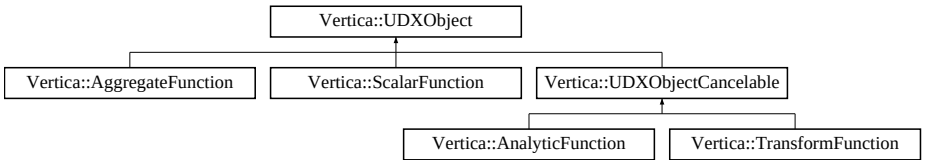
Note

User subclasses should use the appropriate subclass of UDXFactory and not override this method on their own.

Implemented in Vertica::MultiPhaseTransformFunctionFactory, Vertica::AggregateFunctionFactory, Vertica::AnalyticFunctionFactory, Vertica::TransformFunctionFactory, Vertica::ScalarFunctionFactory, Vertica::ParserFactory, Vertica::FilterFactory, and Vertica::IterativeSourceFactory.

Vertica::UDXObject Class Reference

Inheritance diagram for Vertica::UDXObject:



Public Member Functions

- virtual ~UDXObject ()
- virtual void destroy (ServerInterface &srvInterface, const SizedColumnTypes &argTypes)
- virtual void setup (ServerInterface &srvInterface, const SizedColumnTypes &argTypes)

Detailed Description

Base class for Vertica User Defined extensions, the object themselves

Constructor & Destructor Documentation

virtual Vertica::UDXObject::~UDXObject ( ) [inline], [virtual]

Destructors MAY NOT throw errors / exceptions. Exceptions thrown during the destructor will be ignored.

Member Function Documentation

virtual void Vertica::UDXObject::destroy (ServerInterface & *srvInterface,* const SizedColumnTypes & *argTypes*) [inline], [virtual]

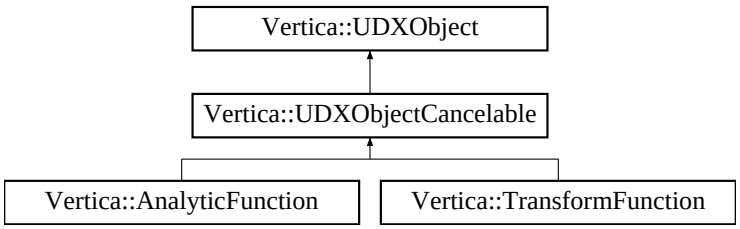
Perform per instance destruction. This function may throw errors

virtual void Vertica::UDXObject::setup (ServerInterface & *srvInterface,* const SizedColumnTypes & *argTypes*) [inline], [virtual]

Perform per instance initialization. This function may throw errors.

Vertica::UDXObjectCancelable Class Reference

Inheritance diagram for Vertica::UDXObjectCancelable:



Public Member Functions

- virtual void cancel (ServerInterface &srvInterface)
- virtual void destroy (ServerInterface &srvInterface, const SizedColumnTypes &argTypes)
- bool isCanceled ()
- virtual void setup (ServerInterface &srvInterface, const SizedColumnTypes &argTypes)

Protected Attributes

- volatile bool **canceled**

Member Function Documentation

virtual void Vertica::UDXObjectCancelable::cancel (ServerInterface & *srvInterface*) `[inline],[virtual]`

This function is invoked from a different thread when the execution is canceled This baseclass cancel should be called in any override.

virtual void Vertica::UDXObject::destroy (ServerInterface & *srvInterface,* const SizedColumnTypes & *argTypes*) `[inline],[virtual],[inherited]`

Perform per instance destruction. This function may throw errors

bool Vertica::UDXObjectCancelable::isCanceled ( ) `[inline]`

Returns true if execution was canceled.

virtual void Vertica::UDXObject::setup (ServerInterface & *srvInterface,* const SizedColumnTypes & *argTypes*) `[inline],[virtual],[inherited]`

Perform per instance initialization. This function may throw errors.

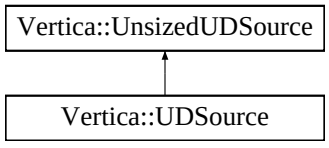
Vertica::UDxRegistrar Struct Reference

Public Member Functions

- **UDxRegistrar** (const char ∗name)

Vertica::UnsizedUDSource Class Reference

Inheritance diagram for Vertica::UnsizedUDSource:



Public Member Functions

- virtual void **destroy** (ServerInterface &srvInterface)
- virtual std::string getUri ()
- virtual StreamState **process** (ServerInterface &srvInterface, DataBuffer &output)=0
- virtual void **setup** (ServerInterface &srvInterface)

Detailed Description

UnsizedUDSource

Base class for UDSource. Used with IterativeSourceFactory if computing the size of a source up front would be prohibitively expensive, or if the number of distinct sources would be prohibitively large to use the standard API.

Not intended or optimized for typical applications.

Member Function Documentation

virtual std::string Vertica::UnsizedUDSource::getUri ( ) [inline], [virtual]

UnsizedUDSource::getUri()

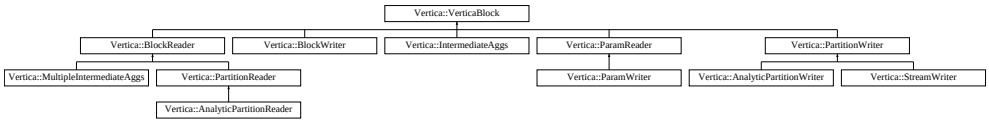
Return the URI of the current source of data.

This function will be invoked during execution to fill in monitoring information.

Vertica::VerticaBlock Class Reference

: Represents an in-memory block of tuples

Inheritance diagram for Vertica::VerticaBlock:



Public Member Functions

- **VerticaBlock** (size_t ncols, int rowcount)
- template<class T >
  const T * getColPtr (size_t idx)
- template<class T >
  T * **getColPtrForWrite** (size_t idx)

- template<class T >
  const T & getColRef (size_t idx)
- template<class T >
  T & **getColRefForWrite** (size_t idx)
- size_t getNumCols () const
- int getNumRows () const
- const SizedColumnTypes & getTypeMetaData () const
- SizedColumnTypes & getTypeMetaData ()

Protected Member Functions

- void addCol (char *arg, int colstride, const VerticaType &dt, const std::string fieldName="")
- void **addCol** (const char *arg, int colstride, const VerticaType &dt, const std::string fieldName="")
- void **validateStringColumn** (size_t idx, const VString &s, const VerticaType &t)

Protected Attributes

- std::vector< char * > **cols**
- std::vector< int > **colstrides**
- int **count**
- int **index**
- size_t **ncols**
- std::vector< VString > **svWrappers**
- SizedColumnTypes **typeMetaData**
- std::vector< VNumeric > **vnWrappers**

Friends

- void **AggregateFunction::updateCols** (BlockReader &arg_reader, char *arg, int count, IntermediateAggs &intAggs, char *aggPtr, std::vector< int > &intOffsets)
- void **deserializeVerticaParametersBlock** (VerticaBlock *block, const std::string &str)
- class **EE::UserDefinedAggregate**
- class **EE::UserDefinedAnalytic**
- class **EE::UserDefinedProcess**
- class **EE::UserDefinedTransform**
- class **VerticaBlockSerializer**

Detailed Description

: Represents an in-memory block of tuples

Member Function Documentation

void Vertica::VerticaBlock::addCol (char * *arg,* int *colstride,* const VerticaType & *dt,* const std::string *fieldName* = "") `[inline],[protected]`

Add the location for reading a particular argument.

Parameters

arg	The base location to find data.
colstride	The stride between data instances.
dt	The type of input.
fieldname	the name of the field

template<class T > const T* Vertica::VerticaBlock::getColPtr (size_t *idx*) `[inline]`

Returns

a pointer to the idx'th argument, cast appropriately.

Example:

```
const vint *a = arg_reader->getColPtr<vint>(0);
```

template<class T > const T& Vertica::VerticaBlock::getColRef (size_t *idx*) `[inline]`

Returns

a pointer to the idx'th argument, cast appropriately.

Example: const vint a = arg_reader->getColRef<vint>(0);

size_t Vertica::VerticaBlock::getNumCols ( ) const `[inline]`

Returns

the number of columns held by this block.

int Vertica::VerticaBlock::getNumRows ( ) const `[inline]`

Returns

the number of rows held by this block.

const SizedColumnTypes& Vertica::VerticaBlock::getTypeMetaData ( ) const `[inline]`

Returns

information about the types and numbers of arguments

SizedColumnTypes& Vertica::VerticaBlock::getTypeMetaData ( ) `[inline]`

Returns

information about the types and numbers of arguments

Vertica::VerticaBuildInfo Struct Reference

Public Attributes

- const char * **branch**
- const char * **brand_name**
- const char * **brand_version**
- const char * **build_date**
- const char * **build_machine**
- const char * **checksum**
- const char * **codename**
- const char * **revision**

Vertica::VerticaType Class Reference

Represents types of data that are passed into and returned back from user's code.

Public Member Functions

- int32 getIntervalPrecision () const

  For INTERVAL data types, returns the precision.
- int32 getIntervalRange () const

For INTERVAL data types, returns the range.

- int32 getMaxSize () const

  Returns the maximum size, in bytes, of a data element of this type.

- int32 getNumericFractional () const

  For NUMERIC data types, returns the number of fractional digits (i.e., digits right of the decimal point)

- int32 getNumericIntegral () const

  For NUMERIC data types, returns the number of integral digits (i.e., digits left of the decimal point)

- int32 getNumericLength () const

  For NUMERIC data types, returns the number of bytes required to store an element. Calling this with a non-numeric data type can cause a crash.

- int32 getNumericPrecision () const

  For NUMERIC data types, returns the precision.

- int32 getNumericScale () const

  For NUMERIC data types, returns the scale.

- int32 **getNumericWordCount** () const
- std::string getPrettyPrintStr () const

  Return human readable type string.

- int32 getStringLength () const

  For VARCHAR/CHAR/VARBINARY/BINARY data types, returns the length of the string.

- int32 getTimePrecision () const

  For TIMESTAMP data types, returns the precision.

- int32 getTimestampPrecision () const

  For TIMESTAMP data types, returns the precision.

- const char * **getTypeStr** () const
- Oid getUnderlyingType () const
- bool isBinary () const

  Returns true if this type is BINARY, false otherwise.

- bool isBool () const

  Returns true if this type is BOOLEAN, false otherwise.

- bool isChar () const

  Returns true if this type is CHAR, false otherwise.

- bool isDate () const

  Returns true if this type is DATE, false otherwise.

- bool isFloat () const

  Returns true if this type is FLOAT, false otherwise.

- bool isInt () const

  Returns true if this type is INTEGER, false otherwise.

- bool isInterval () const

  Returns true if this type is INTERVAL, false otherwise.
- bool isIntervalYM () const

  Returns true if this type is INTERVAL YEAR TO MONTH, false otherwise.
- bool isLongVarbinary () const

  Returns true if this type is LONG VARCHAR, false otherwise.
- bool isLongVarchar () const

  Returns true if this type is LONG VARCHAR, false otherwise.
- bool isNumeric () const

  Returns true if this type is NUMERIC, false otherwise.
- bool **isStringOid** (Oid typeOid) const
- bool isStringType () const

  Return true for VARCHAR/CHAR/VARBINARY/BINARY/LONG VARCHAR/LONG VARBINARY data types.
- bool isTime () const

  Returns true if this type is TIME, false otherwise.
- bool isTimestamp () const

  Returns true if this type is TIMESTAMP, false otherwise.
- bool isTimestampTz () const

  Returns true if this type is TIMESTAMP WITH TIMEZONE, false otherwise.
- bool isTimeTz () const

  Returns true if this type is TIME WITH TIMEZONE, false otherwise.
- bool isVarbinary () const

  Returns true if this type is VARBINARY, false otherwise.
- bool isVarchar () const

  Returns true if this type is VARCHAR, false otherwise.
- bool **operator!=** (const VerticaType &rhs) const
- bool **operator==** (const VerticaType &rhs) const
- void setIntervalPrecision (int32 precision)

  For INTERVAL data types, sets the precision.
- void setIntervalRange (int32 range)

  For INTERVAL data types, sets the range.
- void setNumericPrecision (int32 precision)

  For NUMERIC data types, sets the precision.
- void setNumericScale (int32 scale)

  For NUMERIC data types, sets the scale.
- void setTimePrecision (int32 precision)

  For TIMESTAMP data types, sets the precision.
- void setTimestampPrecision (int32 precision)

  For TIMESTAMP data types, sets the precision.

Detailed Description

Represents types of data that are passed into and returned back from user's code.

Member Function Documentation

Oid Vertica::VerticaType::getUnderlyingType ( ) const `[inline]`

Returns

If this is a built in type, returns the typeOid. Otherwise, if a user defined type returns the base type oid on which the user type is based.

Note: This function is designed so that the common case (aka a built in, non user defined type) is fast – calling isUDType() is a heavyweight operation. – See VER-24673 for an example of when it matters.

Vertica::VInterval Class Reference

Representation of an Interval in Vertica.

Static Public Member Functions

- static void breakUp (Interval i, int64 &days, int64 &hour, int64 &min, float &sec)

  Break up an Interval and set the arguments.

Detailed Description

Representation of an Interval in Vertica.

Member Function Documentation

static void Vertica::VInterval::breakUp (Interval *i,* int64 & *days,* int64 & *hour,* int64 & *min,* float & *sec*) `[inline],[static]`

Break up an Interval and set the arguments.

Returns

None

Parameters

i]	Vertica Interval.
days]	Number of days in the interval.
hour]	Number of hours in the interval.
min]	Number of minutes in the interval.
sec]	Number of seconds including fractions of a second.

Vertica::VIntervalYM Class Reference

Representation of an IntervalYM in Vertica. An Interval can be broken up into years and months.

Static Public Member Functions

- static void breakUp (IntervalYM i, int64 &years, int64 &months)

  Break up an Interval and set the arguments.

Detailed Description

Representation of an IntervalYM in Vertica. An Interval can be broken up into years and months.

Member Function Documentation

static void Vertica::VIntervalYM::breakUp (IntervalYM *i,* int64 & *years,* int64 & *months*) `[inline], [static]`

Break up an Interval and set the arguments.

Returns

None

Parameters

i]	Vertica IntervalYM.
years]	Number of years in the interval.
months]	Number of months in the interval.

Vertica::VNumeric Class Reference

Representation of NUMERIC, fixed point data types in Vertica.

Public Member Functions

- VNumeric (uint64 *words, int32 precision, int32 scale)

  Create a VNumeric with the provided storage location, precision and scale.
- void accumulate (const VNumeric *from)

  Adds another VNumeric to this VNumeric.
- void **add** (const VNumeric *a, const VNumeric *b)
- int compare (const VNumeric *from) const

  Compares this (signed) VNumeric to another.
- int compareUnsigned (const VNumeric *from) const

  Compares this (unsigned) VNumeric to another.
- void copy (const VNumeric *from)

  Copy data from another VNumeric.
- bool copy (ifloat value, bool round=true)

  Copy data from a floating-point number.
- void **div** (const VNumeric *a, const VNumeric *b)
- bool equal (const VNumeric *from) const

  Indicates whether some other VNumeric is equal to this one.
- void **incr** ()
- void invertSign ()

  Inverts the sign of this VNumeric (equivalent to multiplying this VNumeric by -1)
- bool isNeg () const

  Indicates if this VNumeric is negative.
- bool isNull () const

  Indicates if this VNumeric is the SQL NULL value.
- bool isZero () const

  Indicates if this VNumeric is zero.
- void **mul** (const VNumeric *a, const VNumeric *b)
- void setNull ()

  Sets this VNumeric to the SQL NULL value.
- void setZero ()

  Sets this VNumeric to zero.
- void **shiftLeft** (unsigned bitsToShift)
- void **shiftRight** (unsigned bitsToShift)
- void **sub** (const VNumeric *a, const VNumeric *b)
- ifloat toFloat () const

  Convert the VNumeric value into floating-point.
- void **toString** (char *outBuf, int olen) const

Detailed Description

Representation of NUMERIC, fixed point data types in Vertica.

Constructor & Destructor Documentation

Vertica::VNumeric::VNumeric (uint64 * *words*, int32 *precision*, int32 *scale*) [inline]

Create a VNumeric with the provided storage location, precision and scale.

Note

It is the callers responsibility to allocate enough memory for the `words` parameter

Member Function Documentation

int Vertica::VNumeric::compare (const VNumeric * *from*) const [inline]

Compares this (signed) VNumeric to another.

Returns

-1 if this < other, 0 if equal, 1 if this > other

Note

SQL NULL compares less than anything else; two SQL NULLs are considered equal

int Vertica::VNumeric::compareUnsigned (const VNumeric * *from*) const [inline]

Compares this (unsigned) VNumeric to another.

Returns

-1 if this < other, 0 if equal, 1 if this > other

Note

SQL NULL compares less than anything else; two SQL NULLs are considered equal

void Vertica::VNumeric::copy (const VNumeric * *from*) [inline]

Copy data from another VNumeric.

Parameters

from	The source VNumeric

bool Vertica::VNumeric::copy (ifloat *value*, bool *round* = true) [inline]

Copy data from a floating-point number.

Parameters

value	The source floating-point number
round	Truncates if false; otherwise the numeric result is rounded

Returns

false if conversion failed (precision lost or overflow, etc); true otherwise

ifloat Vertica::VNumeric::toFloat ( ) const [inline]

Convert the VNumeric value into floating-point.

Returns

the value in 80-bit floating-point

Vertica::VResources Struct Reference

Public Attributes

- int nFileHandles
- vint scratchMemory

Detailed Description

Representation of the resources user code can ask Vertica for

Member Data Documentation

int Vertica::VResources::nFileHandles

Number of file handles / sockets required

vint Vertica::VResources::scratchMemory

Amount of RAM in bytes used by User defined function

Vertica::VString Class Reference

Representation of a String in Vertica. All character data is internally encoded as UTF-8 characters and is not NULL terminated.

Public Member Functions

- int compare (const VString *other) const

  Compares this VString to another.
- void copy (const char *s, vsize len)

  Copy character data from C string to the VString's internal buffer.
- void copy (const char *s)

  Copy character data from null terminated C string to the VString's internal buffer.
- void copy (const std::string &s)

  Copy character data from std::string.
- void copy (const VString *from)

  Copy data from another VString.
- void copy (const VString &from)

  Copy data from another VString.
- const char * data () const

  Provides a read-only pointer to this VString's internal data.
- char * data ()

  Provides a writeable pointer to this VString's internal data.
- int equal (const VString *other) const

  Indicates whether some other VString is equal to this one.
- bool isNull () const

  Indicates if this VString contains the SQL NULL value.
- vsize length () const

  Returns the length of this VString.
- void setNull ()

Sets this VString to the SQL NULL value.

- std::string str () const

  Provides a copy of this VString's data as a std::string.

Detailed Description

Representation of a String in Vertica. All character data is internally encoded as UTF-8 characters and is not NULL terminated.

Member Function Documentation

int Vertica::VString::compare (const VString * *other*) const [inline]

Compares this VString to another.

Returns

-1 if this < other, 0 if equal, 1 if this > other (just like memcmp)

Note

SQL NULL compares greater than anything else; two SQL NULLs are considered equal

void Vertica::VString::copy (const char * *s*, vsize *len*) [inline]

Copy character data from C string to the VString's internal buffer.

Data is copied from s into this VString. It is the caller's responsibility to not provide a value of 'len' that is larger than the maximum size of this VString

Parameters

s	character input data
len	length in bytes, **not** including the terminating null character

void Vertica::VString::copy (const char * *s*) [inline]

Copy character data from null terminated C string to the VString's internal buffer.

Parameters

s	null-terminated character input data

void Vertica::VString::copy (const std::string & *s*) [inline]

Copy character data from std::string.

Parameters

s	null-terminated character input data

void Vertica::VString::copy (const VString * *from*) [inline]

Copy data from another VString.

Parameters

from	The source VString

void Vertica::VString::copy (const VString & *from*) [inline]

Copy data from another VString.

Parameters

from	The source VString

const char* Vertica::VString::data ( ) const [inline]

Provides a read-only pointer to this VString's internal data.

Returns

the read only character data for this string, as a pointer.

Note

The returned string is **not** null terminated

char* Vertica::VString::data ( ) [inline]

Provides a writeable pointer to this VString's internal data.

Returns

the writeable character data for this string, as a pointer.

Note

The returned string is **not** null terminated

int Vertica::VString::equal (const VString * *other*) const [inline]

Indicates whether some other VString is equal to this one.

Returns

-1 if both are SQL NULL, 0 if not equal, 1 if equal so you can easily consider two NULL values to be equal to each other, or not

bool Vertica::VString::isNull ( ) const [inline]

Indicates if this VString contains the SQL NULL value.

Returns

true if this string contains the SQL NULL value, false otherwise

vsize Vertica::VString::length ( ) const [inline]

Returns the length of this VString.

Returns

the length of the string, in bytes. Does not include any extra space for null characters.

std::string Vertica::VString::str ( ) const [inline]

Provides a copy of this VString's data as a std::string.

Returns

a std::string copy of the data in this VString

Vertica::VTAllocator Class Reference

Public Member Functions

- virtual void * alloc (size_t size)=0

Detailed Description

VTAllocator is a pool based allocator that is provided to simplify memory management for UDF implementors.

Member Function Documentation

virtual void* Vertica::VTAllocator::alloc (size_t *size*) `[pure virtual]`

Allocate size_t bytes of memory on a pool. This memory is guaranteed to persist beyond the destroy call but might have been destroyed when the dtor is run.

File Documentation

Vertica.h File Reference

Contains the classes needed to write User-Defined things in Vertica.

Classes

- class Vertica::AggregateFunction
- class Vertica::AggregateFunctionFactory
- class Vertica::AnalyticFunction
- class Vertica::AnalyticFunctionFactory
- class Vertica::AnalyticPartitionReader
- class Vertica::AnalyticPartitionWriter
- class Vertica::BlockReader

  Iterator interface for reading rows in a Vertica block.
- class Vertica::BlockWriter

  Iterator interface for writing rows to a Vertica block.
- class Vertica::ColumnTypes

  Represents (unsized) types of the columns used as input/output of a User Defined Function/Transform Function.
- struct Vertica::DataBuffer
- class Vertica::DefaultSourceIterator
- class Vertica::FilterFactory
- union Vertica::FIunion
- union Vertica::FIunion
- class Vertica::IntermediateAggs

  : A wrapper around a single intermediate aggregate value
- class Vertica::IterativeSourceFactory
- class Vertica::MultiPhaseTransformFunctionFactory
- class Vertica::MultipleIntermediateAggs

  : A wrapper around multiple intermediate aggregates
- class Vertica::NodeSpecifyingPlanContext
- class Vertica::ParamReader

: A wrapper around Parameters that have a name->value correspondence

- class Vertica::ParamWriter

  Iterator interface for writing rows to a Vertica block.
- class Vertica::ParserFactory
- struct Vertica::PartitionOrderColumnInfo

  Represents the partition by and order by column information for each phase in a multi-phase transform function.
- class Vertica::PartitionReader
- class Vertica::PartitionWriter
- class Vertica::PerColumnParamReader

  : A wrapper around a map from column to ParamReader.
- class Vertica::PlanContext
- struct Vertica::RejectedRecord
- class Vertica::ScalarFunction
- class Vertica::ScalarFunctionFactory
- class Vertica::ServerInterface
- class Vertica::SizedColumnTypes

  Represents types and information to determine the size of the columns as input/output of a User Defined Function/Transform.
- class Vertica::SourceFactory
- class Vertica::SourceIterator
- class Vertica::StreamWriter
- class Vertica::TransformFunction
- class Vertica::TransformFunctionFactory
- class Vertica::TransformFunctionPhase
- class Vertica::UDFilter
- class Vertica::UDLFactory
- class Vertica::UDParser
- class Vertica::UDSource
- class Vertica::UDXFactory
- class Vertica::UDXObject
- class Vertica::UDXObjectCancelable
- struct Vertica::UDxRegistrar
- class Vertica::UnsizedUDSource
- class Vertica::VerticaBlock

  : Represents an in-memory block of tuples
- struct Vertica::VerticaBuildInfo
- class Vertica::VerticaType

  Represents types of data that are passed into and returned back from user's code.
- class Vertica::VInterval

  Representation of an Interval in Vertica.

- class Vertica::VIntervalYM

  Representation of an IntervalYM in Vertica. An Interval can be broken up into years and months.
- class Vertica::VNumeric

  Representation of NUMERIC, fixed point data types in Vertica.
- struct Vertica::VResources
- class Vertica::VString

  Representation of a String in Vertica. All character data is internally encoded as UTF-8 characters and is not NULL terminated.
- class Vertica::VTAllocator

Namespaces

- namespace Vertica

Macros

- #define **__Linux64__**
- #define InlineAggregate()
- #define RegisterFactory(clazz)

Typedefs

- typedef unsigned long long int Vertica::BaseDataOID
- typedef uint8 Vertica::byte

  unsigned 8-bit integer
- typedef int64 Vertica::DateADT
- typedef long double Vertica::ifloat

  vertica 80-bit intermediate result
- typedef signed short Vertica::int16

  signed 16-bit integer
- typedef signed int Vertica::int32

  signed 32-bit integer
- typedef signed long long Vertica::int64

  signed 64-bit integer
- typedef signed char Vertica::int8

  8-bit integer
- typedef int64 Vertica::Interval
- typedef int64 Vertica::IntervalYM
- typedef unsigned long long int **Vertica::Oid**

- typedef int64 Vertica::TimeADT
- typedef int64 Vertica::Timestamp
- typedef int64 Vertica::TimestampTz

  Vertica timestamp data type.
- typedef int64 Vertica::TimeTzADT
- typedef unsigned short Vertica::uint16

  unsigned 16-bit integer
- typedef unsigned int Vertica::uint32

  unsigned 32-bit integer
- typedef unsigned long long Vertica::uint64

  unsigned 64-bit integer
- typedef unsigned char Vertica::uint8

  unsigned 8-bit integer
- typedef uint8 Vertica::vbool

  vertica 8-bit boolean (t,f,null)
- typedef double Vertica::vfloat

  Represents Vertica 64-bit floating-point type for NULL checking use vfloatIsNull(), assignment to NULL use vfloat_null for NaN checking use vfloatIsNaN(), assignment to NaN use vfloat_NaN.
- typedef signed long long Vertica::vint

  vertica 64-bit integer (not int64)
- typedef unsigned long long Vertica::vpos

  64-bit vertica position
- typedef uint32 Vertica::vsize

  vertica 32-bit block size
- typedef void(∗ **Vertica::VT_THROW_EXCEPTION**)(int errcode, const std-::string &message, const char ∗filename, int lineno)

Enumerations

- enum **DTtype** {
  RESERV = 0, **MONTH** = 1, **YEAR** = 2, **DAY** = 3,
  NEG_INTERVAL = 4, **TZ** = 5, **DTZ** = 6, **DTZMOD** = 7,
  IGNORE_DTF = 8, **AMPM** = 9, **HOUR** = 10, **MINUTE** = 11,
  SECOND = 12, **DOY** = 13, **DOW** = 14, **UNITS** = 15,
  ADBC = 16, **AGO** = 17, **ISODATE** = 20, **ISOTIME** = 21,
  UNKNOWN_FIELD = 31 }
- enum Vertica::InputState { **OK**, **END_OF_FILE** }
- enum Vertica::StreamState {
  INPUT_NEEDED, **OUTPUT_NEEDED**, **DONE**, **REJECT**,
  KEEP_GOING }

- enum **strictness** { **DEFAULT_STRICTNESS**, **CALLED_ON_NULL_INPUT**, **RETURN_NULL_ON_NULL_INPUT**, **STRICT** }
- enum Vertica::VBool {
  VFalse = 0, **VTrue** = 1, **VUnknown** = 2, **VFalse** = 0,
  VTrue = 1, **VUnknown** = 2 }

  Enumeration for Boolean data values.
- enum Vertica::VBool {
  VFalse = 0, **VTrue** = 1, **VUnknown** = 2, **VFalse** = 0,
  VTrue = 1, **VUnknown** = 2 }

  Enumeration for Boolean data values.
- enum Vertica::volatility { **DEFAULT_VOLATILITY**, **VOLATILE**, **IMMUTABLE**, **STABLE** }

Functions

- void DateADT **Vertica::dateIn** (const char *str, bool report_error)
- DateADT **Vertica::dateInFormatted** (const char *str, const std::string format, bool report_error)
- static void Vertica::describeIntervalTypeMod (char *result, int32 typemod)
- void **Vertica::dummy** ()
- static DateADT Vertica::getDateFromUnixTime (time_t time)
- static int64 Vertica::getGMTTz (TimeTzADT time)
- static uint64 Vertica::getTime (TimeADT time)
- static TimeADT Vertica::getTimeFromUnixTime (time_t time)
- static Timestamp Vertica::getTimestampFromUnixTime (time_t time)
- static TimestampTz Vertica::getTimestampTzFromUnixTime (time_t time)
- static int64 Vertica::getTimeTz (TimeTzADT time)
- Oid **Vertica::getUDTypeOid** (const char *typeName)
- Oid **Vertica::getUDTypeUnderlyingOid** (Oid typeOid)
- static time_t Vertica::getUnixTimeFromDate (DateADT date)
- static time_t Vertica::getUnixTimeFromTime (TimeADT t)
- static time_t Vertica::getUnixTimeFromTimestamp (Timestamp timestamp)
- static time_t **Vertica::getUnixTimeFromTimestampTz** (TimestampTz timestamp)
- static int32 Vertica::getZoneTz (TimeTzADT time)
- Interval **Vertica::intervalIn** (const char *str, int32 typmod, bool report_error)
- bool **Vertica::isUDType** (Oid typeOid)
- void Vertica::log (const char *format,...) __attribute__((format(printf
- int **Vertica::makeErrMsg** (std::stringstream &err_msg, const char *fmt,...)
- static TimeADT Vertica::setTime (int64 time)
- static TimeTzADT Vertica::setTimeTz (int64 time, int32 zone)
- TimeADT **Vertica::timeIn** (const char *str, int32 typmod, bool report_error)

- TimeADT **Vertica::timeInFormatted** (const char *str, int32 typmod, const std::string format, bool report_error)
- Timestamp **Vertica::timestampIn** (const char *str, int32 typmod, bool report_error)
- Timestamp **Vertica::timestampInFormatted** (const char *str, int32 typmod, const std::string format, bool report_error)
- TimestampTz **Vertica::timestamptzIn** (const char *str, int32 typmod, bool report_error)
- TimestampTz **Vertica::timestamptzInFormatted** (const char *str, int32 typmod, const std::string format, bool report_error)
- TimeTzADT **Vertica::timetzIn** (const char *str, int32 typmod, bool report_error)
- TimeTzADT **Vertica::timetzInFormatted** (const char *str, int32 typmod, const std::string format, bool report_error)
- bool **Vertica::vfloatIsNaN** (const vfloat *f)
- bool **Vertica::vfloatIsNaN** (const vfloat f)
- bool **Vertica::vfloatIsNull** (const vfloat *vf)
- bool **Vertica::vfloatIsNull** (const vfloat vf)
- void **Vertica::vsmemclr** (void *dst, size_t len)
- void Vertica::vsmemcpy (void *dst, const void *src, size_t len)
- bool Vertica::vsmemne (const void *p1, const void *p2, size_t len)

  Version of memcmp for detecting not equal only. Inline function for comparing small things of arbitrary length.

Variables

- const byte Vertica::byte_null = 0xff

  Indicates NULL byte.
- const char Vertica::char_null = 0xff

  Indicates NULL char.
- const vbool Vertica::vbool_false = VFalse

  Value for Boolean FALSE.
- const vbool Vertica::vbool_null = VUnknown

  Value for Boolean NULL.
- const vbool Vertica::vbool_true = VTrue

  Value for Boolean TRUE.
- const vfloat **Vertica::vfloat_null** = vfn.vf
- union Vertica::FIunion **Vertica::vfn** = {0x7fffffffffffffeLL}
- union FIunion **Vertica::vfNaN** = {0xFFF8000000000000LL}
- const vint Vertica::vint_null = 0x8000000000000000LL

  Value for Integer NULLs.
- VT_THROW_EXCEPTION **Vertica::vt_throw_exception**

Detailed Description

Contains the classes needed to write User-Defined things in Vertica.

Macro Definition Documentation

#define InlineAggregate( )

Value:

```
virtual void aggregateArrs(ServerInterface &srvInterface, void **dstTuples,\
                           int doff, const void *arr, int stride, const
      void *rcounts,\
                           int rcstride, int count, IntermediateAggs &
      intAggs,\
                           std::vector<int> &intOffsets, BlockReader &
      arg_reader) {\
        char *arg = const_cast<char*>(static_cast<const char*>(arr));\
        const uint8 *rowCountPtr = static_cast<const uint8*>(rcounts);\
        for (int i=0; i<count; ++i) {\
            vpos rowCount = *reinterpret_cast<const vpos*>(rowCountPtr);\
            char *aggPtr = static_cast<char *>(dstTuples[i]) + doff;\
            updateCols(arg_reader, arg, rowCount, intAggs, aggPtr, intOffsets);
      \
            aggregate(srvInterface, arg_reader, intAggs);\
            arg += rowCount * stride;\
            rowCountPtr += rcstride;\
        }\
    }\
```

InlineAggregate() is used to implement the virtual function "aggregateArrs()" for AggregateFunctions. This allows aggregate functions to work on blocks at a time. This macro should be called from inside an AggregateFunction - for a reference, check the examples in the example folder.

#define RegisterFactory(*clazz*)

Value:

```
clazz clazz##_instance; \
    extern "C" Vertica::UDXFactory *get##clazz() { return &
      clazz##_instance; } \
    Vertica::UDxRegistrar clazz##_registrar(#clazz)
```

Parameters

class	The name of the class to register as a UDx factory. Helper macro for registering UDx's with Vertica

To use: Extend a subclass of UDXFactory (e.g. ScalarFunctionFactory) and then call RegisterFactory with that class name.

For example:

```
... class MyFactory : public ScalarFunctionFactory ...
RegisterFactory(MyFactory);
```