HP Vertica C++ SDK Documentation Version 7.0

Tue Feb 11 2014

Contents

Introduction

Welcome to the HP Vertica C++ SDK Documentation. This documentation covers all of the classes that make up the C++ SDK API. Using this API, you can create User Defined Extensions (UDxs) that integrate your own features into the HP Vertica Analytics Platform.

To learn how UDxs work and how to develop, compile, and deploy them, see the main HP Vertica documentation topic Developing and Using User Defined Functions.

Namespace Index

Namespace List

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

Hierarchical Index

Class Hierarchy

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

Class Index

Class List

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

File Index

File List

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

Namespace Documentation

Basics Namespace Reference

basic utilities mainly for data types.

Classes

- struct BigInt
- struct FiveToScale

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.

Detailed Description

basic utilities mainly for data types.

Vertica Namespace Reference

Classes

- class AggregateFunction
- class AggregateFunctionFactory
- class AnalyticFunction
- class AnalyticFunctionFactory
- class AnalyticPartitionReader
- class AnalyticPartitionWriter
- struct BlockFormatter
- struct BlockFormatterCout

- 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 **IndexedBlockReader**
- class **IndexedBlockWriter**
- class IndexListScalarFunction
- class IntermediateAggs

  : A wrapper around a single intermediate aggregate value
- class IterativeSourceFactory
- struct LibraryRegistrar
- 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 UDChunker
- class UDFileOperator
- class UDFileSystem
- class UDFileSystemFactory
- class UDFilter
- class UDLFactory
- class UDParser

- class UDSource
- class UDXFactory
- class UDXObject
- class UDXObjectCancelable
- struct UDxRegistrar
- class UnsizedUDSource
- class ValueRangeReader

  This class represents the value ranges of the arguments of a UDSF, one range per argument.
- class ValueRangeWriter

  This class represents the output value range of a UDSF.
- 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 VerticaValueRange

  This class represents value ranges used in analyzing the output of UDSFs. A range is expressed as a minimum/maximum value (inclusive) pair.
- 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 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 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 int(∗ **VT_ERRMSG**)(const char ∗fmt,...)
- typedef void(∗ **VT_THROW_EXCEPTION**)(int errcode, const std::string &message, const char ∗filename, int lineno)
- typedef void(∗ **VT_THROW_INTERNAL_EXCEPTION**)(int errcode, const std::string &message, const std::string &info, const char ∗filename, int lineno, const char ∗funcname)

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** = 0, **END_OF_FILE** = 1, **END_OF_CHUNK** = 2 }
- enum StreamState {
  INPUT_NEEDED = 0, **OUTPUT_NEEDED** = 1, **DONE** = 2, **REJECT** = 3,
  KEEP_GOING = 4, **CHUNK_ALIGNED** = 5 }
- enum **strictness** { **DEFAULT_STRICTNESS**, **CALLED_ON_NULL_INPUT**, **RETURN_NULL_ON_NULL_INPUT**, **STRICT** }
- enum UDXDebugLevels { **UDXDebugging_WARNING** = 0x0001, **UDXDebugging_INFO** = 0x0002, **UDXDebugging_BASIC** = 0x0008, **UDXDebugging_ALL** = 0xFFFF }
- enum VBool { **VFalse** = 0, **VTrue** = 1, **VUnknown** = 2 }

  Enumeration for Boolean data values.
- enum volatility { **DEFAULT_VOLATILITY**, **VOLATILE**, **IMMUTABLE**, **STABLE** }

Functions

- 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** ()
- UserLibraryManifest ∗ **get_library_manifest** ()

- void **get_vertica_build_info** (VerticaBuildInfo &vbi)
- 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)
- UserLibraryManifest & **GlobalLibraryManifest** ()
- 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::basic_ostream< char, std::char_traits< char > > &err_msg, const char *fmt,...)
- static TimeADT setTime (int64 time)
- static TimeTzADT setTimeTz (int64 time, int32 zone)
- void **setup_global_function_pointers** (VT_THROW_EXCEPTION throw_exception, VT_THROW_INTERNAL_EXCEPTION throw_internal_exception, VT_ERRMSG server_errmsg, ServerFunctions *fns)
- 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.
- template<typename TRET >
  TRET * **vt_createFuncObject** (VTAllocator *allocator)
- template<typename TRET , typename T1 >
  TRET * **vt_createFuncObject** (VTAllocator *allocator, T1 arg1)
- template<typename TRET , typename T1 , typename T2 >
  TRET * **vt_createFuncObject** (VTAllocator *allocator, T1 arg1, T2 arg2)
- template<typename TRET , typename T1 , typename T2 , typename T3 >
  TRET * **vt_createFuncObject** (VTAllocator *allocator, T1 arg1, T2 arg2, T3 arg3)
- template<typename TRET , typename T1 , typename T2 , typename T3 , typename T4 >
  TRET * **vt_createFuncObject** (VTAllocator *allocator, T1 arg1, T2 arg2, T3 arg3, T4 arg4)
- template<typename TRET , typename T1 , typename T2 , typename T3 , typename T4 , typename T5 >
  TRET * **vt_createFuncObject** (VTAllocator *allocator, T1 arg1, T2 arg2, T3 arg3, T4 arg4, T5 arg5)
- template<typename TRET , typename T1 , typename T2 , typename T3 , typename T4 , typename T5 , typename T6 >
  TRET * **vt_createFuncObject** (VTAllocator *allocator, T1 arg1, T2 arg2, T3 arg3, T4 arg4, T5 arg5, T6 arg6)

- template<typename TRET , typename T1 , typename T2 , typename T3 , typename T4 , typename T5 , typename T6 , typename T7 >
  TRET * **vt_createFuncObject** (VTAllocator *allocator, T1 arg1, T2 arg2, T3 arg3, T4 arg4, T5 arg5, T6 arg6, T7 arg7)
- template<typename TRET , typename T1 , typename T2 , typename T3 , typename T4 , typename T5 , typename T6 , typename T7 , typename T8 >
  TRET * **vt_createFuncObject** (VTAllocator *allocator, T1 arg1, T2 arg2, T3 arg3, T4 arg4, T5 arg5, T6 arg6, T7 arg7, T8 arg8)
- template<typename TRET , typename T1 , typename T2 , typename T3 , typename T4 , typename T5 , typename T6 , typename T7 , typename T8 , typename T9 >
  TRET * **vt_createFuncObject** (VTAllocator *allocator, T1 arg1, T2 arg2, T3 arg3, T4 arg4, T5 arg5, T6 arg6, T7 arg7, T8 arg8, T9 arg9)
- template<typename TRET , typename T1 , typename T2 , typename T3 , typename T4 , typename T5 , typename T6 , typename T7 , typename T8 , typename T9 , typename T10 >
  TRET * **vt_createFuncObject** (VTAllocator *allocator, T1 arg1, T2 arg2, T3 arg3, T4 arg4, T5 arg5, T6 arg6, T7 arg7, T8 arg8, T9 arg9, T10 arg10)

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.
- ServerFunctions * **VERTICA_INTERNAL_fns** = NULL
- const vfloat **vfloat_null** = vfn.vf
- union Vertica::FIunion **vfn** = {0x7fffffffffffffeLL}
- union FIunion **vfNaN** = { static_cast<vint>(0xFFF8000000000000LL) }
- const vint vint_null = 0x8000000000000000LL

  Value for Integer NULLs.
- VT_ERRMSG **vt_server_errmsg** = 0
- VT_THROW_EXCEPTION **vt_throw_exception** = 0
- VT_THROW_INTERNAL_EXCEPTION **vt_throw_internal_exception** = 0

Detailed Description

Defines classes for Vertica User Defined Functions

Typedef Documentation

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
2. 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.

END_OF_CHUNK Reached the end of an input chunk. Applies only to a parser and only when fed by a Chunker, it means the current data block ends on a record boundary. In this state a parser should consume all data in the block before returning from process.

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 last row the Parser consumed 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.

CHUNK_ALIGNED Used by UDChunker only. The chunker has found as many rows as possible in current block, and would like to hand off to parser a data block that is aligned with record boundary (input offset points to the start of the row after last row found in current block).

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

enum Vertica::UDXDebugLevels

Bit mast for communicatnig the available UDx debug logging levels.

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.

Referenced by Vertica::VerticaType::getPrettyPrintStr().

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)

Referenced by getTimeTz().

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 does not need to be 0 as zoneinfo record all those historical change, and historical change may have hour/minute/seconds adjustment More information is at VER-26360)

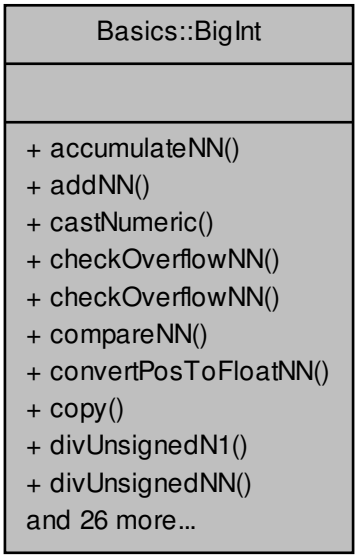
void Vertica::vsmemcpy (void * *dst,* const void * *src,* size_t *len*) `[inline]`

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

Collaboration diagram for Basics::BigInt:



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, int typmod, 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)

Detailed Description

Holds integer utilities A few are inlined for performance reasons. Most are in vertica.cpp.

Member Function Documentation

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

Check integers for equality.

Note

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

Referenced by Vertica::VNumeric::equal().

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

Check an integer for 0.

Note

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

Referenced by Vertica::VNumeric::isZero().

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

Convert Numeric to a float Handles negative / null input.

Parameters

tenthtoscale	10^-scale of the Numeric

Referenced by Vertica::VNumeric::toFloat().

bool Basics::BigInt::setFromFloat (void * *bbuf,* int *bwords,* int *typmod,* long double *value,* bool *round*) `[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

Referenced by Vertica::VNumeric::copy().

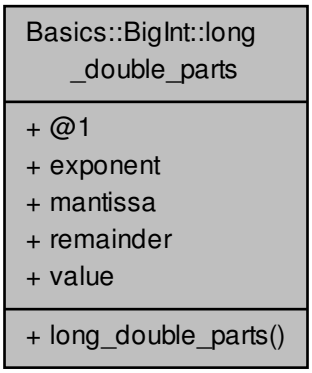
static int Basics::BigInt::ucompareNN (const void * *buf1,* const void * *buf2,* int *nWords*) `[inline]`, `[static]`

Compare integers, return -1, 0, 1

Referenced by Vertica::VNumeric::compareUnsigned().

Basics::BigInt::long_double_parts Union Reference

Collaboration diagram for Basics::BigInt::long_double_parts:



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]
  };

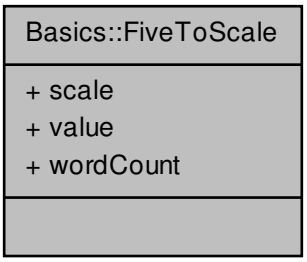
- long double **value**

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

Collaboration diagram for Basics::FiveToScale:

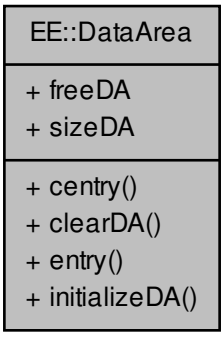


Public Attributes

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

EE::DataArea Class Reference

Collaboration diagram for EE::DataArea:



Public Member Functions

- const char * **centry** (size_t e) const
- void **clearDA** ()
- char * **entry** (size_t e)
- void **initializeDA** (size_t sz)

Public Attributes

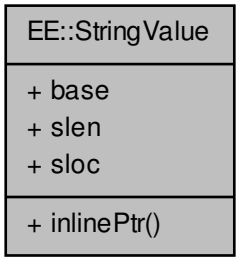
- size_t **freeDA**
- size_t **sizeDA**

Detailed Description

This class holds the strings used by a TupleBlock, VLHash, etc. It is fixed length, applications either use a conservative max or go until it is full.

EE::StringValue Struct Reference

Collaboration diagram for EE::StringValue:



Public Member Functions

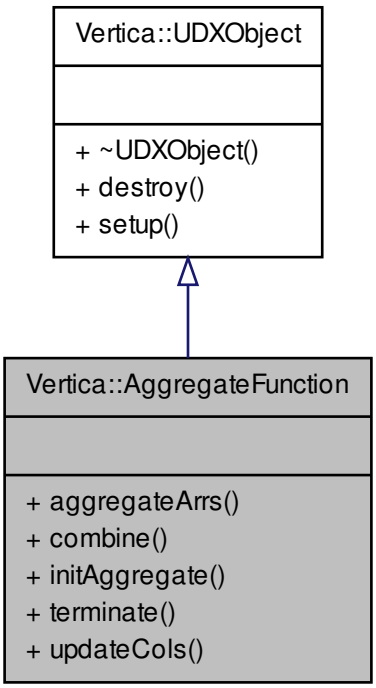
- char * **inlinePtr** ()

Public Attributes

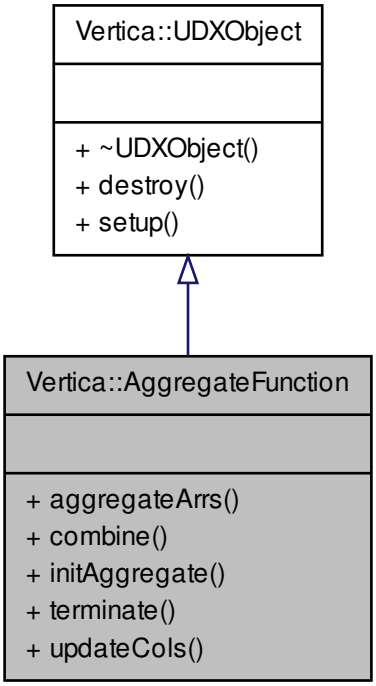
- char **base** [ ]
- vsize **slen**
- uint32 **sloc**

Vertica::AggregateFunction Class Reference

Inheritance diagram for Vertica::AggregateFunction:



Collaboration 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*) `[static]`

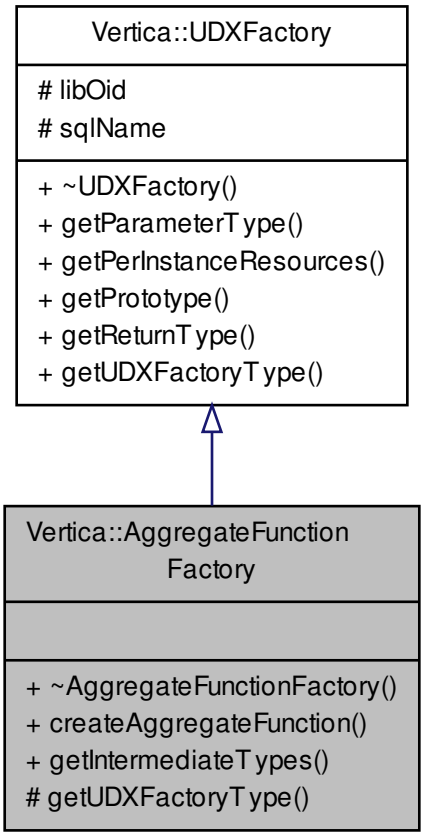
Helper function for aggregateArrs.

Note

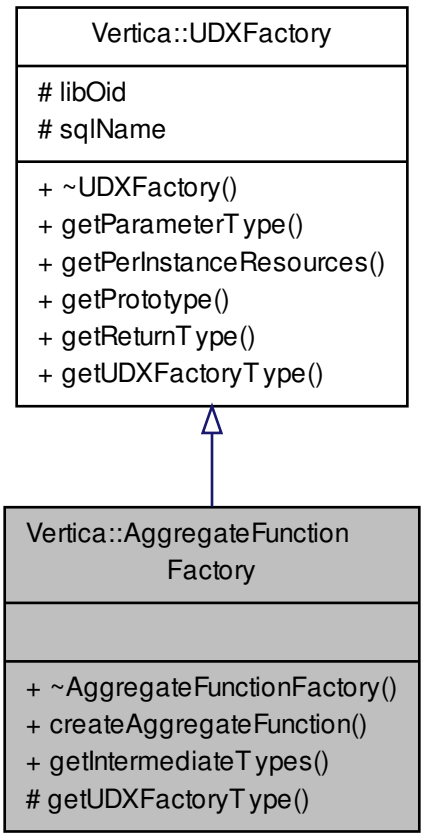
User should not call this function.

Vertica::AggregateFunctionFactory Class Reference

Inheritance diagram for Vertica::AggregateFunctionFactory:



Collaboration diagram for Vertica::AggregateFunctionFactory:



Public Types

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

Reimplemented in Vertica::UDLFactory.

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, Vertica::MultiPhaseTransformFunctionFactory, and Vertica::UDFileSystemFactory.

Referenced by Vertica::ScalarFunctionFactory::getReturnType().

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, Vertica::ScalarFunctionFactory, and Vertica::UDFileSystemFactory.

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

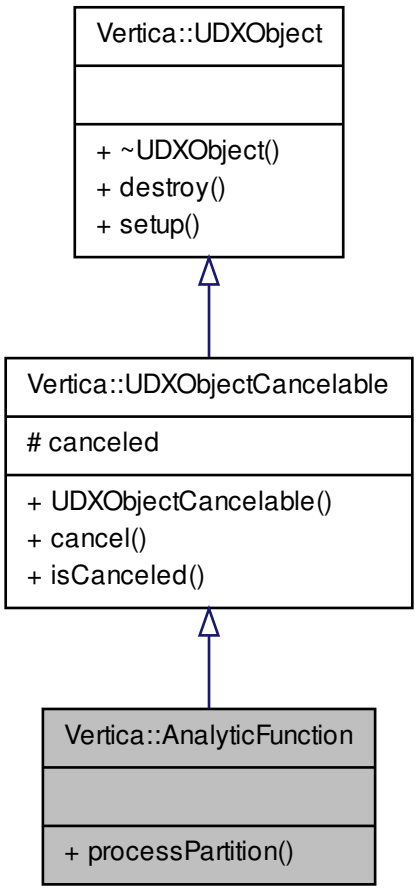
Returns

the object type internally used by Vertica

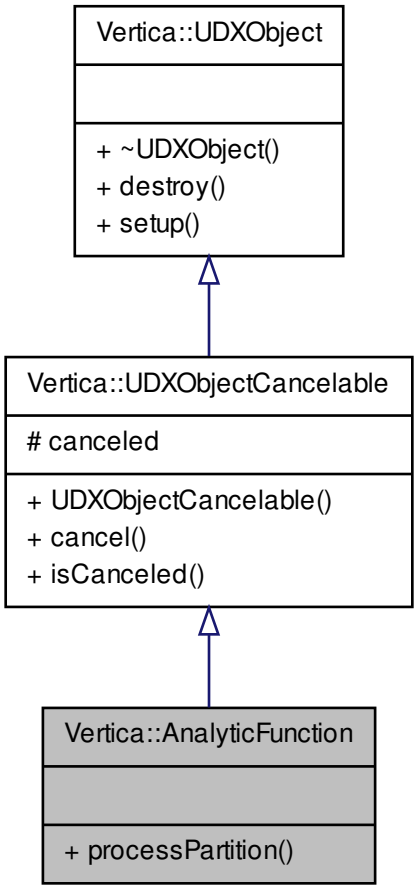
Implements Vertica::UDXFactory.

Vertica::AnalyticFunction Class Reference

Inheritance diagram for Vertica::AnalyticFunction:



Collaboration 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

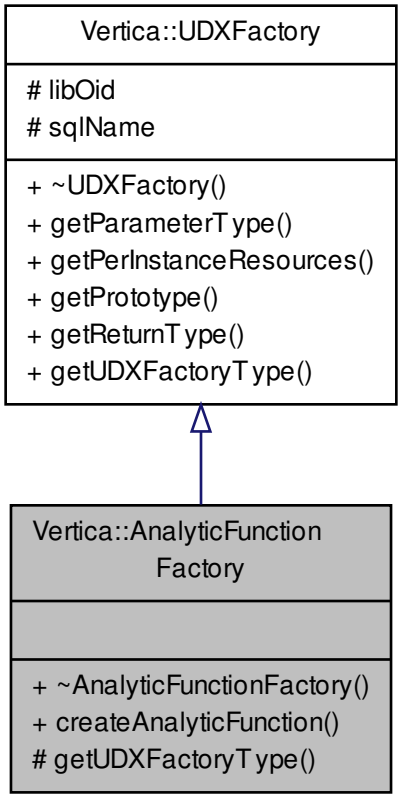
srvInterface	a ServerInterface object used to communicate with Vertica
input_reader	input rows
output_writer	output location

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

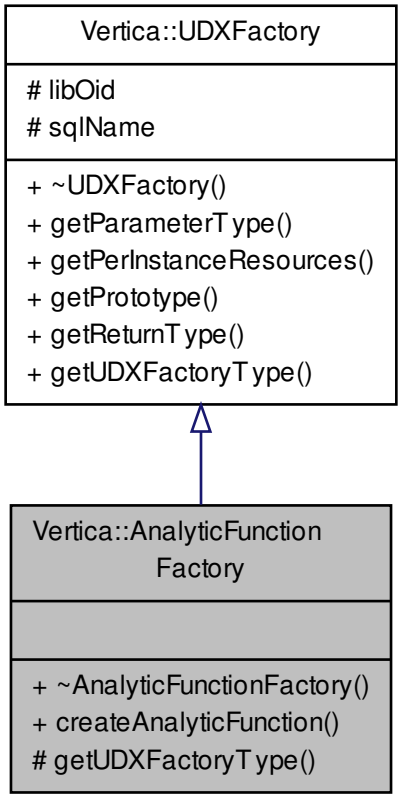
Perform per instance initialization. This function may throw errors.

Vertica::AnalyticFunctionFactory Class Reference

Inheritance diagram for Vertica::AnalyticFunctionFactory:



Collaboration diagram for Vertica::AnalyticFunctionFactory:



Public Types

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

Reimplemented in Vertica::UDLFactory.

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, Vertica::MultiPhaseTransformFunctionFactory, and Vertica::UDFileSystemFactory.

Referenced by Vertica::ScalarFunctionFactory::getReturnType().

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, Vertica::ScalarFunctionFactory, and Vertica::UDFileSystemFactory.

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

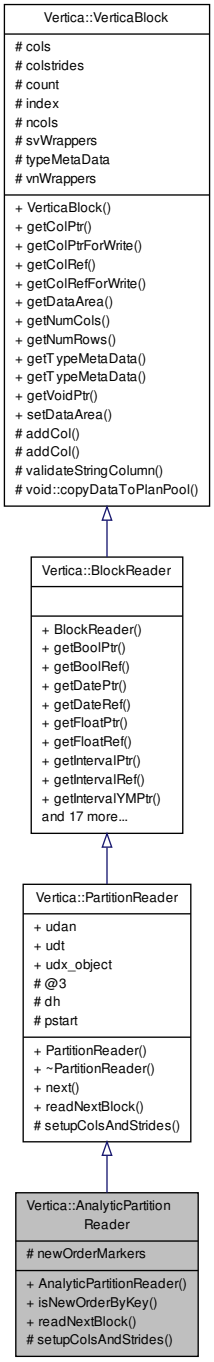
Returns

the object type internally used by Vertica

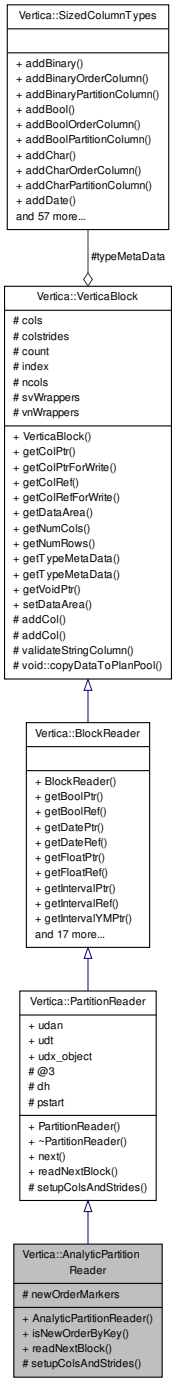
Implements Vertica::UDXFactory.

Vertica::AnalyticPartitionReader Class Reference

Inheritance diagram for Vertica::AnalyticPartitionReader:



Collaboration 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 EE::DataArea * **getDataArea** (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 ()
- void * **getVoidPtr** ()
- bool **isNewOrderByKey** ()
- bool isNull (int col)

  Check if the idx'th argument is null.
- bool **next** ()
- virtual bool readNextBlock ()
- void **setDataArea** (size_t idx, void *dataarea)

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)
- friend **void::copyDataToPlanPool** (VerticaBlock *block)

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

Referenced by Vertica::ParamReader::addParameter().

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.

Referenced by Vertica::BlockReader::getBoolRef().

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.

Referenced by Vertica::BlockReader::isNull().

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);
*
```

Referenced by Vertica::PartitionWriter::copyFromInput().

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.

Referenced by Vertica::BlockReader::getDateRef().

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.

Referenced by Vertica::BlockReader::isNull().

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.

Referenced by Vertica::BlockReader::getFloatRef().

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.

Referenced by Vertica::BlockReader::isNull().

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.

Referenced by Vertica::BlockReader::getIntervalRef().

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.

Referenced by Vertica::BlockReader::isNull().

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.

Referenced by Vertica::BlockReader::getIntervalYMRef().

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.

Referenced by Vertica::BlockReader::isNull().

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);
*
```

Referenced by Vertica::BlockReader::getIntRef().

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);
*
```

Referenced by Vertica::BlockReader::isNull().

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

Returns

the number of columns held by this block.

Referenced by Vertica::BlockReader::isNull().

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.

Referenced by Vertica::BlockReader::getNumericRef().

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.

Referenced by Vertica::BlockReader::isNull().

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.

Referenced by Vertica::PartitionWriter::copyFromInput(), and Vertica::BlockReader::getStringRef().

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.

Referenced by Vertica::BlockReader::isNull().

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.

Referenced by Vertica::BlockReader::getTimeRef().

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.

Referenced by Vertica::BlockReader::isNull().

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.

Referenced by Vertica::BlockReader::getTimestampRef().

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.

Referenced by Vertica::BlockReader::isNull().

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 .

Referenced by Vertica::BlockReader::getTimestampTzRef().

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.

Referenced by Vertica::BlockReader::isNull().

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.

Referenced by Vertica::BlockReader::getTimeTzRef().

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.

Referenced by Vertica::BlockReader::isNull().

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

Returns

information about the types and numbers of arguments

Referenced by Vertica::PartitionWriter::copyFromInput(), Vertica::ParamReader::getType(), Vertica::BlockReader::isNull(), and Vertica::PartitionWriter::setNull().

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.

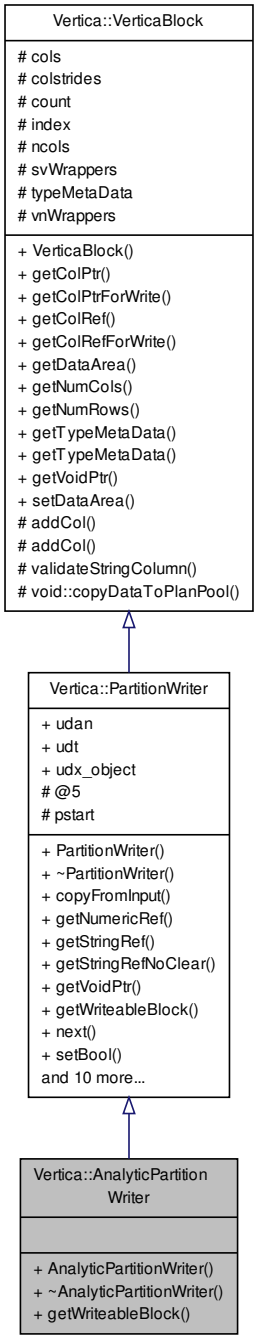
Returns

false if there's no more input data

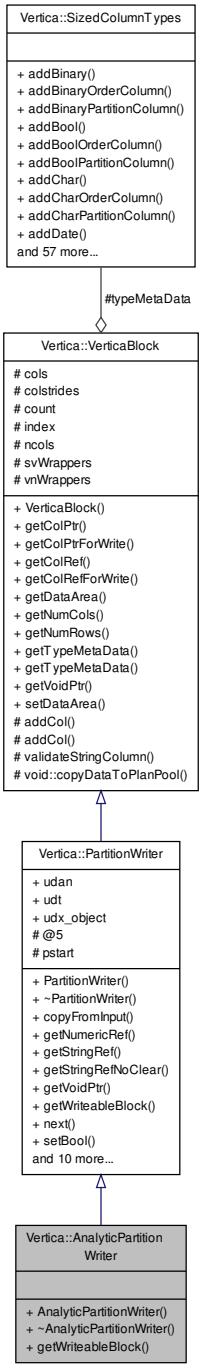
Reimplemented from Vertica::PartitionReader.

Vertica::AnalyticPartitionWriter Class Reference

Inheritance diagram for Vertica::AnalyticPartitionWriter:



Collaboration 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)
- const EE::DataArea * **getDataArea** (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** ()
- void * **getVoidPtr** (size_t idx)
- virtual bool getWriteableBlock ()
- bool **next** ()
- void **setBool** (size_t idx, vbool r)
- void **setDataArea** (size_t idx, void *dataarea)
- 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)
- friend **void::copyDataToPlanPool** (VerticaBlock *block)

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

Referenced by Vertica::ParamReader::addParameter().

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);
*
```

Referenced by Vertica::PartitionWriter::copyFromInput().

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.

Referenced by Vertica::BlockReader::isNull().

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

Referenced by Vertica::PartitionWriter::copyFromInput(), Vertica::ParamReader::getType(), Vertica::BlockReader::isNull(), and Vertica::PartitionWriter::setNull().

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

Referenced by Vertica::PartitionWriter::setNull().

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

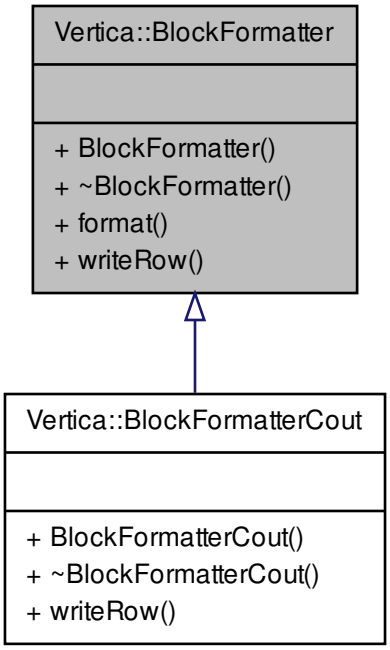
Set the idx'th argument to null.

Parameters

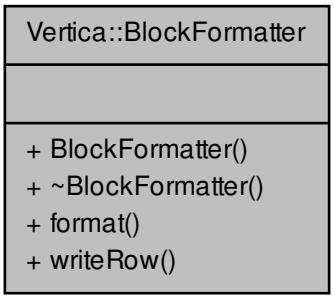
idx	The column number in the row to set to null

Vertica::BlockFormatter Struct Reference

Inheritance diagram for Vertica::BlockFormatter:



Collaboration diagram for Vertica::BlockFormatter:



Public Member Functions

- **BlockFormatter** (const BlockReader &reader)
- void **format** ()

- virtual void **writeRow** (int rowNum, const std::string &row)=0

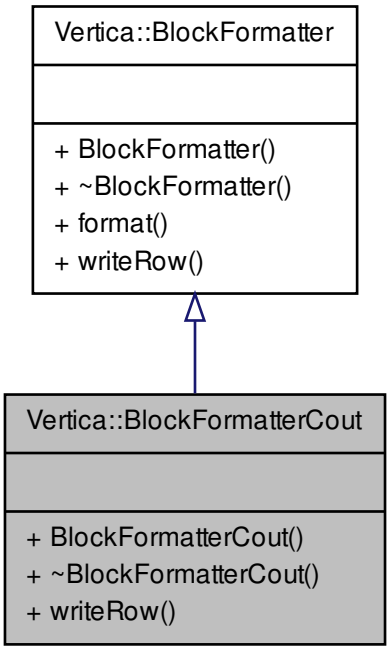
Detailed Description

Class to formatting the data from a UDF arg reader.

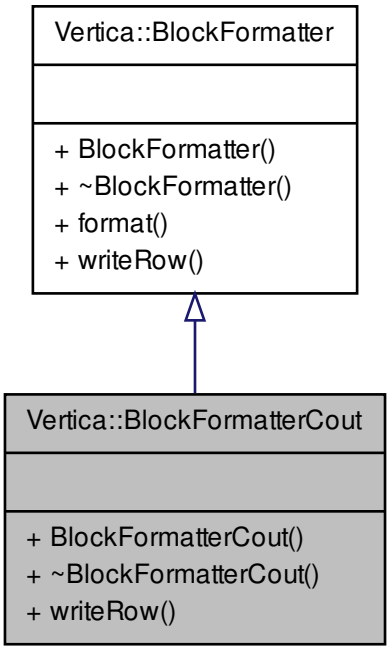
See BlockFormatterCout for example

Vertica::BlockFormatterCout Struct Reference

Inheritance diagram for Vertica::BlockFormatterCout:



Collaboration diagram for Vertica::BlockFormatterCout:



Public Member Functions

- **BlockFormatterCout** (const BlockReader &reader)
- void **format** ()
- void **writeRow** (int rowNum, const std::string &row)

Detailed Description

Formatting rows in a BlockReader to stdout

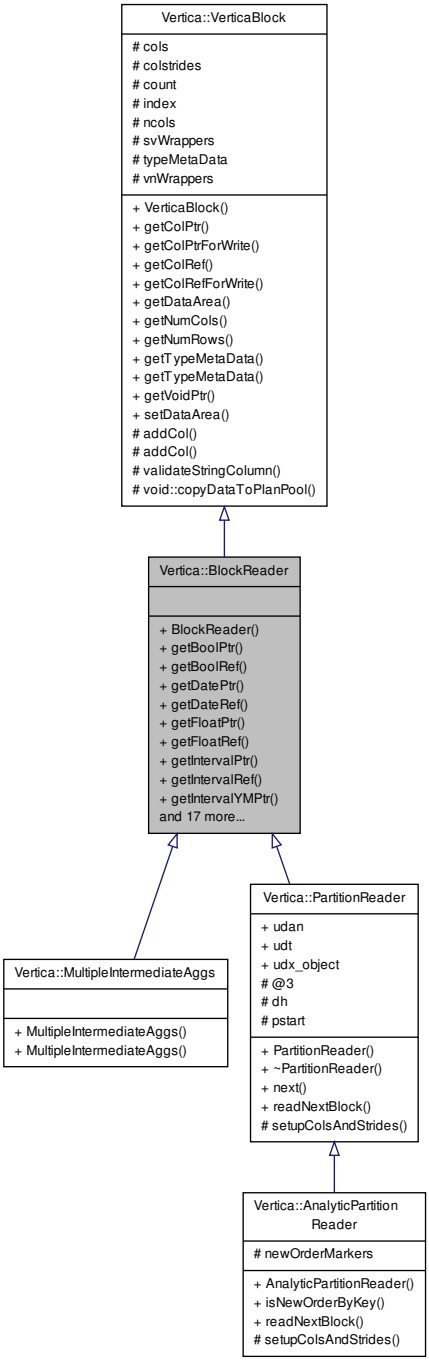
Example Usage with BlockReader ∗arg_reader

BlockFormatterCout(∗arg_reader).format();

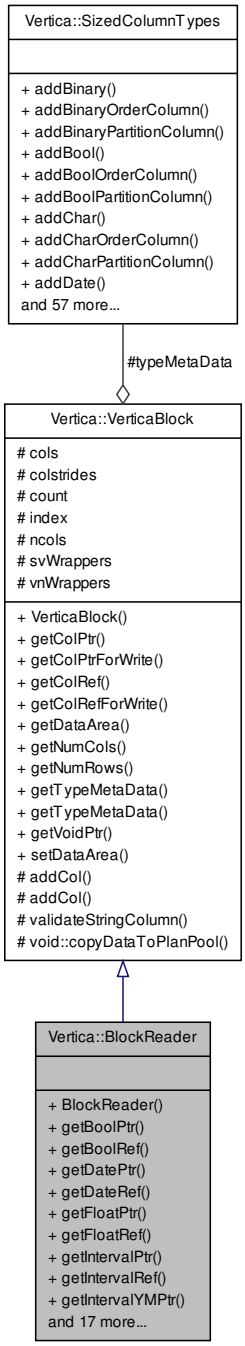
Vertica::BlockReader Class Reference

Iterator interface for reading rows in a Vertica block.

Inheritance diagram for Vertica::BlockReader:



Collaboration 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 EE::DataArea * **getDataArea** (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 ()
- void * **getVoidPtr** ()
- bool isNull (int col)

  Check if the idx'th argument is null.
- bool next ()
- void **setDataArea** (size_t idx, void *dataarea)

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)
- friend **void::copyDataToPlanPool** (VerticaBlock *block)

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

Referenced by Vertica::ParamReader::addParameter().

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.

Referenced by getBoolRef().

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.

Referenced by isNull().

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);
*
```

Referenced by Vertica::PartitionWriter::copyFromInput().

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.

Referenced by getDateRef().

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.

Referenced by isNull().

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.

Referenced by getFloatRef().

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.

Referenced by isNull().

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.

Referenced by getIntervalRef().

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.

Referenced by isNull().

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.

Referenced by getIntervalYMRef().

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.

Referenced by isNull().

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);
*
```

Referenced by getIntRef().

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);
*
```

Referenced by isNull().

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

Returns

the number of columns held by this block.

Referenced by isNull().

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.

Referenced by getNumericRef().

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.

Referenced by isNull().

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.

Referenced by Vertica::PartitionWriter::copyFromInput(), and getStringRef().

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.

Referenced by isNull().

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.

Referenced by getTimeRef().

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.

Referenced by isNull().

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.

Referenced by getTimestampRef().

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.

Referenced by isNull().

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 .

Referenced by getTimestampTzRef().

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.

Referenced by isNull().

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.

Referenced by getTimeTzRef().

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.

Referenced by isNull().

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

Returns

information about the types and numbers of arguments

Referenced by Vertica::PartitionWriter::copyFromInput(), Vertica::ParamReader::getType(), isNull(), and Vertica::-PartitionWriter::setNull().

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.

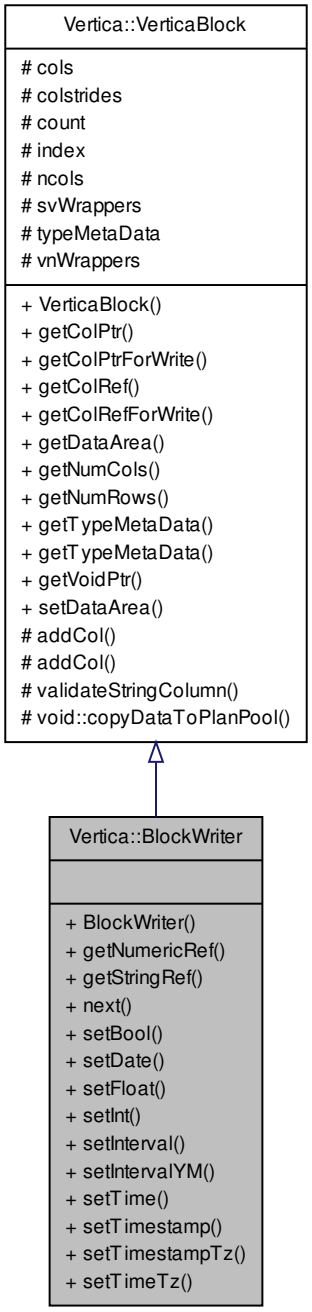
Returns

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

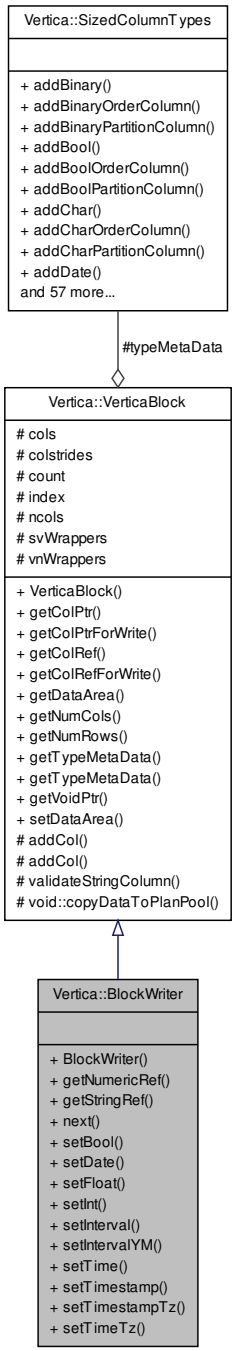
Vertica::BlockWriter Class Reference

Iterator interface for writing rows to a Vertica block.

Inheritance diagram for Vertica::BlockWriter:



Collaboration 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)
- const EE::DataArea * **getDataArea** (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 **setDataArea** (size_t idx, void *dataarea)
- 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)
- friend **void::copyDataToPlanPool** (VerticaBlock *block)

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

Referenced by Vertica::ParamReader::addParameter().

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);
*
```

Referenced by Vertica::PartitionWriter::copyFromInput().

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.

Referenced by Vertica::BlockReader::isNull().

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.

Referenced by next().

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

Returns

information about the types and numbers of arguments

Referenced by Vertica::PartitionWriter::copyFromInput(), Vertica::ParamReader::getType(), Vertica::BlockReader::isNull(), and Vertica::PartitionWriter::setNull().

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.

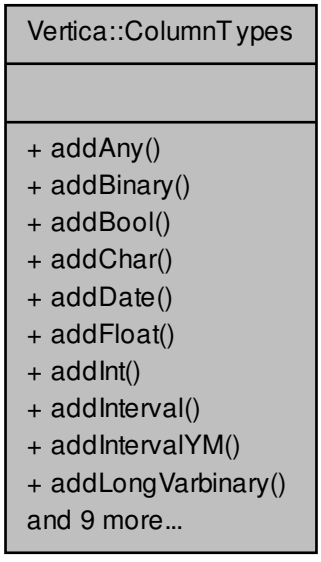
Parameters

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.

Collaboration diagram for Vertica::ColumnTypes:



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.

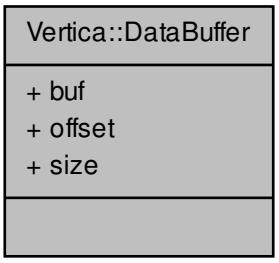
Detailed Description

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

Collaboration diagram for Vertica::DataBuffer:



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.

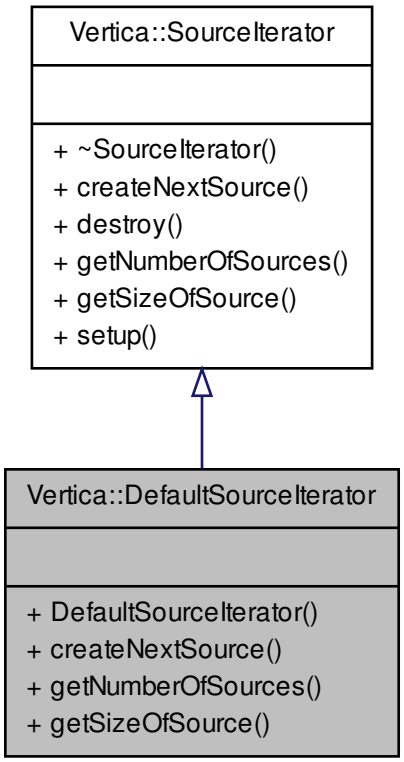
Detailed Description

DataBuffer

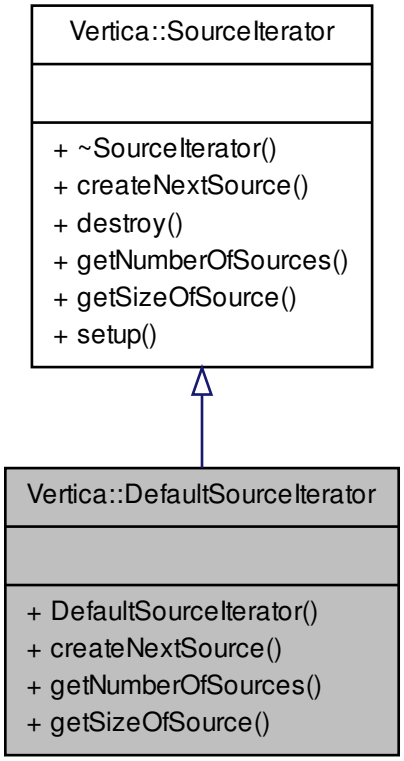
A contiguous in-memory buffer

Vertica::DefaultSourceIterator Class Reference

Inheritance diagram for Vertica::DefaultSourceIterator:



Collaboration diagram for Vertica::DefaultSourceIterator:



Public Member Functions

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

  Create the next UDSource to process.
- virtual void destroy (ServerInterface &srvInterface, NodeSpecifyingPlanContext &planCtxt)

  Tear down this SourceIterator.
- size_t getNumberOfSources ()
- size_t getSizeOfSource (size_t sourceNum)
- virtual void setup (ServerInterface &srvInterface, NodeSpecifyingPlanContext &planCtxt)

  Set up this SourceIterator.

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.

virtual void Vertica::SourceIterator::destroy (ServerInterface & *srvInterface,* NodeSpecifyingPlanContext & *planCtxt*) `[inline],[virtual],[inherited]`

Tear down this SourceIterator.

Should perform clean-up that should not take place in the destructor due to the exception-handling semantics of destructors.

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.

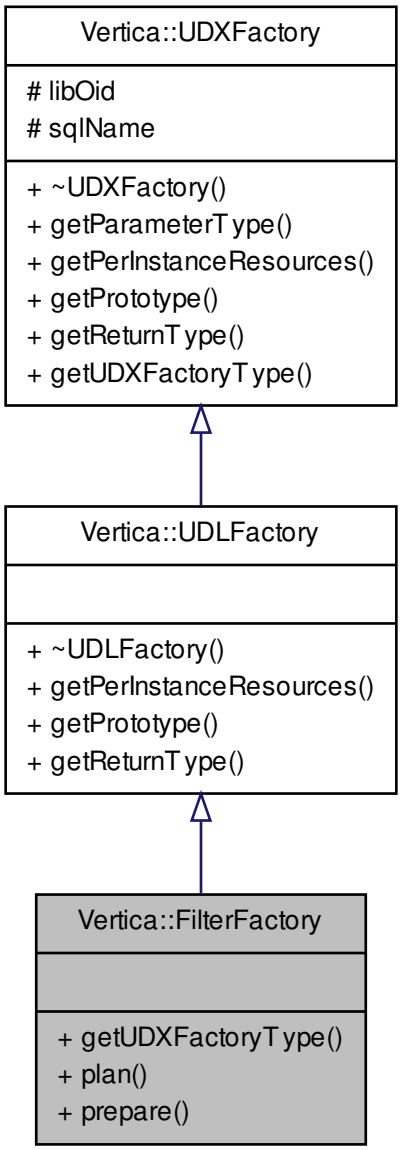
virtual void Vertica::SourceIterator::setup (ServerInterface & *srvInterface,* NodeSpecifyingPlanContext & *planCtxt*) `[inline],[virtual],[inherited]`

Set up this SourceIterator.

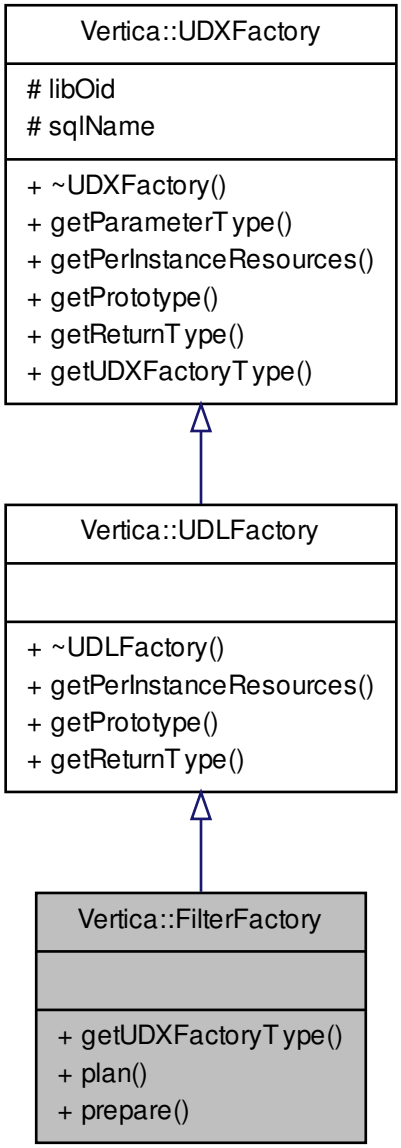
Should perform setup that should not take place in the constructor due to the exception-handling semantics of constructors

Vertica::FilterFactory Class Reference

Inheritance diagram for Vertica::FilterFactory:



Collaboration diagram for Vertica::FilterFactory:



Public Types

- enum UDXType {
  FUNCTION, **TRANSFORM**, **ANALYTIC**, **MULTI_TRANSFORM**,
  AGGREGATE, **LOAD_SOURCE**, **LOAD_FILTER**, **LOAD_PARSER**,
  FILESYSTEM, **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::UDLFactory::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

Reimplemented from Vertica::UDXFactory.

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

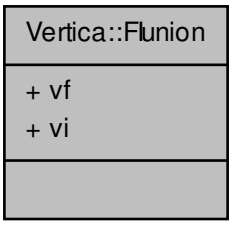
srvInterface	Interface to server operations and functionality, including (not-per-column) parameter lookup
planCtxt	Context for storing and retrieving arbitrary data, for use just by this instance of this query. The same context is shared with plan().

Returns

UDFilter instance to use for this query

Vertica::Flunion Union Reference

Collaboration diagram for Vertica::Flunion:

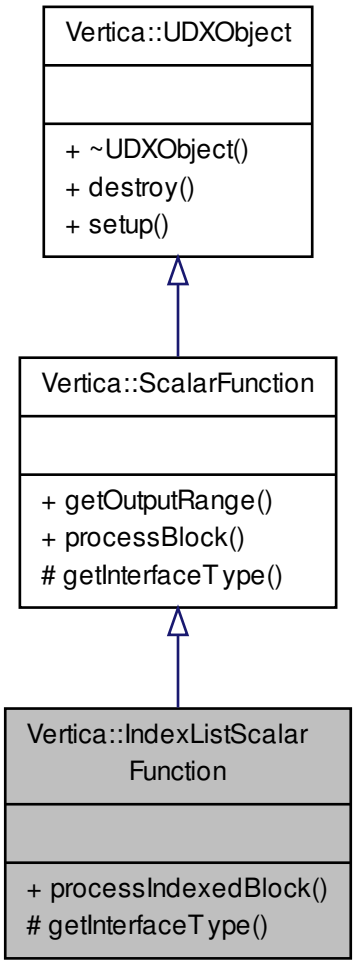


Public Attributes

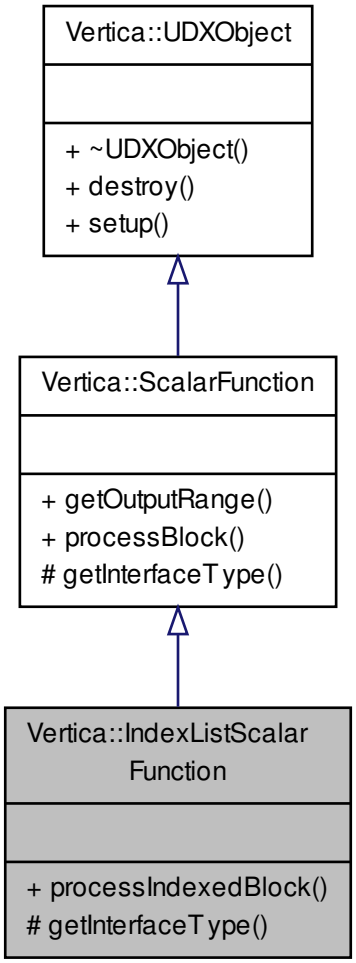
- vfloat **vf**
- vint **vi**

Vertica::IndexListScalarFunction Class Reference

Inheritance diagram for Vertica::IndexListScalarFunction:



Collaboration diagram for Vertica::IndexListScalarFunction:



Public Member Functions

- virtual void destroy (ServerInterface &srvInterface, const SizedColumnTypes &argTypes)
- virtual void getOutputRange (ServerInterface &srvInterface, ValueRangeReader &inRange, ValueRangeWriter &outRange)
- virtual void processBlock (ServerInterface &srvInterface, BlockReader &arg_reader, BlockWriter &res_writer)=0
- virtual void **processIndexedBlock** (ServerInterface &srvInterface, IndexedBlockReader &arg_reader, IndexedBlockWriter &res_writer)=0
- virtual void setup (ServerInterface &srvInterface, const SizedColumnTypes &argTypes)

Protected Types

- enum **InterfaceType** { **FunctionT**, **IndexListFunctionT** }

Protected Member Functions

- virtual InterfaceType **getInterfaceType** ()

Detailed Description

Specialization of ScalarFunction to allow the user to override the function that only operates on a subset of rows in a block.

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::getOutputRange (ServerInterface & *srvInterface,* ValueRangeReader & *inRange,* ValueRangeWriter & *outRange*) `[inline],[virtual],[inherited]`

Invoke a user defined function to determine the output value range of this function. Ranges are represented by a minimum/maximum pair of values (inclusive). The developer is responsible to provide an output value range on the basis of the input argument ranges. Minimum/maximum values of ranges are of the same type as defined in the metadata class getPrototype() function.

Parameters

srvInterface	a ServerInterface object used to communicate with Vertica
inRange	input value range
outRange	output value range

Remarks

By default, the ValueRangeWriter object can have NULL values, values in the range are unsorted, and it is unbounded, i.e., the following functions return as follows:

- outRange.canHaveNulls() == true
- outRange.getSortedness() == EE::SORT_UNORDERED
- outRange.isBounded() == false

Note

- This methods may be invoked by different threads at different times, and by a different thread than the constructor.
- C++ exceptions may NOT be thrown out of this method. Use the vertica specific vt_throw_exception() function or vt_report_error() macro instead
- Invoking vt_throw_exception() or vt_report_error() from this method will not stop the function execution, which may still complete successfully. Instead, the output range will be discarded, and a WARNING message will be written to the Vertica log

virtual void Vertica::ScalarFunction::processBlock (ServerInterface & *srvInterface,* BlockReader & *arg_reader,* BlockWriter & *res_writer*) `[pure virtual],[inherited]`

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

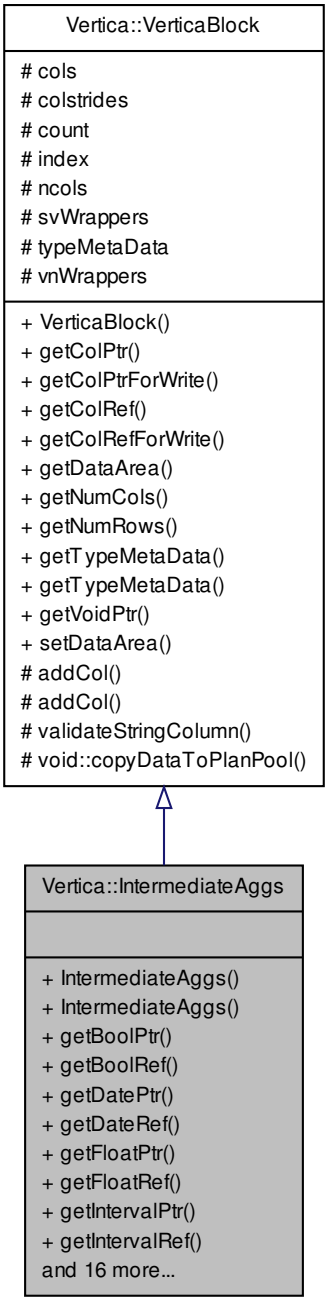
virtual void Vertica::UDXObject::setup (ServerInterface & *srvInterface,* const SizedColumnTypes & *argTypes*) `[inline],[virtual],[inherited]`

Perform per instance initialization. This function may throw errors.

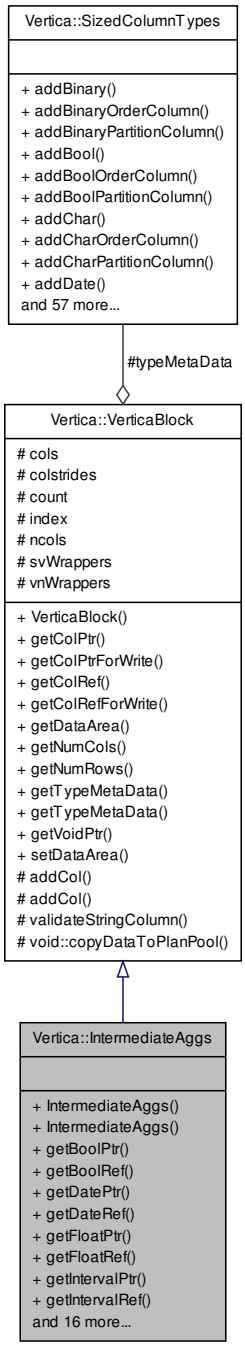
Vertica::IntermediateAggs Class Reference

: A wrapper around a single intermediate aggregate value

Inheritance diagram for Vertica::IntermediateAggs:



Collaboration 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 intermediate results set.
- vbool & getBoolRef (size_t idx)

  Get a reference to a BOOLEAN value from the intermediate results set.

- 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 EE::DataArea * **getDataArea** (size_t idx)
- DateADT * getDatePtr (size_t idx)

  Get a pointer to a DATE value from the intermediate results set.
- DateADT & getDateRef (size_t idx)

  Get a reference to a DATE value from the intermediate results set.
- vfloat * getFloatPtr (size_t idx)

  Get a pointer to a FLOAT value from the intermediate results set.
- vfloat & getFloatRef (size_t idx)

  Get a reference to a FLOAT value from the intermediate results set.
- Interval * getIntervalPtr (size_t idx)

  Get a pointer to an INTERVAL value from the intermediate results set.
- Interval & getIntervalRef (size_t idx)

  Get a reference to an INTERVAL value from the intermediate results set.
- IntervalYM * getIntervalYMPtr (size_t idx)

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

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

  Get a pointer to an INTEGER value from the intermediate results set.
- vint & getIntRef (size_t idx)

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

  Get a pointer to a VNumeric value from the intermediate results set.
- VNumeric & getNumericRef (size_t idx)

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

  Get a pointer to a VString value from the intermediate results set.
- VString & getStringRef (size_t idx)

  Get a reference to an VString value from the intermediate results set.
- TimeADT * getTimePtr (size_t idx)

  Get a pointer to a TIME value from the intermediate results set.
- TimeADT & getTimeRef (size_t idx)

  Get a reference to a TIME value from the intermediate results set.
- Timestamp * getTimestampPtr (size_t idx)

  Get a pointer to a TIMESTAMP value from the intermediate results set.
- Timestamp & getTimestampRef (size_t idx)

  Get a reference to a TIMESTAMP value from the intermediate results set.
- TimestampTz * getTimestampTzPtr (size_t idx)

  Get a pointer to a TIMESTAMP WITH TIMEZONE value from the intermediate results set.
- TimestampTz & getTimestampTzRef (size_t idx)

  Get a reference to a TIMESTAMP WITH TIMEZONE value from the intermediate results set.
- TimeTzADT * getTimeTzPtr (size_t idx)

Get a pointer to a TIME WITH TIMEZONE value from the intermediate results set.

- TimeTzADT & getTimeTzRef (size_t idx)

  Get a reference to a TIME WITH TIMEZONE value from the intermediate results set.
- const SizedColumnTypes & getTypeMetaData () const
- SizedColumnTypes & getTypeMetaData ()
- void * **getVoidPtr** ()
- void **setDataArea** (size_t idx, void *dataarea)

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)
- friend **void::copyDataToPlanPool** (VerticaBlock *block)

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

Referenced by Vertica::ParamReader::addParameter().

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

Get a pointer to a BOOLEAN value from the intermediate results set.

Parameters

idx	The column number in the intermediate results set to retrieve.

Returns

A pointer to the retrieved value cast as a BOOLEAN.

Referenced by getBoolRef().

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

Get a reference to a BOOLEAN value from the intermediate results set.

Parameters

idx	The column number to retrieve from the intermediate results set.

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);
*
```

Referenced by Vertica::PartitionWriter::copyFromInput().

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 intermediate results set.

Parameters

idx	The column number in the intermediate results set to retrieve.

Returns

A pointer to the retrieved value cast as a DATE.

Referenced by getDateRef().

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

Get a reference to a DATE value from the intermediate results set.

Parameters

idx	The column number to retrieve from the intermediate results set.

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 intermediate results set.

Parameters

idx	The column number in the intermediate results set to retrieve.

Returns

A pointer to the retrieved value cast as a FLOAT.

Referenced by getFloatRef().

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

Get a reference to a FLOAT value from the intermediate results set.

Parameters

idx	The column number to retrieve from the intermediate results set.

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 intermediate results set.

Parameters

idx	The column number in the intermediate results set to retrieve.

Returns

A pointer to the retrieved value cast as an INTERVAL.

Referenced by getIntervalRef().

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

Get a reference to an INTERVAL value from the intermediate results set.

Parameters

idx	The column number to retrieve from the intermediate results set.

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 intermediate results set.

Parameters

idx	The column number in the intermediate results set to retrieve.

Returns

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

Referenced by getIntervalYMRef().

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

Get a reference to an INTERVAL YEAR TO MONTH value from the intermediate results set.

Parameters

idx	The column number to retrieve from the intermediate results set.

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 intermediate results set.

Returns

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

Parameters

idx	The column number to retrieve from the intermediate results set.

Example:

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

Referenced by getIntRef().

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

Get a reference to an INTEGER value from the intermediate results set.

Parameters

idx	The column number to retrieve from the intermediate results set.

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.

Referenced by Vertica::BlockReader::isNull().

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

Get a pointer to a VNumeric value from the intermediate results set.

Parameters

idx	The column number to retrieve from the intermediate results set.

Returns

A pointer to the retrieved value cast as a Numeric.

Referenced by getNumericRef().

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

Get a reference to a VNumeric value from the intermediate results set.

Parameters

idx	The column number to retrieve from the intermediate results set.

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 intermediate results set.

Parameters

idx	The column number to retrieve from the intermediate results set.

Returns

A pointer to the retrieved value cast as a VString.

Referenced by getStringRef().

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

Get a reference to an VString value from the intermediate results set.

Parameters

idx	The column number to retrieve from the intermediate results set.

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 intermediate results set.

Parameters

idx	The column number in the intermediate results set to retrieve.

Returns

A pointer to the retrieved value cast as a TIME.

Referenced by getTimeRef().

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

Get a reference to a TIME value from the intermediate results set.

Parameters

idx	The column number to retrieve from the intermediate results set.

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 intermediate results set.

Parameters

idx	The column number in the intermediate results set to retrieve.

Returns

A pointer to the retrieved value cast as a TIMESTAMP.

Referenced by getTimestampRef().

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

Get a reference to a TIMESTAMP value from the intermediate results set.

Parameters

idx	The column number to retrieve from the intermediate results set.

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 intermediate results set.

Parameters

idx	The column number in the intermediate results set to retrieve.

Returns

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

Referenced by getTimestampTzRef().

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

Get a reference to a TIMESTAMP WITH TIMEZONE value from the intermediate results set.

Parameters

idx	The column number to retrieve from the intermediate results set.

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 intermediate results set.

Parameters

idx	The column number in the intermediate results set to retrieve.

Returns

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

Referenced by getTimeTzRef().

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

Get a reference to a TIME WITH TIMEZONE value from the intermediate results set.

Parameters

idx	The column number to retrieve from the intermediate results set.

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

Referenced by Vertica::PartitionWriter::copyFromInput(), Vertica::ParamReader::getType(), Vertica::BlockReader-::isNull(), and Vertica::PartitionWriter::setNull().

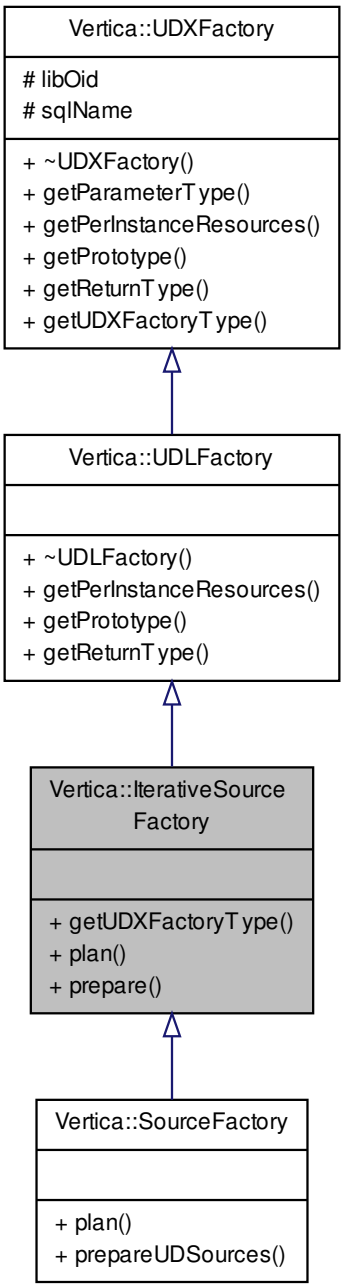
SizedColumnTypes& Vertica::VerticaBlock::getTypeMetaData ( ) `[inline],[inherited]`

Returns

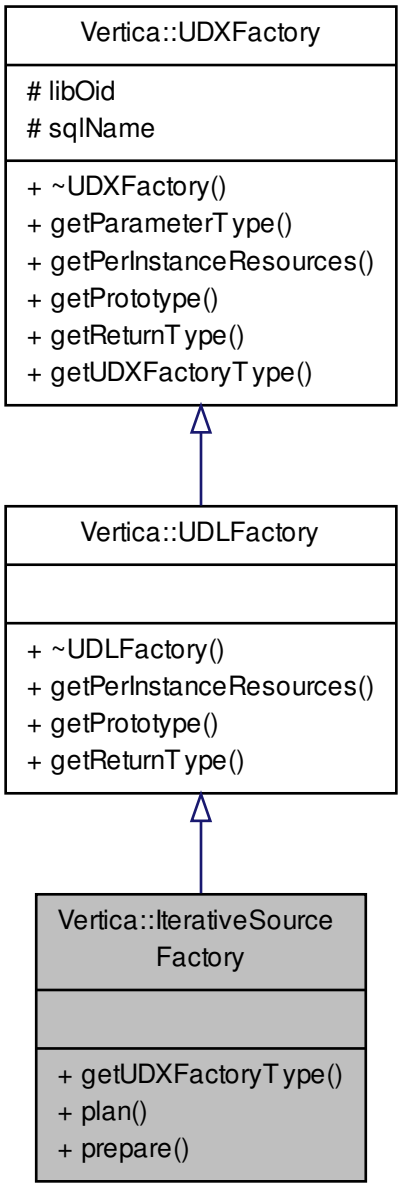
information about the types and numbers of arguments

Vertica::IterativeSourceFactory Class Reference

Inheritance diagram for Vertica::IterativeSourceFactory:



Collaboration diagram for Vertica::IterativeSourceFactory:



Public Types

- enum UDXType {
  FUNCTION, **TRANSFORM**, **ANALYTIC**, **MULTI_TRANSFORM**,
  AGGREGATE, **LOAD_SOURCE**, **LOAD_FILTER**, **LOAD_PARSER**,
  FILESYSTEM, **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::UDLFactory::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

Reimplemented from Vertica::UDXFactory.

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]

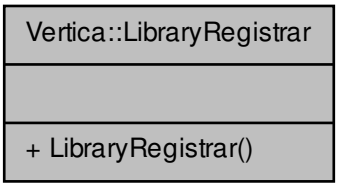
Prepare this SourceFactory to start creating sources. This function will be called on each node, prior to the Load operator starting to execute and prior to any other virtual functions on this class being called.

'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::LibraryRegistrar Struct Reference

Collaboration diagram for Vertica::LibraryRegistrar:

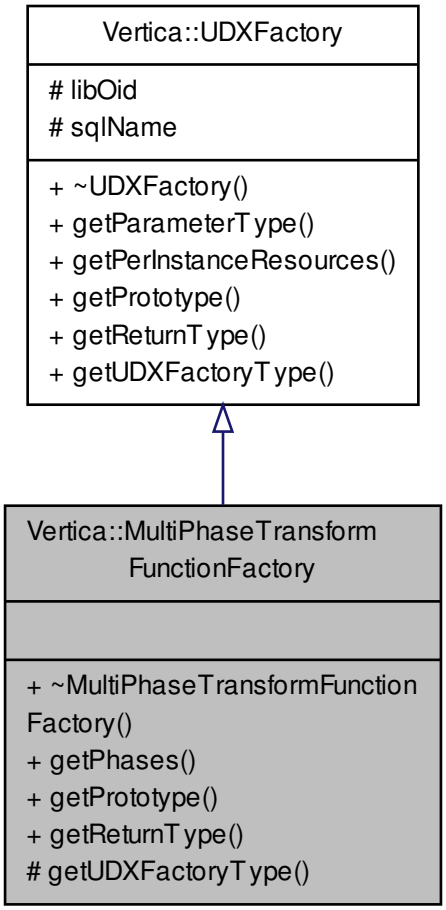


Public Member Functions

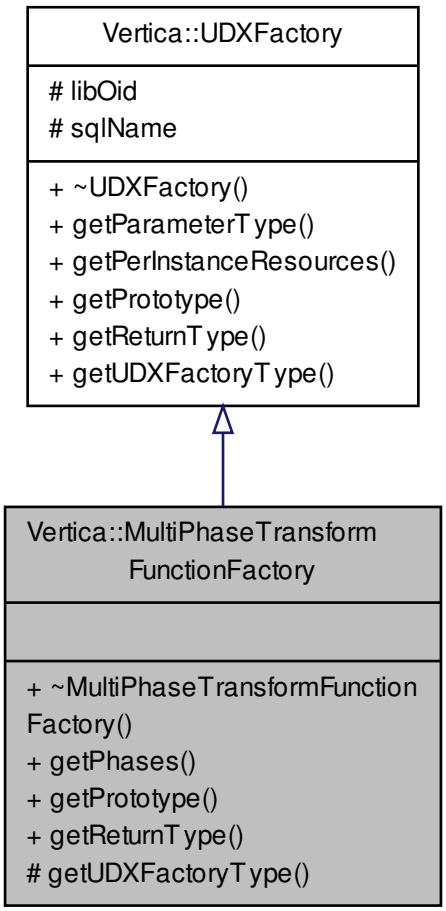
- **LibraryRegistrar** (const char ∗author, const char ∗library_build_tag, const char ∗library_version, const char ∗library_sdk_version, const char ∗source_url, const char ∗description, const char ∗licenses_required, const char ∗signature)

Vertica::MultiPhaseTransformFunctionFactory Class Reference

Inheritance diagram for Vertica::MultiPhaseTransformFunctionFactory:



Collaboration diagram for Vertica::MultiPhaseTransformFunctionFactory:



Public Types

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

Reimplemented in Vertica::UDLFactory.

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

Referenced by getReturnType().

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

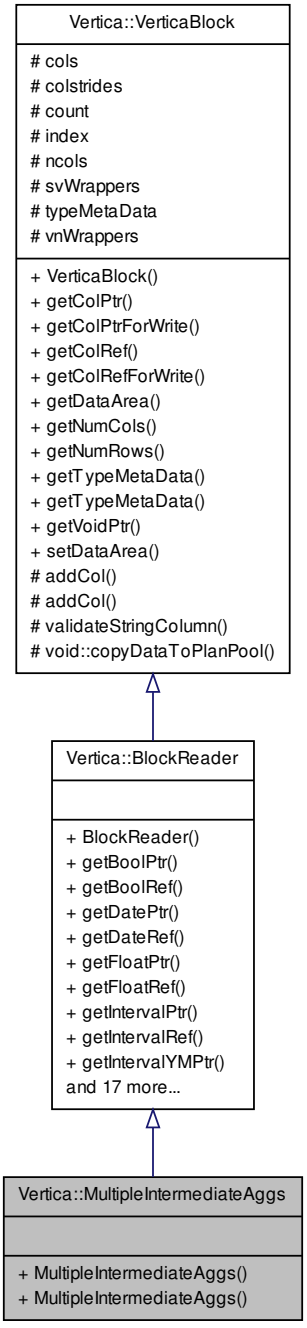
the object type internally used by Vertica

Implements Vertica::UDXFactory.

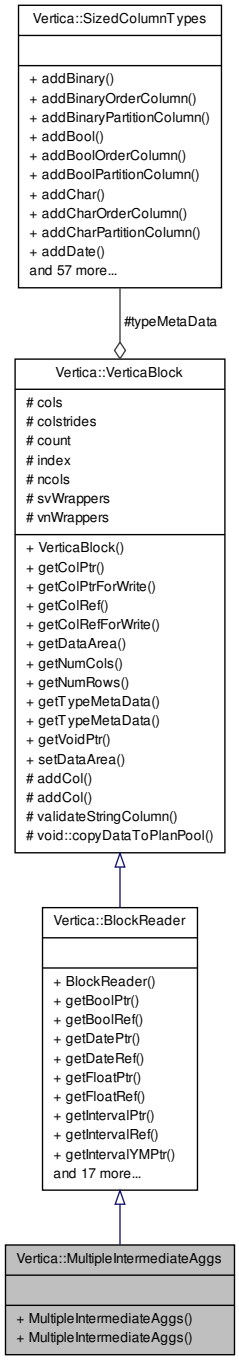
Vertica::MultipleIntermediateAggs Class Reference

: A wrapper around multiple intermediate aggregates

Inheritance diagram for Vertica::MultipleIntermediateAggs:



Collaboration 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 EE::DataArea * **getDataArea** (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 ()
- void * **getVoidPtr** ()
- bool isNull (int col)

  Check if the idx'th argument is null.
- bool next ()
- void **setDataArea** (size_t idx, void *dataarea)

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)
- friend **void::copyDataToPlanPool** (VerticaBlock *block)

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

Referenced by Vertica::ParamReader::addParameter().

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.

Referenced by Vertica::BlockReader::getBoolRef().

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.

Referenced by Vertica::BlockReader::isNull().

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);
*
```

Referenced by Vertica::PartitionWriter::copyFromInput().

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.

Referenced by Vertica::BlockReader::getDateRef().

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.

Referenced by Vertica::BlockReader::isNull().

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.

Referenced by Vertica::BlockReader::getFloatRef().

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.

Referenced by Vertica::BlockReader::isNull().

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.

Referenced by Vertica::BlockReader::getIntervalRef().

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.

Referenced by Vertica::BlockReader::isNull().

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.

Referenced by Vertica::BlockReader::getIntervalYMRef().

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.

Referenced by Vertica::BlockReader::isNull().

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);
*
```

Referenced by Vertica::BlockReader::getIntRef().

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);
*
```

Referenced by Vertica::BlockReader::isNull().

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

Returns

the number of columns held by this block.

Referenced by Vertica::BlockReader::isNull().

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.

Referenced by Vertica::BlockReader::getNumericRef().

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.

Referenced by Vertica::BlockReader::isNull().

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.

Referenced by Vertica::PartitionWriter::copyFromInput(), and Vertica::BlockReader::getStringRef().

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.

Referenced by Vertica::BlockReader::isNull().

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.

Referenced by Vertica::BlockReader::getTimeRef().

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.

Referenced by Vertica::BlockReader::isNull().

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.

Referenced by Vertica::BlockReader::getTimestampRef().

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.

Referenced by Vertica::BlockReader::isNull().

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 .

Referenced by Vertica::BlockReader::getTimestampTzRef().

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.

Referenced by Vertica::BlockReader::isNull().

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.

Referenced by Vertica::BlockReader::getTimeTzRef().

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.

Referenced by Vertica::BlockReader::isNull().

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

Returns

information about the types and numbers of arguments

Referenced by Vertica::PartitionWriter::copyFromInput(), Vertica::ParamReader::getType(), Vertica::BlockReader-::isNull(), and Vertica::PartitionWriter::setNull().

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]`

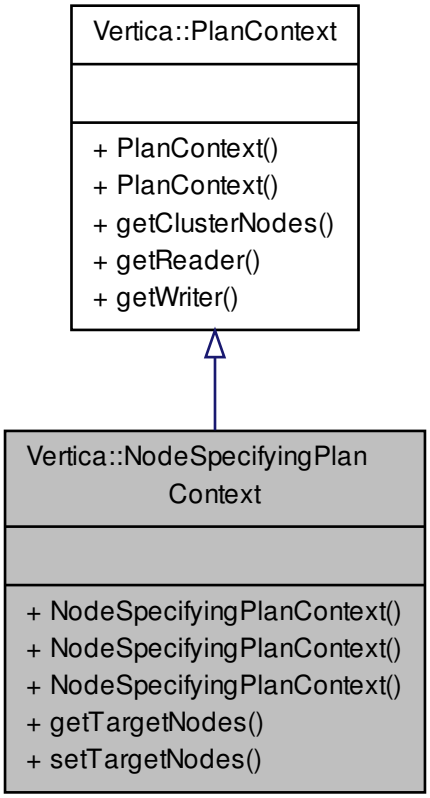
Advance to the next record.

Returns

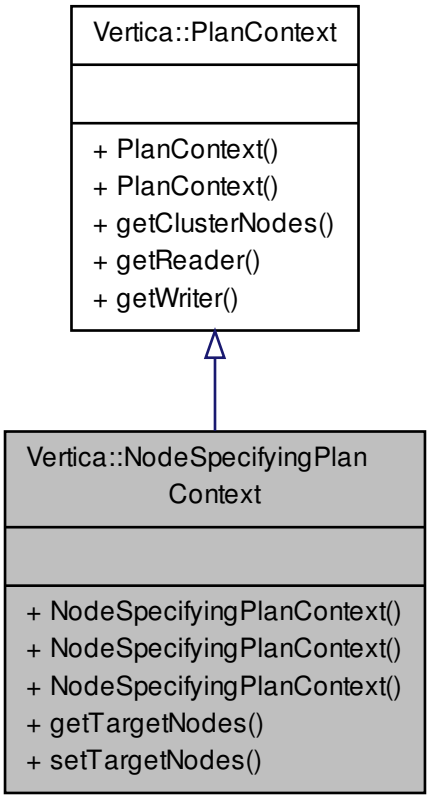
true if there are more rows to read, false otherwise.

Vertica::NodeSpecifyingPlanContext Class Reference

Inheritance diagram for Vertica::NodeSpecifyingPlanContext:



Collaboration 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

Referenced by setTargetNodes().

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

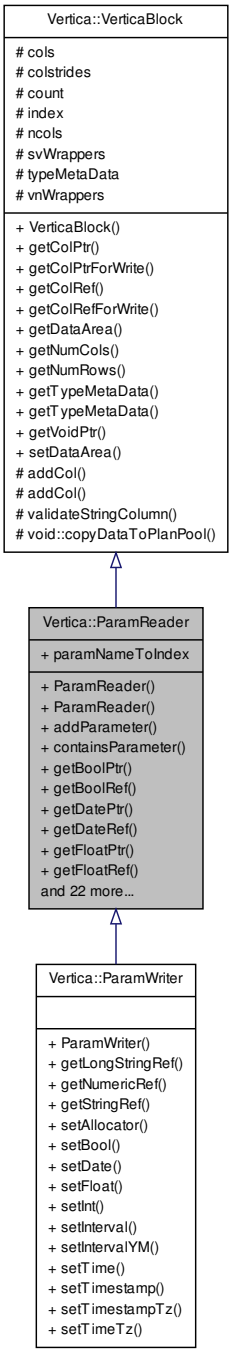
void Vertica::NodeSpecifyingPlanContext::setTargetNodes (const std::vector< std::string > & *nodes*) `[inline]`

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.

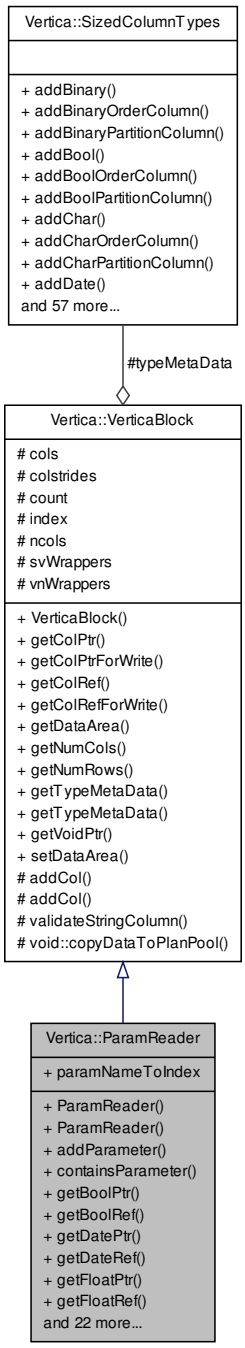
Vertica::ParamReader Class Reference

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

Inheritance diagram for Vertica::ParamReader:



Collaboration 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 EE::DataArea * **getDataArea** (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 ()
- void * **getVoidPtr** ()
- bool isEmpty () const

  Returns true if there are no parameters.

- void **setDataArea** (size_t idx, void *dataarea)

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)
- friend **void::copyDataToPlanPool** (VerticaBlock *block)

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

Referenced by addParameter().

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.

Referenced by getBoolRef().

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);
*
```

Referenced by Vertica::PartitionWriter::copyFromInput().

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.

Referenced by getDateRef().

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.

Referenced by getFloatRef().

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.

Referenced by getIntervalRef().

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.

Referenced by getIntervalYMRef().

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");
*
```

Referenced by getIntRef().

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.

Referenced by Vertica::BlockReader::isNull().

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.

Referenced by getNumericRef().

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.

Referenced by getStringRef().

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.

Referenced by getTimeRef().

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.

Referenced by getTimestampRef().

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 .

Referenced by getTimestampTzRef().

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.

Referenced by getTimeTzRef().

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

Referenced by Vertica::PartitionWriter::copyFromInput(), getType(), Vertica::BlockReader::isNull(), and Vertica::PartitionWriter::setNull().

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

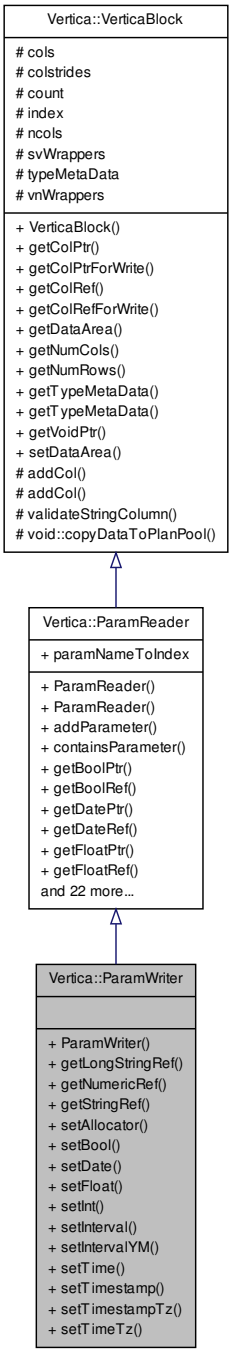
Bookkeepinp to make a parameter and its position in the block

Referenced by addParameter(), containsParameter(), Vertica::ParamWriter::getLongStringRef(), Vertica::ParamWriter::getNumericRef(), getParamNames(), Vertica::ParamWriter::getStringRef(), and isEmpty().

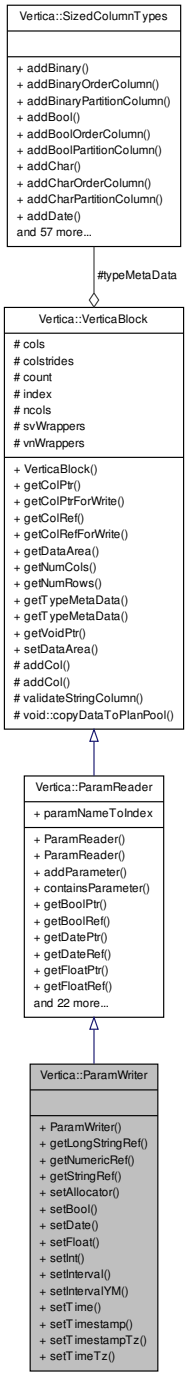
Vertica::ParamWriter Class Reference

Iterator interface for writing rows to a Vertica block.

Inheritance diagram for Vertica::ParamWriter:



Collaboration 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 EE::DataArea * **getDataArea** (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 ()
- void * **getVoidPtr** ()
- 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 **setDataArea** (size_t idx, void *dataarea)
- 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)
- friend **void::copyDataToPlanPool** (VerticaBlock *block)

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

Referenced by Vertica::ParamReader::addParameter().

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.

Referenced by Vertica::ParamReader::getBoolRef().

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);
*
```

Referenced by Vertica::PartitionWriter::copyFromInput().

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.

Referenced by Vertica::ParamReader::getDateRef().

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.

Referenced by Vertica::ParamReader::getFloatRef().

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.

Referenced by Vertica::ParamReader::getIntervalRef().

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.

Referenced by Vertica::ParamReader::getIntervalYMRef().

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");
*
```

Referenced by Vertica::ParamReader::getIntRef().

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.

Referenced by Vertica::BlockReader::isNull().

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.

Referenced by Vertica::ParamReader::getNumericRef().

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.

Referenced by Vertica::ParamReader::getStringRef().

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.

Referenced by Vertica::ParamReader::getTimeRef().

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.

Referenced by Vertica::ParamReader::getTimestampRef().

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 .

Referenced by Vertica::ParamReader::getTimestampTzRef().

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.

Referenced by Vertica::ParamReader::getTimeTzRef().

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

Referenced by Vertica::PartitionWriter::copyFromInput(), Vertica::ParamReader::getType(), Vertica::BlockReader-::isNull(), and Vertica::PartitionWriter::setNull().

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

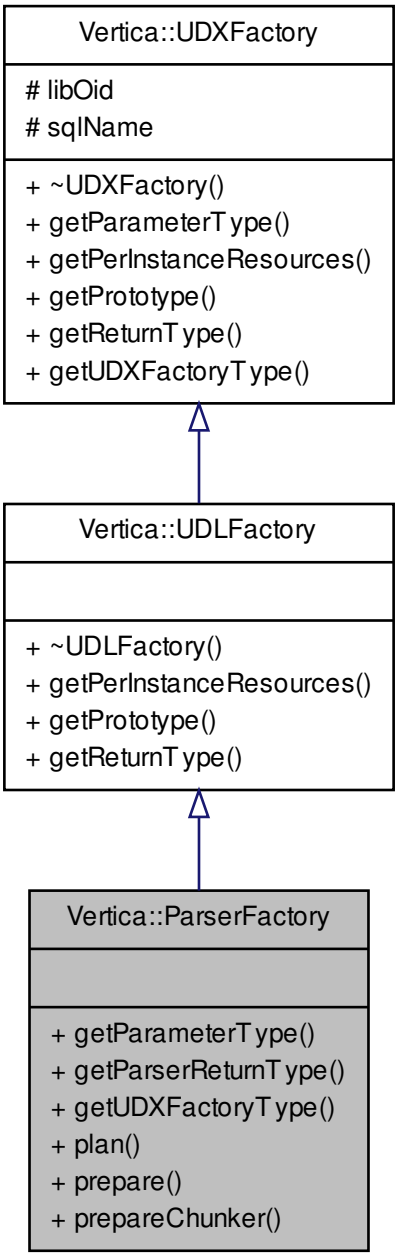
std::map<std::string, size_t> Vertica::ParamReader::paramNameToIndex `[inherited]`

Bookkeepinp to make a parameter and its position in the block

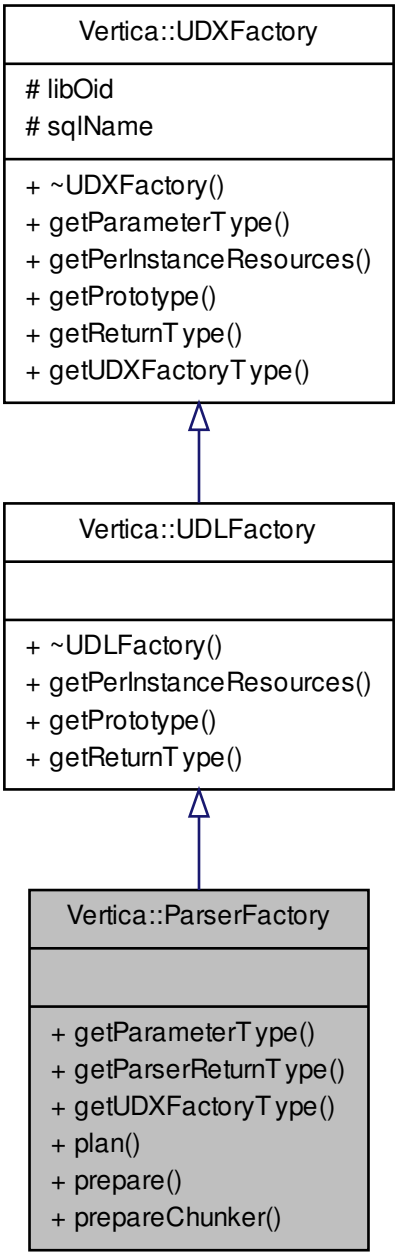
Referenced by Vertica::ParamReader::addParameter(), Vertica::ParamReader::containsParameter(), getLongStringRef(), getNumericRef(), Vertica::ParamReader::getParamNames(), getStringRef(), and Vertica::ParamReader::isEmpty().

Vertica::ParserFactory Class Reference

Inheritance diagram for Vertica::ParserFactory:



Collaboration diagram for Vertica::ParserFactory:



Public Types

- enum UDXType {
  FUNCTION, **TRANSFORM**, **ANALYTIC**, **MULTI_TRANSFORM**,
  AGGREGATE, **LOAD_SOURCE**, **LOAD_FILTER**, **LOAD_PARSER**,
  FILESYSTEM, **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
- virtual UDChunker * **prepareChunker** (ServerInterface &srvInterface, PerColumnParamReader &perColumnParamReader, PlanContext &planCtxt, const SizedColumnTypes &returnType)

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::UDLFactory::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

Reimplemented from Vertica::UDXFactory.

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-ParamReader	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-ParamReader	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

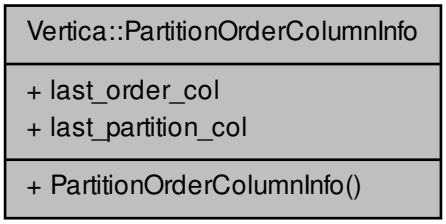
Returns

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.

Collaboration diagram for Vertica::PartitionOrderColumnInfo:



Public Attributes

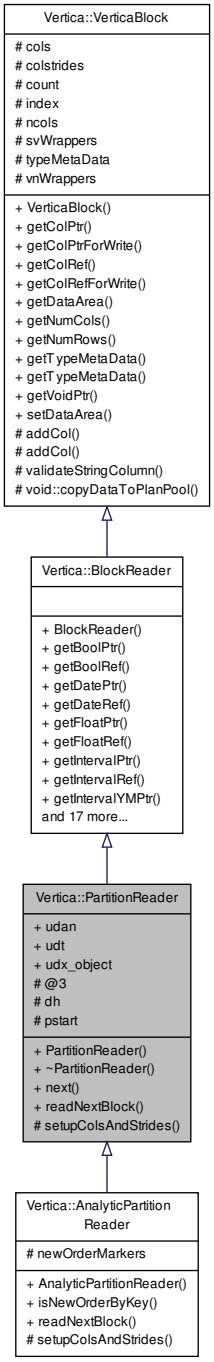
- int **last_order_col**
- int **last_partition_col**

Detailed Description

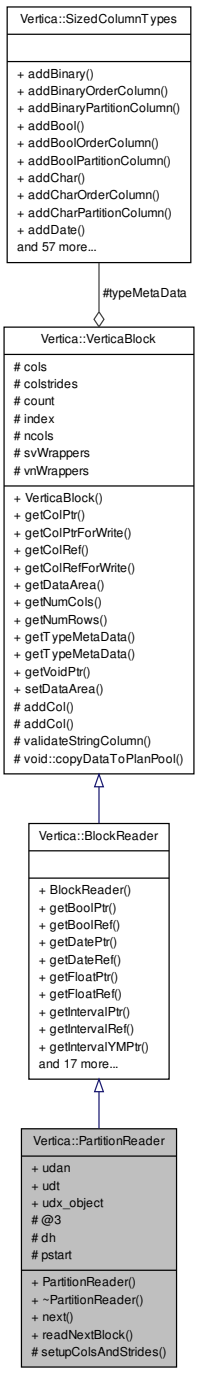
Represents the partition by and order by column information for each phase in a multi-phase transform function.

Vertica::PartitionReader Class Reference

Inheritance diagram for Vertica::PartitionReader:



Collaboration 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 EE::DataArea * **getDataArea** (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 ()
- void * **getVoidPtr** ()
- bool isNull (int col)

  Check if the idx'th argument is null.
- bool **next** ()
- virtual bool readNextBlock ()
- void **setDataArea** (size_t idx, void *dataarea)

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)
- friend **void::copyDataToPlanPool** (VerticaBlock *block)

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**
- struct **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

Referenced by Vertica::ParamReader::addParameter().

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.

Referenced by Vertica::BlockReader::getBoolRef().

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.

Referenced by Vertica::BlockReader::isNull().

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);
*
```

Referenced by Vertica::PartitionWriter::copyFromInput().

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.

Referenced by Vertica::BlockReader::getDateRef().

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.

Referenced by Vertica::BlockReader::isNull().

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.

Referenced by Vertica::BlockReader::getFloatRef().

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.

Referenced by Vertica::BlockReader::isNull().

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.

Referenced by Vertica::BlockReader::getIntervalRef().

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.

Referenced by Vertica::BlockReader::isNull().

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.

Referenced by Vertica::BlockReader::getIntervalYMRef().

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.

Referenced by Vertica::BlockReader::isNull().

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);
*
```

Referenced by Vertica::BlockReader::getIntRef().

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);
*
```

Referenced by Vertica::BlockReader::isNull().

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

Returns

the number of columns held by this block.

Referenced by Vertica::BlockReader::isNull().

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.

Referenced by Vertica::BlockReader::getNumericRef().

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.

Referenced by Vertica::BlockReader::isNull().

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.

Referenced by Vertica::PartitionWriter::copyFromInput(), and Vertica::BlockReader::getStringRef().

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.

Referenced by Vertica::BlockReader::isNull().

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.

Referenced by Vertica::BlockReader::getTimeRef().

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.

Referenced by Vertica::BlockReader::isNull().

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.

Referenced by Vertica::BlockReader::getTimestampRef().

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.

Referenced by Vertica::BlockReader::isNull().

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 .

Referenced by Vertica::BlockReader::getTimestampTzRef().

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.

Referenced by Vertica::BlockReader::isNull().

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.

Referenced by Vertica::BlockReader::getTimeTzRef().

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.

Referenced by Vertica::BlockReader::isNull().

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

Returns

information about the types and numbers of arguments

Referenced by Vertica::PartitionWriter::copyFromInput(), Vertica::ParamReader::getType(), Vertica::BlockReader::isNull(), and Vertica::PartitionWriter::setNull().

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.

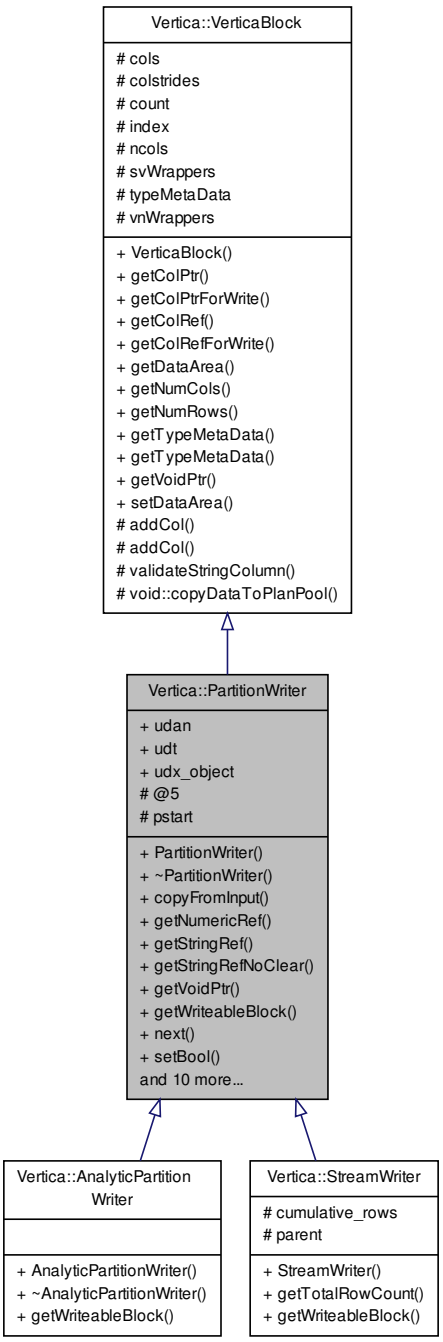
Returns

false if there's no more input data

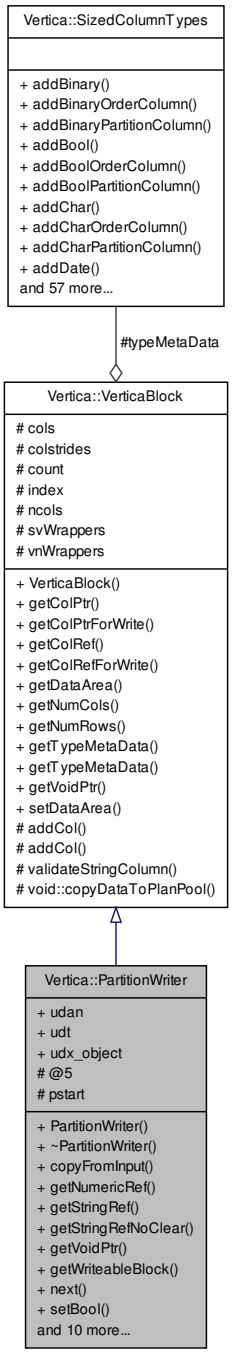
Reimplemented in Vertica::AnalyticPartitionReader.

Vertica::PartitionWriter Class Reference

Inheritance diagram for Vertica::PartitionWriter:



Collaboration 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)
- const EE::DataArea * **getDataArea** (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** ()
- void * **getVoidPtr** (size_t idx)
- virtual bool getWriteableBlock ()
- bool **next** ()
- void **setBool** (size_t idx, vbool r)
- void **setDataArea** (size_t idx, void *dataarea)
- 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)
- friend **void::copyDataToPlanPool** (VerticaBlock *block)

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

Referenced by Vertica::ParamReader::addParameter().

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);
*
```

Referenced by copyFromInput().

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.

Referenced by Vertica::BlockReader::isNull().

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

Referenced by copyFromInput(), Vertica::ParamReader::getType(), Vertica::BlockReader::isNull(), and setNull().

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

Referenced by setNull().

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

Set the idx'th argument to null.

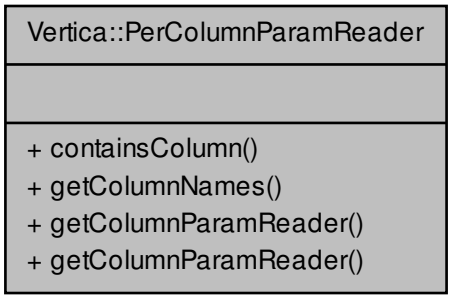
Parameters

idx	The column number in the row to set to null

Vertica::PerColumnParamReader Class Reference

: A wrapper around a map from column to ParamReader.

Collaboration diagram for Vertica::PerColumnParamReader:



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.
- const ParamReader & **getColumnParamReader** (const std::string &column) const

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

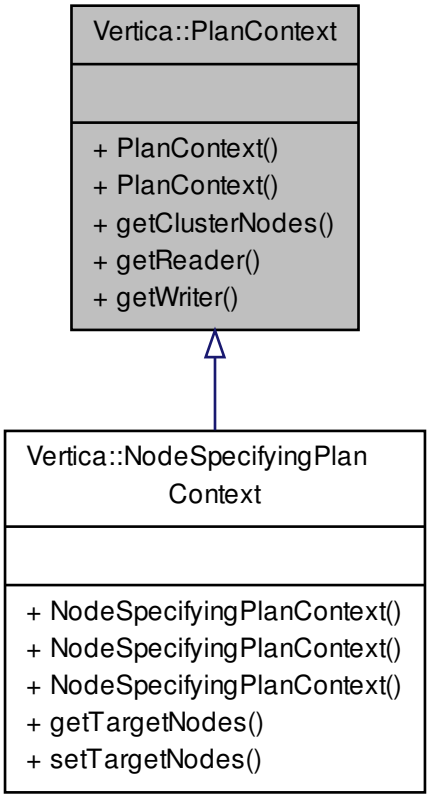
the	name of the column of interest

Returns

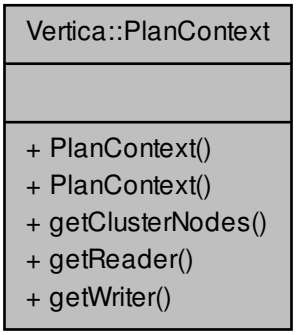
the parameters of the given column

Vertica::PlanContext Class Reference

Inheritance diagram for Vertica::PlanContext:



Collaboration 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

Referenced by Vertica::NodeSpecifyingPlanContext::setTargetNodes().

ParamReader& Vertica::PlanContext::getReader ( ) [inline]

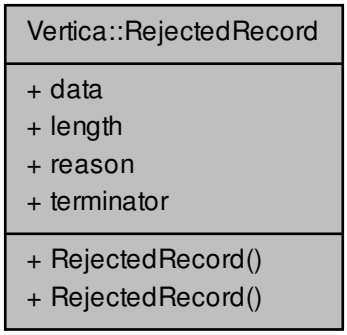
Get a read-only instance of the current context

ParamWriter& Vertica::PlanContext::getWriter ( ) [inline]

Get the current context for writing

Vertica::RejectedRecord Struct Reference

Collaboration diagram for Vertica::RejectedRecord:



Public Member Functions

- **RejectedRecord** (const std::string &reason, const char ∗data=NULL, size_t length=0, const std::string &terminator="\n")

Public Attributes

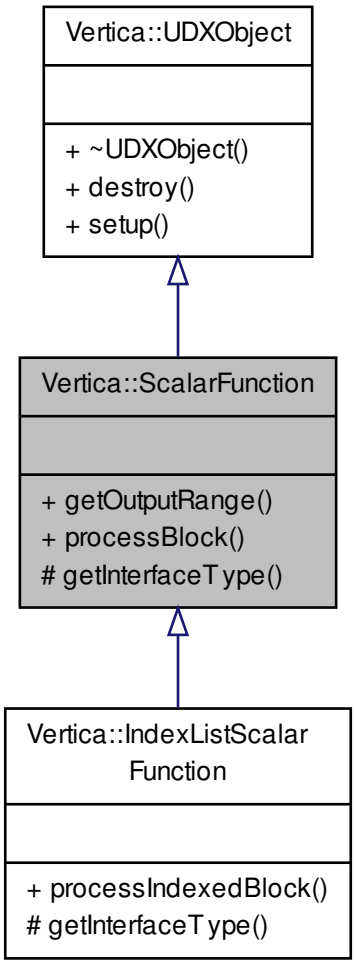
- const char ∗ **data**
- size_t **length**
- std::string **reason**
- std::string **terminator**

Detailed Description

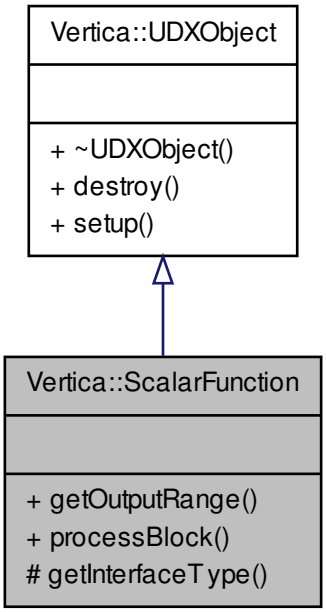
Information about a rejected record.

Vertica::ScalarFunction Class Reference

Inheritance diagram for Vertica::ScalarFunction:



Collaboration diagram for Vertica::ScalarFunction:



Public Member Functions

- virtual void destroy (ServerInterface &srvInterface, const SizedColumnTypes &argTypes)
- virtual void getOutputRange (ServerInterface &srvInterface, ValueRangeReader &inRange, ValueRangeWriter &outRange)
- virtual void processBlock (ServerInterface &srvInterface, BlockReader &arg_reader, BlockWriter &res_writer)=0
- virtual void setup (ServerInterface &srvInterface, const SizedColumnTypes &argTypes)

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::getOutputRange (ServerInterface & *srvInterface,* ValueRangeReader & *inRange,* ValueRangeWriter & *outRange*) `[inline],[virtual]`

Invoke a user defined function to determine the output value range of this function. Ranges are represented by a minimum/maximum pair of values (inclusive). The developer is responsible to provide an output value range on the basis of the input argument ranges. Minimum/maximum values of ranges are of the same type as defined in the metadata class getPrototype() function.

Parameters

srvInterface	a ServerInterface object used to communicate with Vertica
inRange	input value range
outRange	output value range

Remarks

By default, the ValueRangeWriter object can have NULL values, values in the range are unsorted, and it is unbounded, i.e., the following functions return as follows:

- outRange.canHaveNulls() == true
- outRange.getSortedness() == EE::SORT_UNORDERED
- outRange.isBounded() == false

Note

- This methods may be invoked by different threads at different times, and by a different thread than the constructor.
- C++ exceptions may NOT be thrown out of this method. Use the vertica specific vt_throw_exception() function or vt_report_error() macro instead
- Invoking vt_throw_exception() or vt_report_error() from this method will not stop the function execution, which may still complete successfully. Instead, the output range will be discarded, and a WARNING message will be written to the Vertica log

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

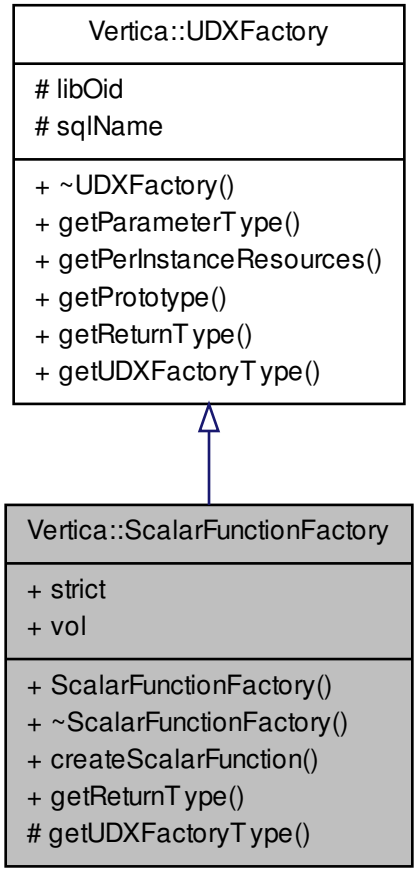
- This methods may be invoked by different threads at different times, and by a different thread than the constructor.
- The order in which the function sees rows is not guaranteed.
- C++ exceptions may NOT be thrown out of this method. Use the vertica specific vt_throw_exception() function or vt_report_error() macro instead

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

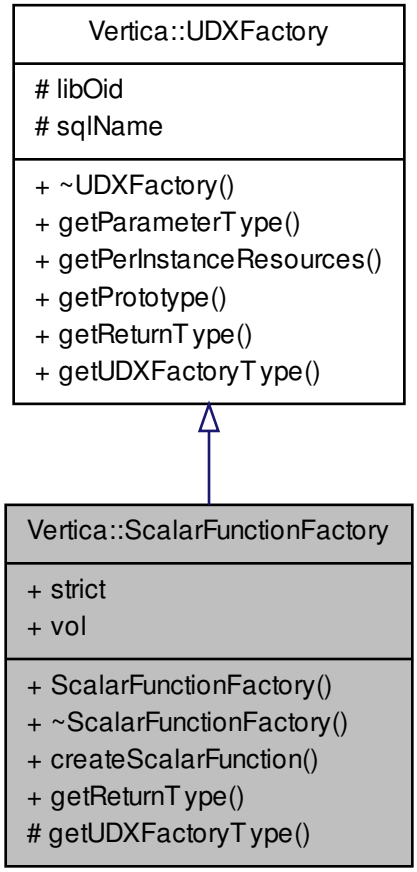
Perform per instance initialization. This function may throw errors.

Vertica::ScalarFunctionFactory Class Reference

Inheritance diagram for Vertica::ScalarFunctionFactory:



Collaboration diagram for Vertica::ScalarFunctionFactory:



Public Types

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

Reimplemented in Vertica::UDLFactory.

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, Vertica::MultiPhaseTransformFunctionFactory, and Vertica::UDFileSystemFactory.

Referenced by getReturnType().

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.

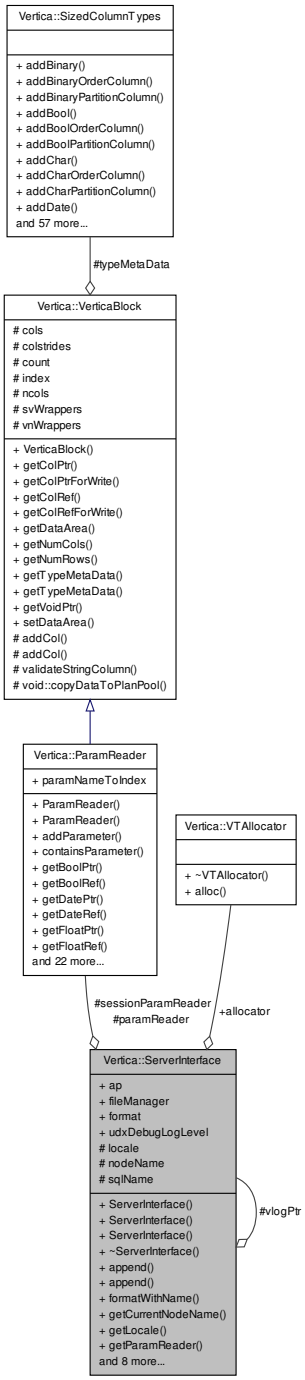
Member Data Documentation

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

Collaboration diagram for Vertica::ServerInterface:



Public Types

- typedef void(∗ **LoggingFunc**)(ServerInterface ∗, const char ∗fmt, va_list ap)

Public Member Functions

- ServerInterface (VTAllocator *allocator, FileManager *fileManager, LoggingFunc func, const std::string &sqlName, const ParamReader &paramReader, vint udxDebugLogLevel=0)
- **ServerInterface** (VTAllocator *allocator, LoggingFunc func, const std::string &sqlName, const ParamReader &paramReader, vint udxDebugLogLevel=0)
- **ServerInterface** (VTAllocator *allocator, LoggingFunc func, const std::string &sqlName, vint udxDebugLogLevel=0)
- formatWithName **append** (" - ")
- formatWithName **append** (format)
- void std::string **formatWithName** (sqlName)
- const std::string & getCurrentNodeName () const
- const std::string & getLocale () const
- ParamReader getParamReader ()
- ParamReader getSessionParamReader ()
- void log (const char *format,...) __attribute__((format(printf
- void setParamReader (const ParamReader &paramReader)
- void setSessionParamReader (const ParamReader &sessionParamReader)
- **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**
- FileManager * fileManager
- **format** = formatWithName.c_str()
- vint udxDebugLogLevel

Protected Attributes

- std::string locale
- std::string nodeName
- ParamReader paramReader
- ParamReader sessionParamReader
- std::string sqlName
- LoggingFunc vlogPtr

Friends

- class **::UdfSupport**
- class **EE::UserDefinedAggregate**
- class **EE::UserDefinedAnalytic**
- class **EE::UserDefinedTransform**

Detailed Description

Interface that UDX writers can use to interact with the Vertica Server

Constructor & Destructor Documentation

Vertica::ServerInterface::ServerInterface (VTAllocator * *allocator,* FileManager * *fileManager,* LoggingFunc *func,* const std::string & *sqlName,* const ParamReader & *paramReader,* vint *udxDebugLogLevel* = 0) `[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

ParamReader Vertica::ServerInterface::getSessionParamReader ( ) `[inline]`

Get the sessionParamReader

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

Returns the FileManager that allows interfaction with Catalog and storage system. 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::setSessionParamReader (const ParamReader & *sessionParamReader*) `[inline]`

Set the sessionParamReader

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.

FileManager∗ Vertica::ServerInterface::fileManager

File manager of the session context

std::string Vertica::ServerInterface::locale `[protected]`

The locale of the current session

Referenced by getLocale().

std::string Vertica::ServerInterface::nodeName `[protected]`

Store the name of the current node

Referenced by getCurrentNodeName().

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

Referenced by getParamReader(), and setParamReader().

ParamReader Vertica::ServerInterface::sessionParamReader `[protected]`

A map for session paremeters UDx might specify what session parameters it wants in its "manifest" Server will try to provide, if it agrees with that request

Referenced by getSessionParamReader(), and setSessionParamReader().

std::string Vertica::ServerInterface::sqlName `[protected]`

Store the name for error logging

vint Vertica::ServerInterface::udxDebugLogLevel

The level of UDx debug logging which is turned on as a UDXDebugLevel set of enumeration values. Used so UDxs may forgo generating debug log messages if debug logging is off. XXX jfraumeni TODO XXX Migrate to Alexi's communication storage, when ready XXX TODO XXX

LoggingFunc Vertica::ServerInterface::vlogPtr `[protected]`

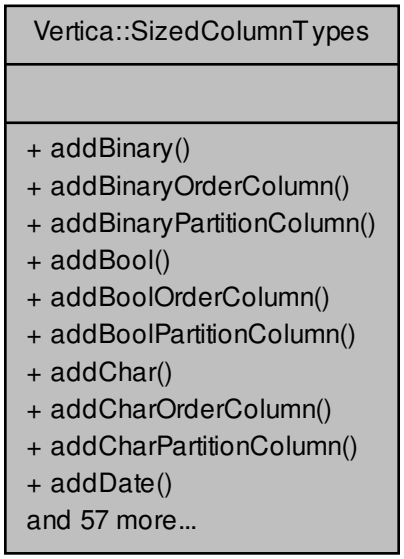
Callback for logging, set by the server

Referenced by vlog().

Vertica::SizedColumnTypes Class Reference

Represents types and information to determine the size of the columns as input/output of a User Defined Function/-Transform.

Collaboration diagram for Vertica::SizedColumnTypes:



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.

Referenced by Vertica::ParamWriter::setBool().

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.

Referenced by Vertica::ParamWriter::setDate().

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.

Referenced by Vertica::ParamWriter::setFloat().

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.

Referenced by Vertica::ParamWriter::setInterval().

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.

Referenced by Vertica::ParamWriter::setIntervalYM().

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.

Referenced by Vertica::ParamWriter::getLongStringRef().

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.

Referenced by Vertica::ParamWriter::getNumericRef().

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.

Referenced by Vertica::ParamWriter::setTime().

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.

Referenced by Vertica::ParamWriter::setTimestamp().

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.

Referenced by Vertica::ParamWriter::setTimestampTz().

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.

Referenced by Vertica::ParamWriter::setTimeTz().

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.

Referenced by Vertica::ParamWriter::getStringRef().

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

Referenced by Vertica::ParserFactory::getParserReturnType().

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.

Referenced by Vertica::PartitionWriter::copyFromInput(), Vertica::ParserFactory::getParserReturnType(), Vertica::ParamReader::getType(), Vertica::BlockReader::isNull(), Vertica::BlockWriter::next(), and Vertica::PartitionWriter::setNull().

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]`

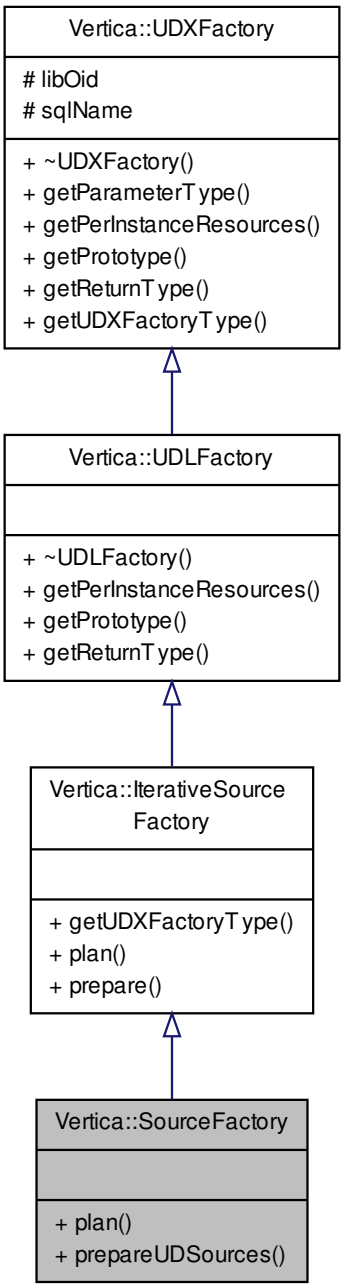
Sets the PARTITION BY and ORDER BY column indexes from another SizedColumnTypes object.

Parameters

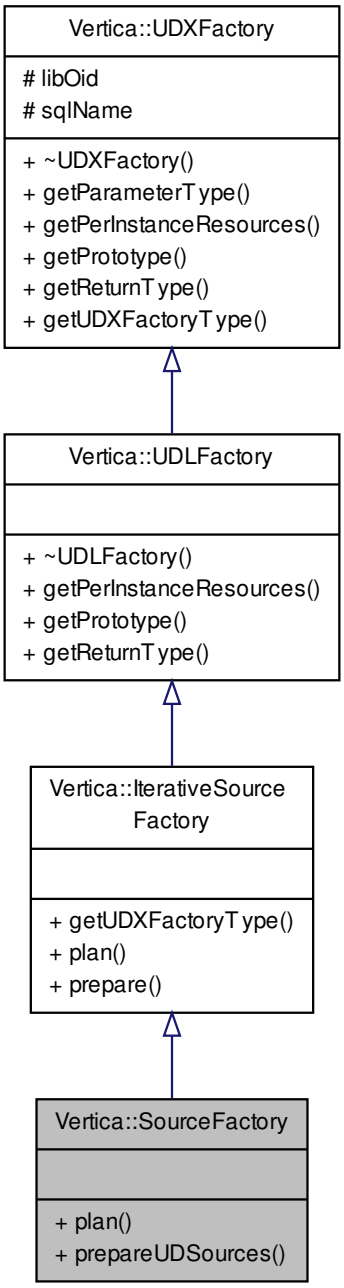
other	The SizedColumnTypes object to set the indexes from

Vertica::SourceFactory Class Reference

Inheritance diagram for Vertica::SourceFactory:



Collaboration diagram for Vertica::SourceFactory:



Public Types

- enum UDXType {
  FUNCTION, **TRANSFORM**, **ANALYTIC**, **MULTI_TRANSFORM**,
  AGGREGATE, **LOAD_SOURCE**, **LOAD_FILTER**, **LOAD_PARSER**,
  FILESYSTEM, **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::UDLFactory::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

Reimplemented from Vertica::UDXFactory.

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

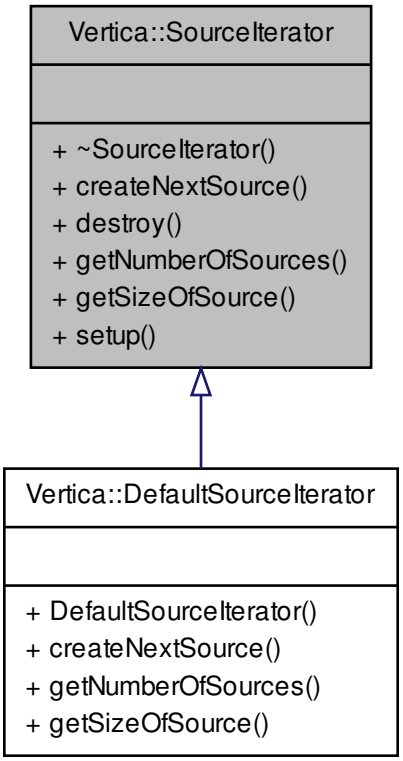
srvInterface	Interface to server operations and functionality, including (not-per-column) parameter lookup
planCtxt	Context for storing and retrieving arbitrary data, for use just by this instance of this query. The same context is shared with plan(). Also provides functionality for determining which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

Returns

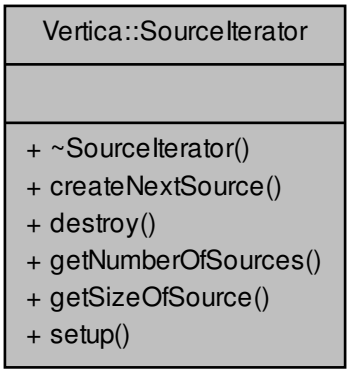
A vector of UDSources to use for this query. Sources will be loaded in a pooled manner, several at a time.

Vertica::SourceIterator Class Reference

Inheritance diagram for Vertica::SourceIterator:



Collaboration diagram for Vertica::SourceIterator:



Public Member Functions

- virtual UnsizedUDSource * createNextSource (ServerInterface &srvInterface)=0

  Create the next UDSource to process.
- virtual void destroy (ServerInterface &srvInterface, NodeSpecifyingPlanContext &planCtxt)

  Tear down this SourceIterator.
- virtual size_t getNumberOfSources ()=0
- virtual size_t getSizeOfSource (size_t sourceNum)
- virtual void setup (ServerInterface &srvInterface, NodeSpecifyingPlanContext &planCtxt)

  Set up this SourceIterator.

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 void Vertica::SourceIterator::destroy (ServerInterface & *srvInterface,* NodeSpecifyingPlanContext & *planCtxt*) `[inline], [virtual]`

Tear down this SourceIterator.

Should perform clean-up that should not take place in the destructor due to the exception-handling semantics of destructors.

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.

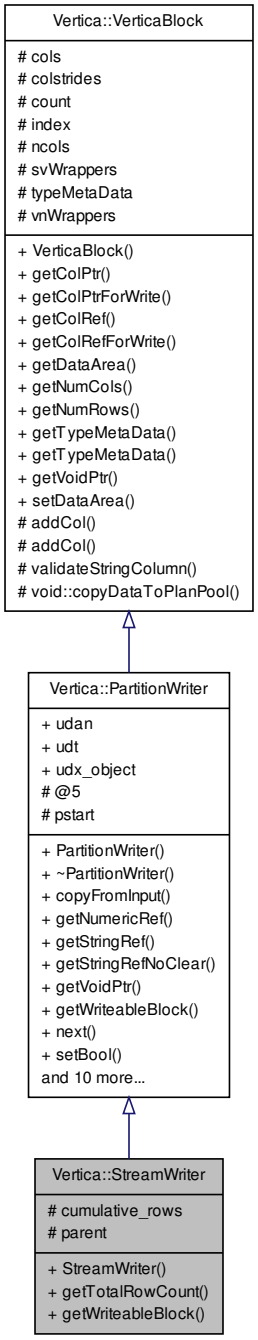
virtual void Vertica::SourceIterator::setup (ServerInterface & *srvInterface,* NodeSpecifyingPlanContext & *planCtxt*) `[inline], [virtual]`

Set up this SourceIterator.

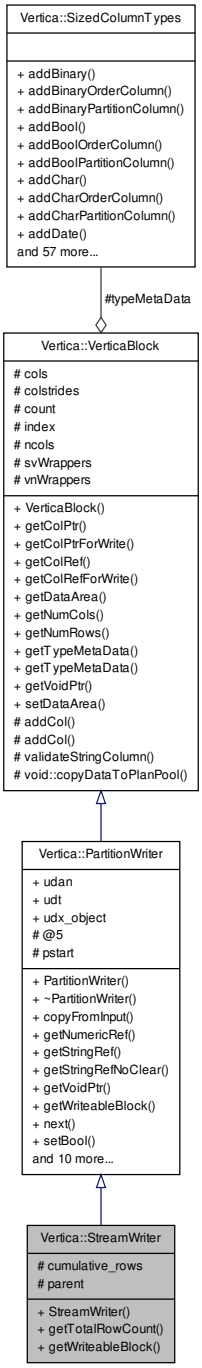
Should perform setup that should not take place in the constructor due to the exception-handling semantics of constructors

Vertica::StreamWriter Class Reference

Inheritance diagram for Vertica::StreamWriter:



Collaboration diagram for Vertica::StreamWriter:



Public Member Functions

- **StreamWriter** (size_t narg, void ∗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)
- const EE::DataArea * **getDataArea** (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)
- uint64 **getTotalRowCount** ()
- const SizedColumnTypes & getTypeMetaData () const
- SizedColumnTypes & getTypeMetaData ()
- void * **getVoidPtr** ()
- void * **getVoidPtr** (size_t idx)
- virtual bool getWriteableBlock ()
- bool **next** ()
- void **setBool** (size_t idx, vbool r)
- void **setDataArea** (size_t idx, void *dataarea)
- 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)
- friend **void::copyDataToPlanPool** (VerticaBlock *block)

Protected Attributes

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

- std::vector< char * > **cols**
- std::vector< int > **colstrides**
- int **count**
- uint64 **cumulative_rows**
- int **index**
- size_t **ncols**
- void * **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

Referenced by Vertica::ParamReader::addParameter().

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);
*
```

Referenced by Vertica::PartitionWriter::copyFromInput().

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.

Referenced by Vertica::BlockReader::isNull().

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

Referenced by Vertica::PartitionWriter::copyFromInput(), Vertica::ParamReader::getType(), Vertica::BlockReader::isNull(), and Vertica::PartitionWriter::setNull().

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

Referenced by Vertica::PartitionWriter::setNull().

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

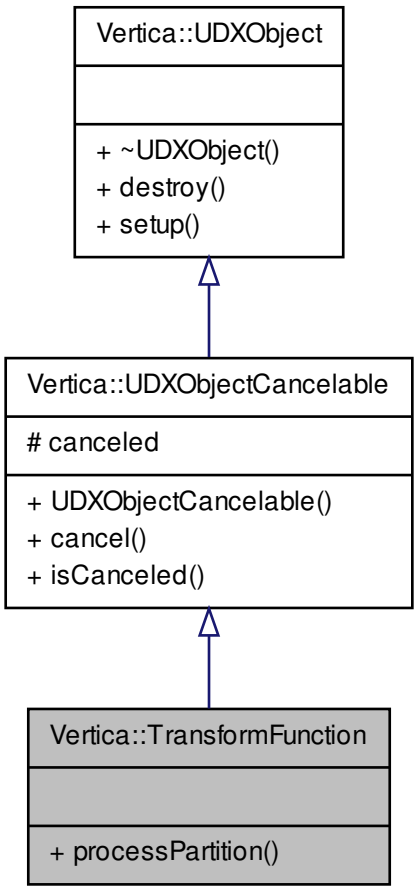
Set the idx'th argument to null.

Parameters

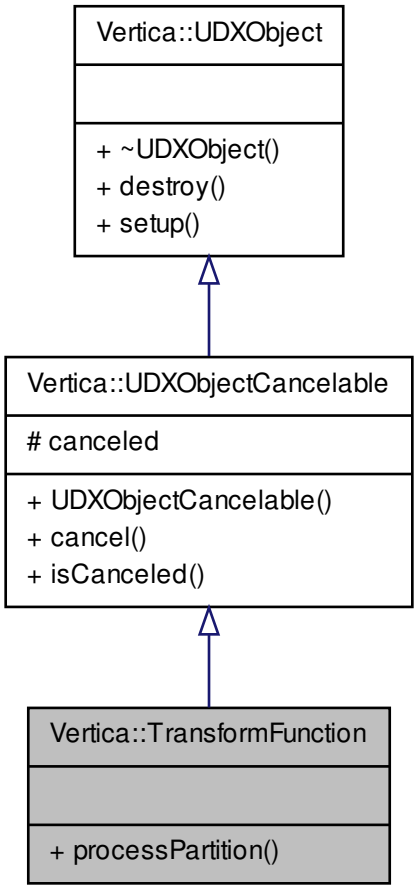
idx	The column number in the row to set to null

Vertica::TransformFunction Class Reference

Inheritance diagram for Vertica::TransformFunction:



Collaboration diagram for Vertica::TransformFunction:



Public Member Functions

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

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

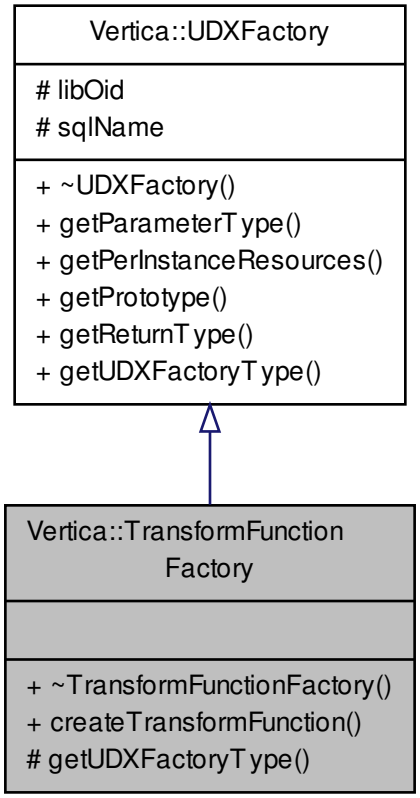
srvInterface	a ServerInterface object used to communicate with Vertica
input_reader	input rows
output_writer	output location

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

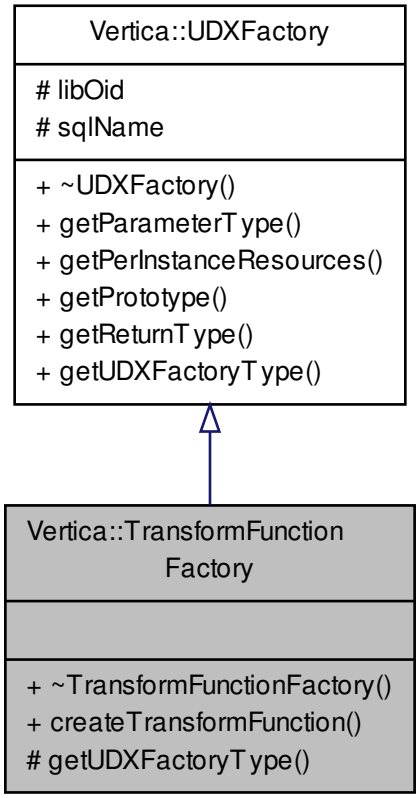
Perform per instance initialization. This function may throw errors.

Vertica::TransformFunctionFactory Class Reference

Inheritance diagram for Vertica::TransformFunctionFactory:



Collaboration diagram for Vertica::TransformFunctionFactory:



Public Types

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

Reimplemented in Vertica::UDLFactory.

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, Vertica::MultiPhaseTransformFunctionFactory, and Vertica::UDFileSystemFactory.

Referenced by Vertica::ScalarFunctionFactory::getReturnType().

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, Vertica::ScalarFunctionFactory, and Vertica::UDFileSystemFactory.

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

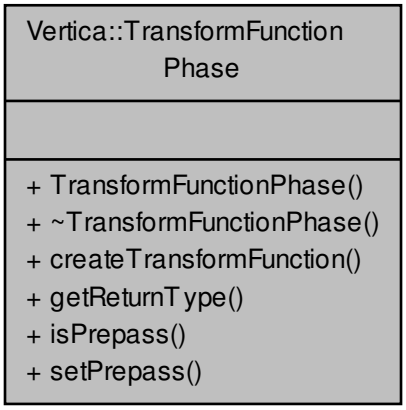
Returns

the object type internally used by Vertica

Implements Vertica::UDXFactory.

Vertica::TransformFunctionPhase Class Reference

Collaboration diagram for Vertica::TransformFunctionPhase:



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::createTransformFunction (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

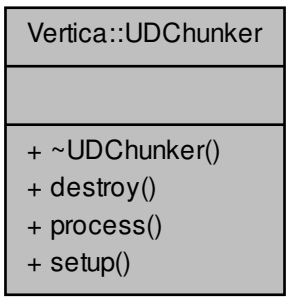
For all other types, no length/precision specification needed

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::UDChunker Class Reference

Collaboration diagram for Vertica::UDChunker:



Public Member Functions

- virtual ~UDChunker ()
- virtual void destroy (ServerInterface &srvInterface, SizedColumnTypes &returnType)
- virtual StreamState process (ServerInterface &srvInterface, DataBuffer &input, InputState input_state)=0
- virtual void setup (ServerInterface &srvInterface, SizedColumnTypes &returnType)

Detailed Description

UDChunker

Class that allows the separation of parsing record boundaries, which in some cases cannot be parallelized, from parsing "column" values, which is typically the bottleneck and which, thankfully, can usually be parallelized.

Constructor & Destructor Documentation

virtual Vertica::UDChunker::~UDChunker ( ) `[inline], [virtual]`

Dtor.

Member Function Documentation

virtual void Vertica::UDChunker::destroy (ServerInterface & *srvInterface,* SizedColumnTypes & *returnType*) `[inline], [virtual]`

UDChunker::destroy()

Will be invoked during query execution, after the last time that process() is called on this UDChunker instance for a particular input.

May optionally be overridden to perform tear-down/destruction.

See UDChunker::setup() for a note about the restartability of UDChunkers.

virtual StreamState Vertica::UDChunker::process (ServerInterface & *srvInterface,* DataBuffer & *input,* InputState *input_state*) `[pure virtual]`

UDChunker::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, find record boundaries and align input.offset with the end of the last record in the buffer. Once it has processed as much as it reasonably can (for example, once it has processed 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 a few rows were found in current block, move offset forward to point at the start of next (potential) row, and mark state as CHUNK_ALIGNED, indicating the chunker is ready to hand this chunk to parser.

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.

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.

Returns

INPUT_NEEDED if this UDChunker has more data to process; DONE if has no more data to produce; CHUNK_ALIGNED if this UDChunker successfully aligned some rows and is ready to give them to parser.

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::UDChunker::setup (ServerInterface & *srvInterface,* SizedColumnTypes & *returnType*) `[inline], [virtual]`

UDChunker::setup()

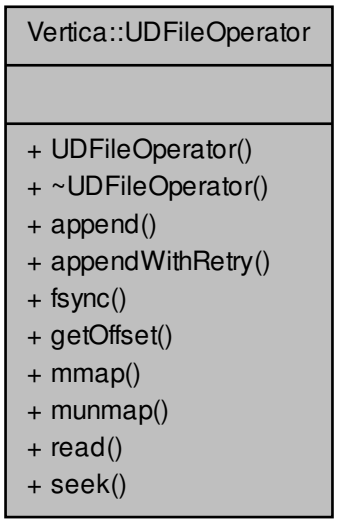
Will be invoked during query execution, prior to the first time that process() is called on this UDChunker instance for a particular input source.

May optionally be overridden to perform setup/initialzation.

Note that UDChunkers MUST BE RESTARTABLE! If loading large numbers of files, a given UDChunker may be re-used for multiple files. Vertica follows the worker-pool design pattern: At the start of COPY execution, several Chunkers and several Filters are instantiated per node, by calling the corresponding prepare() method multiple times. Each Filter/Chunker 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/UDChunker 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::UDFileOperator Class Reference

Collaboration diagram for Vertica::UDFileOperator:



Public Member Functions

- virtual size_t **append** (const void ∗buf, size_t count)=0
- void appendWithRetry (const void ∗buffer, size_t size)
- virtual void **fsync** ()
- virtual off_t **getOffset** () const =0
- virtual void ∗ **mmap** (void ∗addr, size_t length, int prot, int flags, off_t offset)
- virtual void **munmap** (void ∗addr, size_t length)
- virtual size_t **read** (void ∗buf, size_t count)=0
- virtual off_t **seek** (off_t offset, int whence)=0

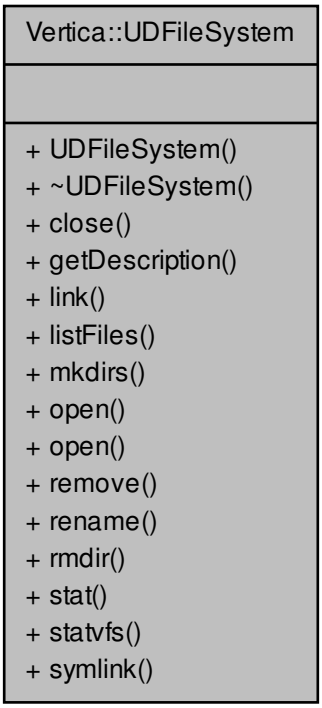
Member Function Documentation

void Vertica::UDFileOperator::appendWithRetry (const void ∗ *buffer,* size_t *size*)

A wrapper around the append(). Attempt to write the specified data to disk. Retry in cases that didn't succeed in writing all data, but that stand a decent chance of succeeding on a retry. Throws on error.

Vertica::UDFileSystem Class Reference

Collaboration diagram for Vertica::UDFileSystem:

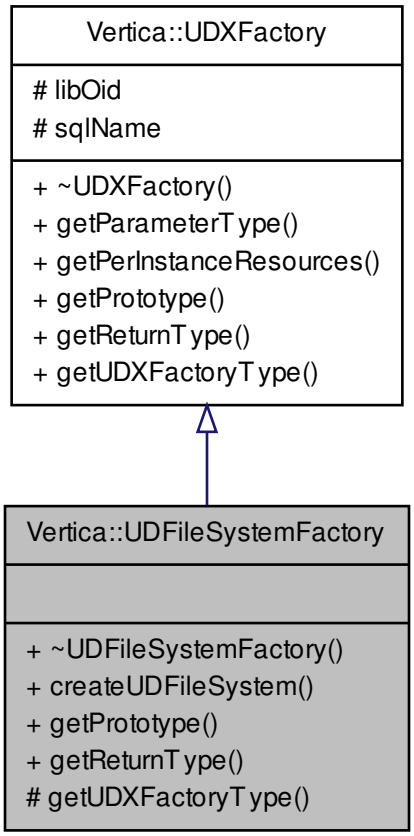


Public Member Functions

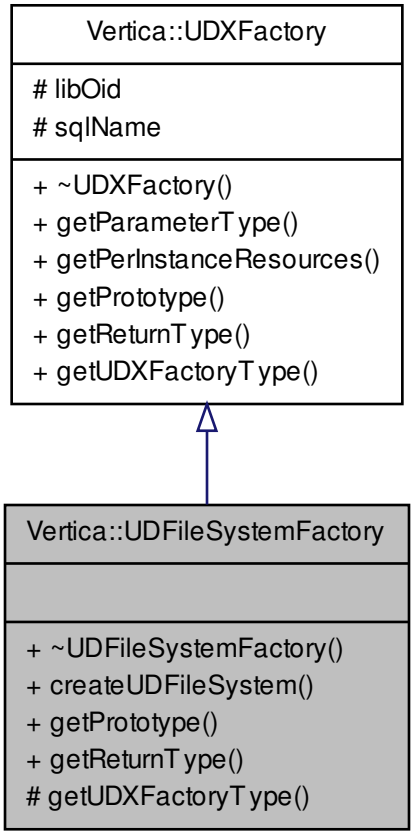
- virtual void **close** (UDFileOperator ∗udfo) const =0
- virtual std::string **getDescription** () const
- virtual void **link** (const char ∗oldpath, const char ∗newpath) const
- virtual void **listFiles** (const char ∗path, std::vector< std::string > &result) const =0
- virtual void **mkdirs** (const char ∗path, mode_t mode) const =0
- virtual UDFileOperator ∗ **open** () const =0
- virtual UDFileOperator ∗ **open** (const char ∗path, int flags, mode_t mode, bool removeOnClose=false) const =0
- virtual void **remove** (const char ∗path) const =0
- virtual void **rename** (const char ∗oldpath, const char ∗newpath) const =0
- virtual void **rmdir** (const char ∗path) const =0
- virtual void **stat** (const char ∗path, struct::stat ∗buf) const =0
- virtual void **statvfs** (const char ∗path, struct::statvfs ∗buf) const =0
- virtual void **symlink** (const char ∗oldpath, const char ∗newpath) const

Vertica::UDFileSystemFactory Class Reference

Inheritance diagram for Vertica::UDFileSystemFactory:



Collaboration diagram for Vertica::UDFileSystemFactory:



Public Types

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

Public Member Functions

- virtual UDFileSystem * **createUDFileSystem** (ServerInterface &srvInterface)=0
- 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)

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

Reimplemented in Vertica::UDLFactory.

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

UDFileSystems take no input tuples; as such, their prototype must be empty.

Implements Vertica::UDXFactory.

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

UDFileSystems return no output; as such, their prototype must be empty.

Implements Vertica::UDXFactory.

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

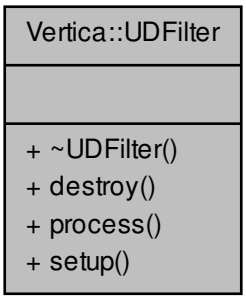
Returns

the object type internally used by Vertica

Implements Vertica::UDXFactory.

Vertica::UDFilter Class Reference

Collaboration diagram for Vertica::UDFilter:



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.

Note that, unless OUTPUT_NEEDED is returned, `output` will be UNMODIFIED the next time process() is called. This means that pointers into the buffer will continue to be valid. It also means that `output.offset` may be set. So, in general, process() code should assume that buffers start at `output.buf[output.offset]`. The same goes for `input` and INPUT_NEEDED. Note also that, as a performance optimization, upstream operators may start processing emitted data (data between output.buf[0] and output.buf[output.offset]) before OUTPUT_NEEDED is returned. For this reason, `output.offset` must be strictly increasing.

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()

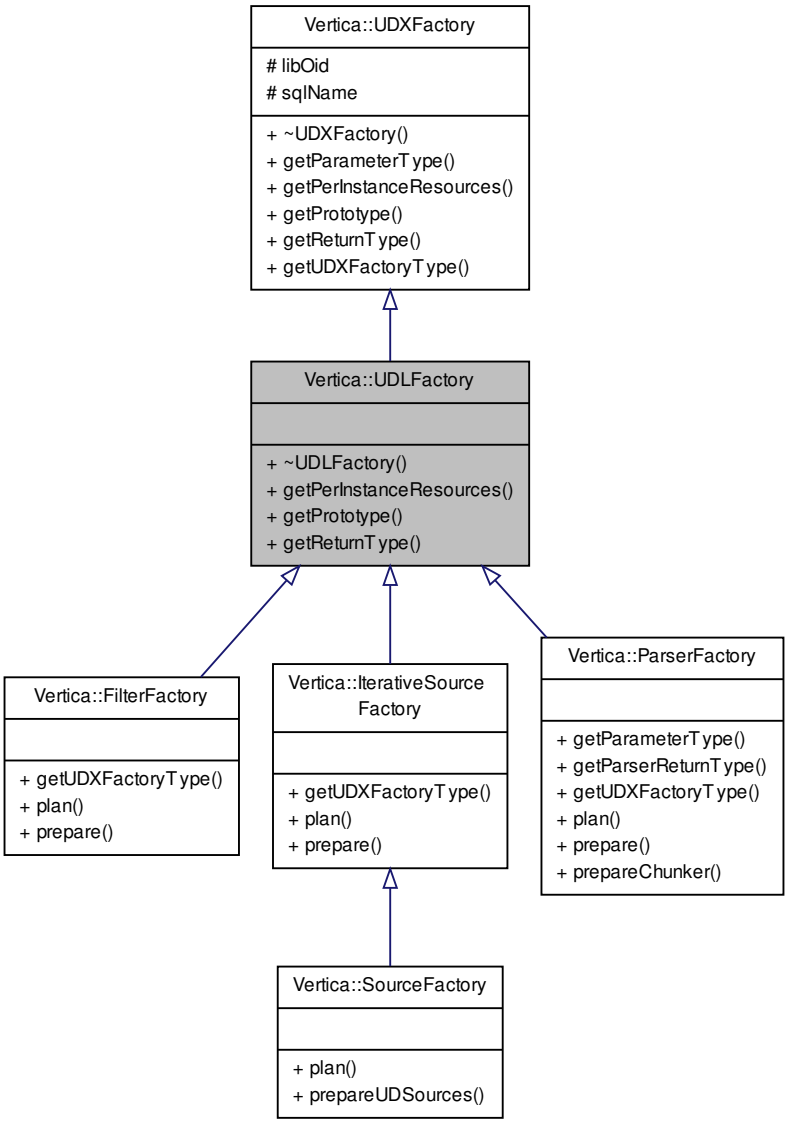
Will be invoked during query execution, prior to the first time that process() is called on this UDFilter instance for a particular input file.

May optionally be overridden to perform setup/initialzation.

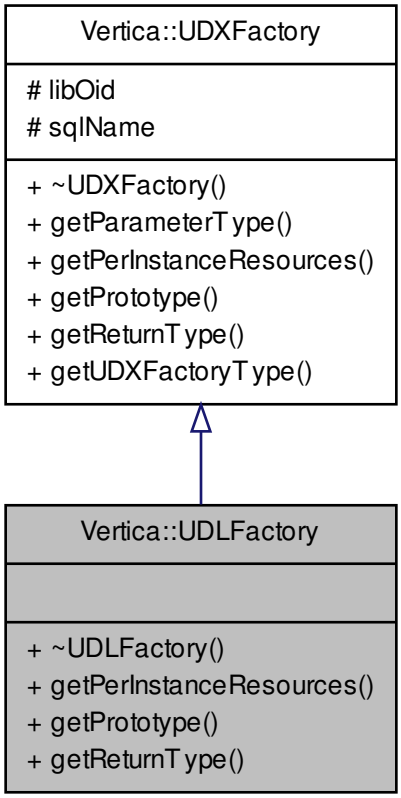
Note that UDFilters MUST BE RESTARTABLE! If loading large numbers of files, a given UDFilter may be re-used for multiple files. Vertica follows the worker-pool design pattern: At the start of COPY execution, several Parsers and several Filters are instantiated per node, by calling the corresponding prepare() 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:



Collaboration diagram for Vertica::UDLFactory:



Public Types

- enum UDXType {
  FUNCTION, **TRANSFORM**, **ANALYTIC**, **MULTI_TRANSFORM**,
  AGGREGATE, **LOAD_SOURCE**, **LOAD_FILTER**, **LOAD_PARSER**,
  FILESYSTEM, **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::UDLFactory::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

Reimplemented from Vertica::UDXFactory.

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.

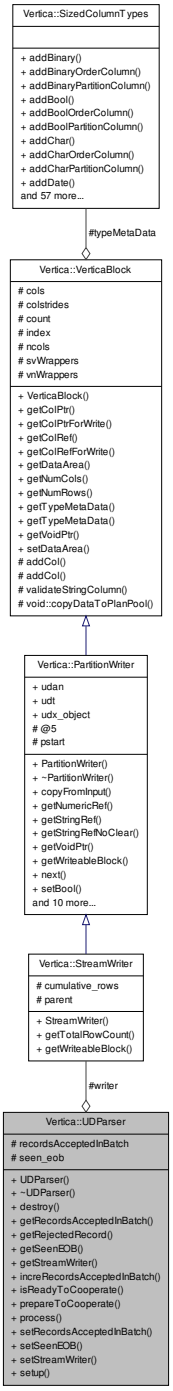
Note

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, Vertica::IterativeSourceFactory, and Vertica::UDFileSystemFactory.

Vertica::UDParser Class Reference

Collaboration diagram for Vertica::UDParser:



Public Member Functions

- virtual void destroy (ServerInterface &srvInterface, SizedColumnTypes &returnType)
- vint **getRecordsAcceptedInBatch** ()
- virtual RejectedRecord getRejectedRecord ()

- bool **getSeenEOB** ()
- StreamWriter * **getStreamWriter** ()
- void **increRecordsAcceptedInBatch** ()
- virtual bool isReadyToCooperate (ServerInterface &srvInterface)
- virtual void prepareToCooperate (ServerInterface &srvInterface, bool isLeader)
- virtual StreamState process (ServerInterface &srvInterface, DataBuffer &input, InputState input_state)=0
- void **setRecordsAcceptedInBatch** (vint i)
- void **setSeenEOB** (bool b)
- void **setStreamWriter** (StreamWriter *writer)
- virtual void setup (ServerInterface &srvInterface, SizedColumnTypes &returnType)

Protected Attributes

- vint **recordsAcceptedInBatch**
- bool **seen_eob**
- 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 bool Vertica::UDParser::isReadyToCooperate (ServerInterface & *srvInterface*) `[inline], [virtual]`

UDParser::isReadyToCooperate()

Called after UDParser::prepareToCooperate(), returns false if this parser is not yet ready to cooperate. Once this method returns true the parser can begin to cooperate. Default implementation returns true, override if some preparation is required before the parser can cooperate (e.g. a certain # of rows must be skipped).

virtual void Vertica::UDParser::prepareToCooperate (ServerInterface & *srvInterface,* bool *isLeader*) `[inline], [virtual]`

UDParser::prepareToCooperate()

Notification to this parser that it should prepare to share parsing input with another. This can only happen when a parser has an associated chunker. Default implementation does nothing.

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.

Note that, unless INPUT_NEEDED is returned, `input` will be UNMODIFIED the next time process() is called. This means that pointers into the buffer will continue to be valid. It also means that `input.offset` may be set. So, in general, process() code should assume that buffers start at `input.buf[input.offset]`.

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. All previous input must have been consumed by a 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.

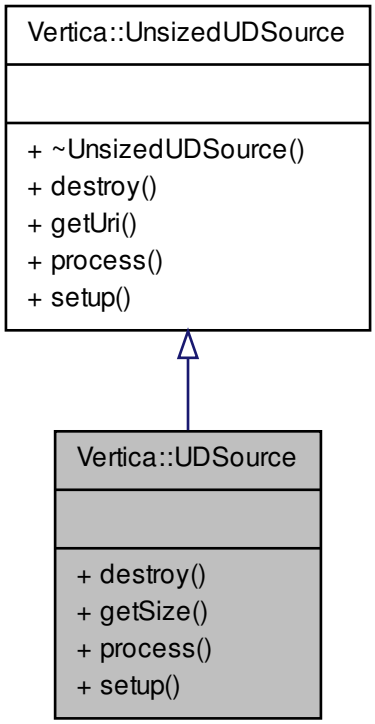
Member Data Documentation

StreamWriter* **Vertica::UDParser::writer** `[protected]`

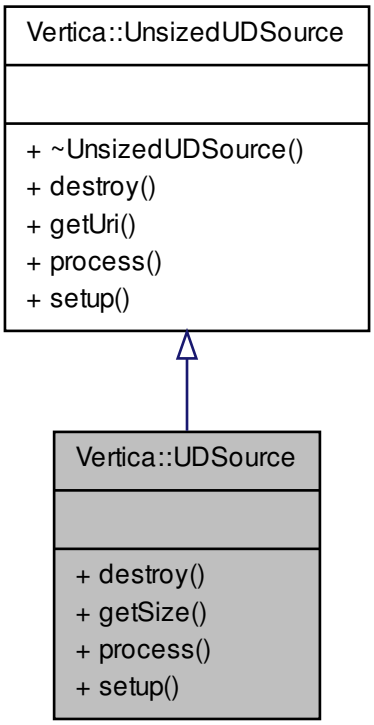
Writer to write parsed tuples to. Has the same API as PartitionWriter, from the UDT framework.

Vertica::UDSource Class Reference

Inheritance diagram for Vertica::UDSource:



Collaboration 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.

Note that, unless OUTPUT_NEEDED is returned, `output` will be UNMODIFIED the next time process() is called. This means that pointers into the buffer will continue to be valid. It also means that `output.offset` may be set. So, in general, process() code should assume that buffers start at `output.buf[output.offset]`. Note also that, as a performance optimization, upstream operators may start processing emitted data (data between output.buf[0] and output.buf[output.offset]) before OUTPUT_NEEDED is returned. For this reason, `output.offset` must be strictly increasing.

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()

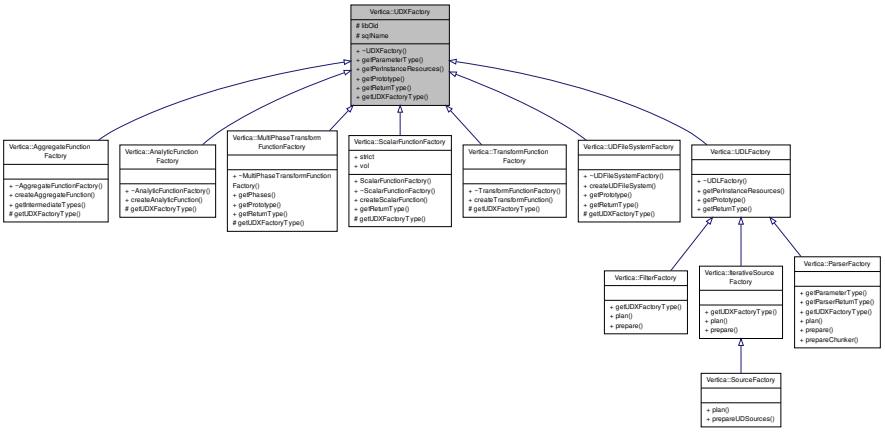
Will be invoked during query execution, prior to the first time that process() is called on this UDSource instance.

May optionally be overridden to perform setup/initialzation.

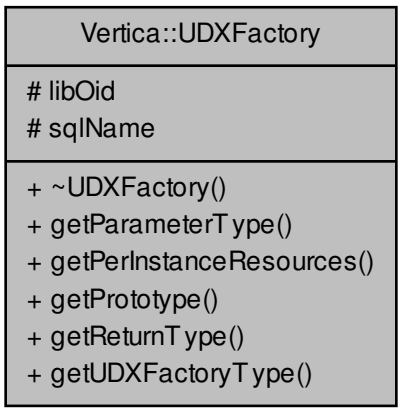
Reimplemented from Vertica::UnsizedUDSource.

Vertica::UDXFactory Class Reference

Inheritance diagram for Vertica::UDXFactory:



Collaboration diagram for Vertica::UDXFactory:



Public Types

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

Reimplemented in Vertica::UDLFactory.

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, Vertica::MultiPhaseTransformFunctionFactory, and Vertica::UDFileSystemFactory.

Referenced by Vertica::ScalarFunctionFactory::getReturnType().

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, Vertica::ScalarFunctionFactory, and Vertica::UDFileSystemFactory.

virtual UDXType Vertica::UDXFactory::getUDXFactoryType ( ) [pure virtual]

Returns

the type of UDX Object instance this factory returns.

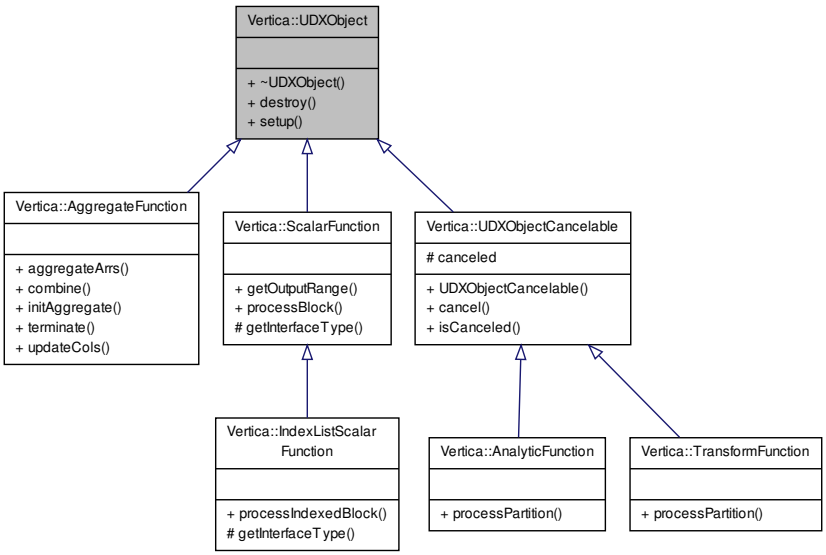
Note

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

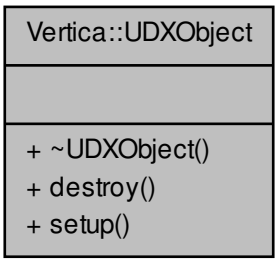
Implemented in Vertica::MultiPhaseTransformFunctionFactory, Vertica::AggregateFunctionFactory, Vertica::AnalyticFunctionFactory, Vertica::TransformFunctionFactory, Vertica::ScalarFunctionFactory, Vertica::ParserFactory, Vertica::FilterFactory, Vertica::IterativeSourceFactory, and Vertica::UDFileSystemFactory.

Vertica::UDXObject Class Reference

Inheritance diagram for Vertica::UDXObject:



Collaboration 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]`

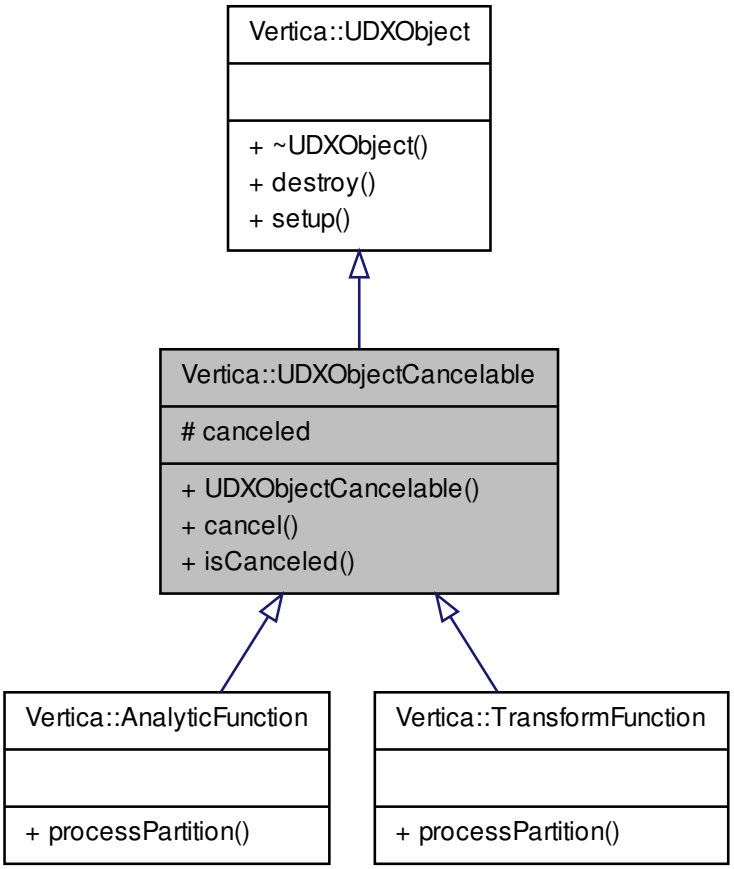
Perform per instance destruction. This function may throw errors

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

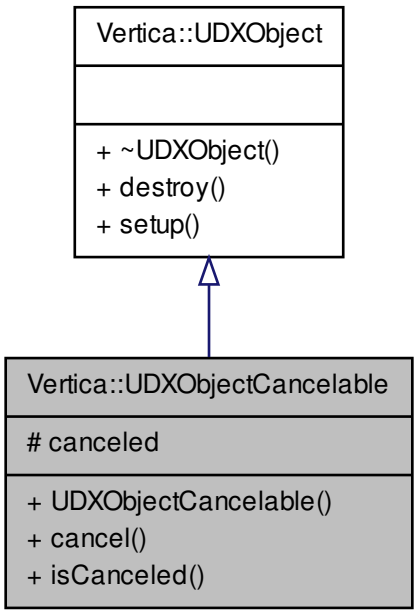
Perform per instance initialization. This function may throw errors.

Vertica::UDXObjectCancelable Class Reference

Inheritance diagram for Vertica::UDXObjectCancelable:



Collaboration 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]

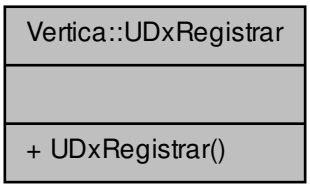
Returns true if execution was canceled.

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

Collaboration diagram for Vertica::UDxRegistrar:

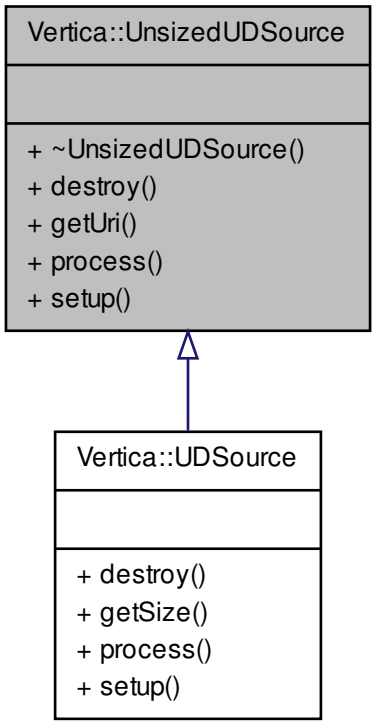


Public Member Functions

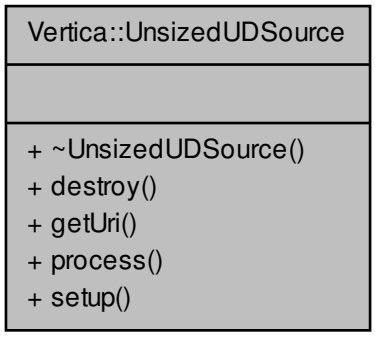
- **UDxRegistrar** (const char ∗name)

Vertica::UnsizedUDSource Class Reference

Inheritance diagram for Vertica::UnsizedUDSource:



Collaboration 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()

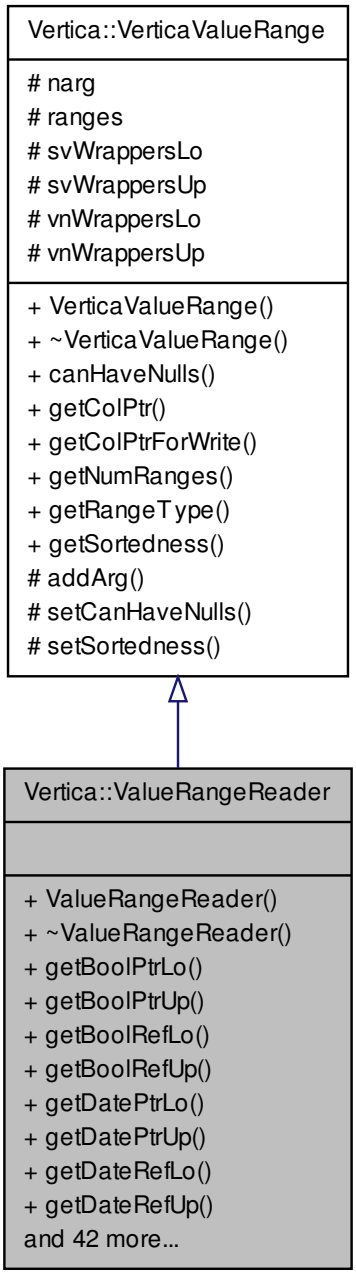
Return the URI of the current source of data.

This function will be invoked during execution to fill in monitoring information.

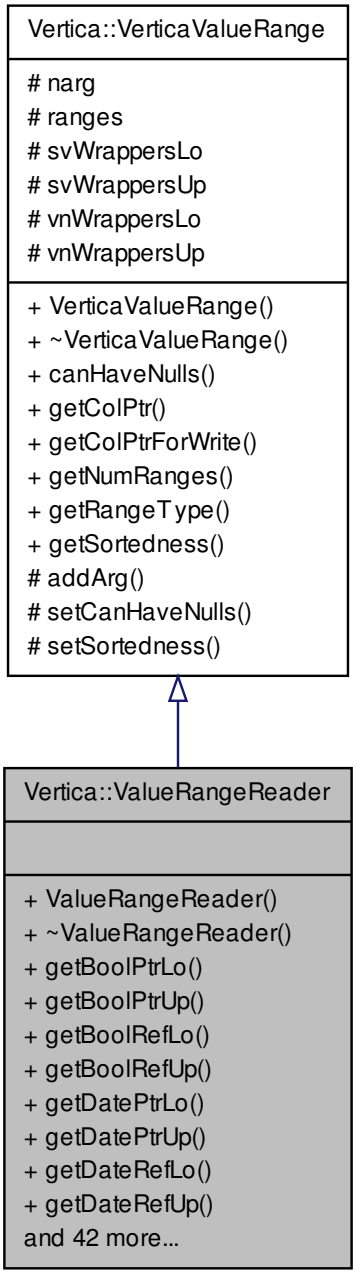
Vertica::ValueRangeReader Class Reference

This class represents the value ranges of the arguments of a UDSF, one range per argument.

Inheritance diagram for Vertica::ValueRangeReader:



Collaboration diagram for Vertica::ValueRangeReader:



Public Member Functions

- **ValueRangeReader** (size_t narg)
- bool canHaveNulls (size_t idx)

  Indicates if there can be NULL values in the range.
- const vbool * getBoolPtrLo (size_t idx)

  Get a pointer to a BOOLEAN value from a range bound.

- const vbool * **getBoolPtrUp** (size_t idx)
- const vbool & getBoolRefLo (size_t idx)

  Get a reference to a BOOLEAN value from a range bound.
- const vbool & **getBoolRefUp** (size_t idx)
- template<class T , BoundType b>
  const T * **getColPtr** (size_t idx)
- template<class T , BoundType b>
  T * **getColPtrForWrite** (size_t idx)
- const DateADT * getDatePtrLo (size_t idx)

  Get a pointer to a DATE value from a range bound.
- const DateADT * **getDatePtrUp** (size_t idx)
- const DateADT & getDateRefLo (size_t idx)

  Get a reference to a DATE value from a range bound.
- const DateADT & **getDateRefUp** (size_t idx)
- const vfloat * getFloatPtrLo (size_t idx)

  Get a pointer to a FLOAT value from a range bound.
- const vfloat * **getFloatPtrUp** (size_t idx)
- const vfloat & getFloatRefLo (size_t idx)

  Get a reference to a FLOAT value from a range bound.
- const vfloat & **getFloatRefUp** (size_t idx)
- const Interval * getIntervalPtrLo (size_t idx)

  Get a pointer to an INTERVAL value from a range bound.
- const Interval * **getIntervalPtrUp** (size_t idx)
- const Interval & getIntervalRefLo (size_t idx)

  Get a reference to an INTERVAL value from a range bound.
- const Interval & **getIntervalRefUp** (size_t idx)
- const IntervalYM * getIntervalYMPtrLo (size_t idx)

  Get a pointer to a INTERVAL YEAR TO MONTH value from a range bound.
- const IntervalYM * **getIntervalYMPtrUp** (size_t idx)
- const IntervalYM & getIntervalYMRefLo (size_t idx)

  Get a reference to an INTERVAL YEAR TO MONTH value from a range bound.
- const IntervalYM & **getIntervalYMRefUp** (size_t idx)
- const vint * getIntPtrLo (size_t idx)

  Get a pointer to an INTEGER value from a range bound.
- const vint * **getIntPtrUp** (size_t idx)
- const vint & getIntRefLo (size_t idx)

  Get a reference to an INTEGER value from a range bound.
- const vint & **getIntRefUp** (size_t idx)
- const VNumeric * getNumericPtrLo (size_t idx)

  Get a pointer to a VNumeric value from a range bound.
- const VNumeric * **getNumericPtrUp** (size_t idx)
- const VNumeric & getNumericRefLo (size_t idx)

  Get a reference to a VNumeric value from a range bound.
- const VNumeric & **getNumericRefUp** (size_t idx)
- size_t getNumRanges () const

  Retrieve the number of range arguments.
- const VerticaType & getRangeType (size_t idx) const

  Returns the data type of the values in a range.
- EE::ValueSort getSortedness (size_t idx)

  Gets the sortedness of values in a range.
- const VString * getStringPtrLo (size_t idx)

  Get a pointer to a VString value from a range bound.

- const VString * **getStringPtrUp** (size_t idx)
- const VString & getStringRefLo (size_t idx)

  Get a reference to an VString value from a range bound.
- const VString & **getStringRefUp** (size_t idx)
- const TimeADT * getTimePtrLo (size_t idx)

  Get a pointer to a TIME value from a range bound.
- const TimeADT * **getTimePtrUp** (size_t idx)
- const TimeADT & getTimeRefLo (size_t idx)

  Get a reference to a TIME value from a range bound.
- const TimeADT & **getTimeRefUp** (size_t idx)
- const Timestamp * getTimestampPtrLo (size_t idx)

  Get a pointer to a TIMESTAMP value from a range bound.
- const Timestamp * **getTimestampPtrUp** (size_t idx)
- const Timestamp & getTimestampRefLo (size_t idx)

  Get a reference to a TIMESTAMP value from a range bound.
- const Timestamp & **getTimestampRefUp** (size_t idx)
- const TimestampTz * getTimestampTzPtrLo (size_t idx)

  Get a pointer to a TIMESTAMP WITH TIMEZONE value from a range bound.
- const TimestampTz * **getTimestampTzPtrUp** (size_t idx)
- const TimestampTz & getTimestampTzRefLo (size_t idx)

  Get a reference to a TIMESTAMP WITH TIMEZONE value from a range bound.
- const TimestampTz & **getTimestampTzRefUp** (size_t idx)
- const TimeTzADT * getTimeTzPtrLo (size_t idx)

  Get a pointer to a TIME WITH TIMEZONE value from a range bound.
- const TimeTzADT * **getTimeTzPtrUp** (size_t idx)
- const TimeTzADT & getTimeTzRefLo (size_t idx)

  Get a reference to a TIME WITH TIMEZONE value from a range bound.
- const TimeTzADT & **getTimeTzRefUp** (size_t idx)
- bool hasBounds (size_t idx)

  Check if this range has lower and upper bounds set.
- bool isNull (int idx)

  Check if all values in the idx'th input range are NULL.

Protected Types

- enum **BoundType** { **LO_BOUND**, **UP_BOUND** }

Protected Member Functions

- void addArg (char *loBound, char *upBound, const VerticaType &dt, EE::ValueSort sortedness, bool canHaveNulls)
- void setCanHaveNulls (size_t idx, bool u)

  Set a flag to indicate that some values in this range can be NULL.
- void setSortedness (size_t idx, EE::ValueSort s)

  Set the sortedness of values in the range.

Protected Attributes

- size_t **narg**
- std::vector< ValueRange > **ranges**
- std::vector< VString > **svWrappersLo**
- std::vector< VString > **svWrappersUp**
- std::vector< VNumeric > **vnWrappersLo**
- std::vector< VNumeric > **vnWrappersUp**

Friends

- class **EE::VEval**

Detailed Description

This class represents the value ranges of the arguments of a UDSF, one range per argument.

Instances of this class are used to let UDSF developers specify the output range of UDSFs via the optional ScalarFunction::getOutputRange() function.

Member Function Documentation

void Vertica::VerticaValueRange::addArg (char * *loBound,* char * *upBound,* const VerticaType & *dt,* EE::ValueSort *sortedness,* bool *canHaveNulls*) `[inline],[protected],[inherited]`

Add a value range of a particular function argument

Parameters

loBound	Base location to find the lower bound data
upBound	Base location to find the upper bound data
sortedness	Sortedness of values in the range
dt	The data type of range bounds

bool Vertica::VerticaValueRange::canHaveNulls (size_t *idx*) `[inline],[inherited]`

Indicates if there can be NULL values in the range.

Parameters

idx	the range argument number.

Returns

TRUE if some range values can be NULL, else FALSE.

const vbool* Vertica::ValueRangeReader::getBoolPtrLo (size_t *idx*) `[inline]`

Get a pointer to a BOOLEAN value from a range bound.

Parameters

idx	The argument number to retrieve the range bound.

Returns

A pointer to the retrieved value cast as a BOOLEAN.

Referenced by getBoolRefLo().

const vbool& Vertica::ValueRangeReader::getBoolRefLo (size_t *idx*) `[inline]`

Get a reference to a BOOLEAN value from a range bound.

Parameters

idx	The argument number to retrieve the range bound.

Returns

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

Referenced by isNull().

const DateADT* Vertica::ValueRangeReader::getDatePtrLo (size_t *idx*) `[inline]`

Get a pointer to a DATE value from a range bound.

Parameters

idx	The argument number to retrieve the range bound.

Returns

A pointer to the retrieved value cast as a DATE.

Referenced by getDateRefLo().

const DateADT& Vertica::ValueRangeReader::getDateRefLo (size_t *idx*) `[inline]`

Get a reference to a DATE value from a range bound.

Parameters

idx	The argument number to retrieve the range bound.

Returns

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

Referenced by isNull().

const vfloat* Vertica::ValueRangeReader::getFloatPtrLo (size_t *idx*) `[inline]`

Get a pointer to a FLOAT value from a range bound.

Parameters

idx	The argument number to retrieve the range bound.

Returns

A pointer to the retrieved value cast as a FLOAT.

Referenced by getFloatRefLo().

const vfloat& Vertica::ValueRangeReader::getFloatRefLo (size_t *idx*) `[inline]`

Get a reference to a FLOAT value from a range bound.

Parameters

idx	The argument number to retrieve the range bound.

Returns

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

Referenced by isNull().

const Interval* Vertica::ValueRangeReader::getIntervalPtrLo (size_t *idx*) `[inline]`

Get a pointer to an INTERVAL value from a range bound.

Parameters

idx	The argument number to retrieve the range bound.

Returns

A pointer to the retrieved value cast as an INTERVAL.

Referenced by getIntervalRefLo().

const Interval& Vertica::ValueRangeReader::getIntervalRefLo (size_t *idx*) `[inline]`

Get a reference to an INTERVAL value from a range bound.

Parameters

idx	The argument number to retrieve the range bound.

Returns

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

Referenced by isNull().

const IntervalYM* Vertica::ValueRangeReader::getIntervalYMPtrLo (size_t *idx*) `[inline]`

Get a pointer to a INTERVAL YEAR TO MONTH value from a range bound.

Parameters

idx	The argument number to retrieve the range bound.

Returns

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

Referenced by getIntervalYMRefLo().

const IntervalYM& Vertica::ValueRangeReader::getIntervalYMRefLo (size_t *idx*) `[inline]`

Get a reference to an INTERVAL YEAR TO MONTH value from a range bound.

Parameters

idx	The argument number to retrieve the range bound.

Returns

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

Referenced by isNull().

const vint* Vertica::ValueRangeReader::getIntPtrLo (size_t *idx*) `[inline]`

Get a pointer to an INTEGER value from a range bound.

Parameters

idx	The argument number to retrieve the range bound.

Returns

a pointer to a chosen bound value, cast appropriately.

Example:

```
* const vint *a = range->getIntPtrLo(0); // gets a pointer to the lower bound of 1st arg
*
```

Referenced by getIntRefLo().

const vint& Vertica::ValueRangeReader::getIntRefLo (size_t *idx*) `[inline]`

Get a reference to an INTEGER value from a range bound.

Parameters

idx	The argument number to retrieve the range bound.

Returns

a reference to the chosen bound value, cast as INTEGER.

Example:

```
* const vint& a = range->getIntRefUp(0); // get upper bound of the 1st range argument
*
```

Referenced by isNull().

const VNumeric* Vertica::ValueRangeReader::getNumericPtrLo (size_t *idx*) `[inline]`

Get a pointer to a VNumeric value from a range bound.

Parameters

idx	The argument number to retrieve the range bound.

Returns

A pointer to the retrieved value cast as a Numeric.

Referenced by getNumericRefLo().

const VNumeric& Vertica::ValueRangeReader::getNumericRefLo (size_t *idx*) `[inline]`

Get a reference to a VNumeric value from a range bound.

Parameters

idx	The argument number to retrieve the range bound.

Returns

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

Referenced by isNull().

size_t Vertica::VerticaValueRange::getNumRanges ( ) const `[inline], [inherited]`

Retrieve the number of range arguments.

Returns

the number of range arguments held by this object.

const VerticaType& Vertica::VerticaValueRange::getRangeType (size_t *idx*) const `[inline], [inherited]`

Returns the data type of the values in a range.

Parameters

idx	The index of the range

Returns

a VerticaType object describing the data type of the range values.

EE::ValueSort Vertica::VerticaValueRange::getSortedness (size_t *idx*) `[inline], [inherited]`

Gets the sortedness of values in a range.

Parameters

idx	the range argument number.

Returns

the range sortedness. Possible values are: EE::SORT_UNORDERED - Unsorted EE::SORT_MONOTONIC-_INCREASING - Ascending EE::SORT_MONOTONIC_DECREASING - Descending EE::SORT_CONSTANT - Single value

const VString* Vertica::ValueRangeReader::getStringPtrLo (size_t *idx*) `[inline]`

Get a pointer to a VString value from a range bound.

Parameters

idx	The argument number to retrieve the range bound.

Returns

A pointer to the retrieved value cast as a VString.

Referenced by getStringRefLo().

const VString& Vertica::ValueRangeReader::getStringRefLo (size_t *idx*) [inline]

Get a reference to an VString value from a range bound.

Parameters

idx	The argument number to retrieve the range bound.

Returns

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

Note

the returned object is a variable-length prefix of the actual string values in the range. For example, if the range values are {'abc', 'abb', 'bc', 'cab'} the range bounds may be like: Lo='a' and Up='cb'. In the example, the upper bound is not even a prefix of any value in the range, but all values in the range are greater than or equal than *Lo* and less than or equal to *Up*.

Referenced by isNull().

const TimeADT* Vertica::ValueRangeReader::getTimePtrLo (size_t *idx*) [inline]

Get a pointer to a TIME value from a range bound.

Parameters

idx	The argument number to retrieve the range bound.

Returns

A pointer to the retrieved value cast as a TIME.

Referenced by getTimeRefLo().

const TimeADT& Vertica::ValueRangeReader::getTimeRefLo (size_t *idx*) [inline]

Get a reference to a TIME value from a range bound.

Parameters

idx	The argument number to retrieve the range bound.

Returns

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

Referenced by isNull().

const Timestamp* Vertica::ValueRangeReader::getTimestampPtrLo (size_t *idx*) `[inline]`

Get a pointer to a TIMESTAMP value from a range bound.

Parameters

idx	The argument number to retrieve the range bound.

Returns

A pointer to the retrieved value cast as a TIMESTAMP.

Referenced by getTimestampRefLo().

const Timestamp& Vertica::ValueRangeReader::getTimestampRefLo (size_t *idx*) `[inline]`

Get a reference to a TIMESTAMP value from a range bound.

Parameters

idx	The argument number to retrieve the range bound.

Returns

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

Referenced by isNull().

const TimestampTz* Vertica::ValueRangeReader::getTimestampTzPtrLo (size_t *idx*) `[inline]`

Get a pointer to a TIMESTAMP WITH TIMEZONE value from a range bound.

Parameters

idx	The argument number to retrieve the range bound.

Returns

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

Referenced by getTimestampTzRefLo().

const TimestampTz& Vertica::ValueRangeReader::getTimestampTzRefLo (size_t *idx*) `[inline]`

Get a reference to a TIMESTAMP WITH TIMEZONE value from a range bound.

Parameters

idx	The argument number to retrieve the range bound.

Returns

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

Referenced by isNull().

const TimeTzADT* Vertica::ValueRangeReader::getTimeTzPtrLo (size_t *idx*) `[inline]`

Get a pointer to a TIME WITH TIMEZONE value from a range bound.

Parameters

idx	The argument number to retrieve the range bound.

Returns

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

Referenced by getTimeTzRefLo().

const TimeTzADT& Vertica::ValueRangeReader::getTimeTzRefLo (size_t *idx*) [inline]

Get a reference to a TIME WITH TIMEZONE value from a range bound.

Parameters

idx	The argument number to retrieve the range bound.

Returns

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

Referenced by isNull().

bool Vertica::ValueRangeReader::hasBounds (size_t *idx*) [inline]

Check if this range has lower and upper bounds set.

Note

Callers of the get reference or get pointer functions should always first call this method, and only access the range bounds when this function returns TRUE.

Example:

```
*   if (rangeReader.hasBounds(0)) { // Check if bounds of the 1st range argument are set
*     // OK to access bounds
*     const vint& i = rangeReader.getIntRefLo(0);
*     const vint& j = rangeReader.getIntRefUp(0);
*   }
*
```

Returns

TRUE if the range bounds are safe to access, else FALSE.

bool Vertica::ValueRangeReader::isNull (int *idx*) [inline]

Check if all values in the idx'th input range are NULL.

Parameters

idx	The argument range number.

Returns

true if all values in the range are NULL, false otherwise.

void Vertica::VerticaValueRange::setCanHaveNulls (size_t *idx,* bool *u*) [inline], [protected], [inherited]

Set a flag to indicate that some values in this range can be NULL.

Parameters

idx	The argument range number.
u	true if there can be NULL values in the range, else false.

void Vertica::VerticaValueRange::setSortedness (size_t *idx*, EE::ValueSort *s*) `[inline], [protected], [inherited]`

Set the sortedness of values in the range.

Parameters

idx	The range argument number.
s	The sortedness value to set. Possible values are: EE::SORT_UNORDERED - Unsorted EE::SORT_MONOTONIC_INCREASING - Ascending EE::SORT_MONOTONIC_DECREASING - Descending EE::SORT_CONSTANT - Single value

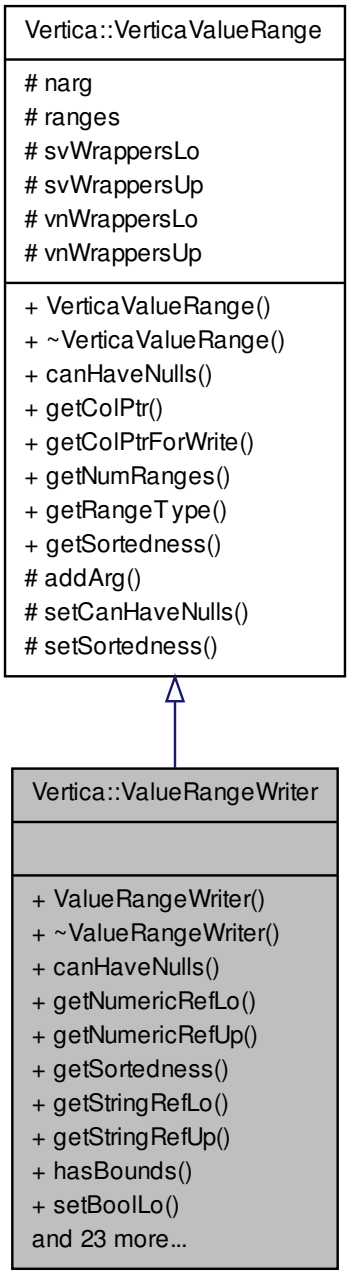
Example:

```
* range.setSortedness(0, EE:SORT_CONSTANT); // defines a single-valued range
*
```

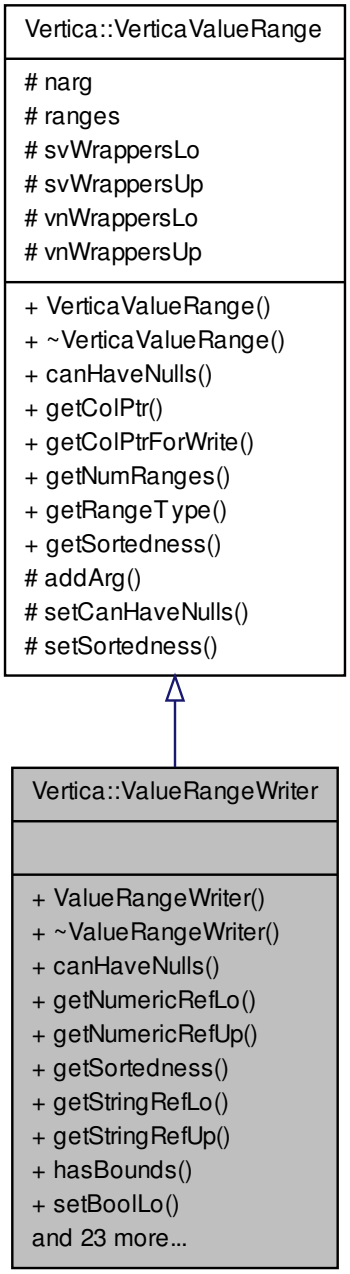
Vertica::ValueRangeWriter Class Reference

This class represents the output value range of a UDSF.

Inheritance diagram for Vertica::ValueRangeWriter:



Collaboration diagram for Vertica::ValueRangeWriter:



Public Member Functions

- **ValueRangeWriter** (char ∗outLoBound, char ∗outUpBound, const VerticaType &dt, EE::ValueSort sortedness, bool canHaveNulls)
- bool canHaveNulls (size_t idx)

  Indicates if there can be NULL values in the range.
- bool **canHaveNulls** ()

- template<class T , BoundType b>
  const T * **getColPtr** (size_t idx)
- template<class T , BoundType b>
  T * **getColPtrForWrite** (size_t idx)
- VNumeric & getNumericRefLo ()

  Gets a VNumeric object reference to set the range lower bound.
- VNumeric & getNumericRefUp ()

  Gets a VNumeric object reference to set the range upper bound.
- size_t getNumRanges () const

  Retrieve the number of range arguments.
- const VerticaType & getRangeType (size_t idx) const

  Returns the data type of the values in a range.
- EE::ValueSort getSortedness (size_t idx)

  Gets the sortedness of values in a range.
- EE::ValueSort **getSortedness** ()
- VString & getStringRefLo ()

  Gets a VString object reference to set the range lower bound.
- VString & getStringRefUp ()

  Gets a VString object reference to set the range upper bound.
- bool **hasBounds** ()
- void setBoolLo (vbool r)

  Sets a range bound as a BOOLEAN value.
- void **setBoolUp** (vbool r)
- void **setCanHaveNulls** (bool u)
- void setDateLo (DateADT r)

  Sets a range bound as a DATE value.
- void **setDateUp** (DateADT r)
- void setFloatLo (vfloat r)

  Sets a range bound as a FLOAT value to the output row.
- void **setFloatUp** (vfloat r)
- void setHasBounds ()

  Lets Vertica know that this output range has user-defined bounds.
- void setIntervalLo (Interval r)

  Sets a range bound as an INTERVAL value.
- void **setIntervalUp** (Interval r)
- void setIntervalYMLo (IntervalYM r)

  Sets a range bound as an INTERVAL YEAR TO MONTH value.
- void **setIntervalYMUp** (IntervalYM r)
- void setIntLo (vint r)

  Sets a range bound as an INTEGER value.
- void **setIntUp** (vint r)
- void setNull ()

  Sets to NULL all values in this range.
- void **setSortedness** (EE::ValueSort s)
- void setTimeLo (TimeADT r)

  Sets a range bound as a TIMESTAMP value.
- void setTimestampLo (Timestamp r)

  Sets a range bound as a TIMESTAMP value.
- void setTimestampTzLo (TimestampTz r)

  Sets a range bound as a TIMESTAMP WITH TIMEZONE value.
- void **setTimestampTzUp** (TimestampTz r)
- void **setTimestampUp** (Timestamp r)

- void setTimeTzLo (TimeTzADT r)

  Sets a range bound as a TIMESTAMP WITH TIMEZONE value.
- void **setTimeTzUp** (TimeTzADT r)
- void **setTimeUp** (TimeADT r)

Protected Types

- enum **BoundType** { **LO_BOUND**, **UP_BOUND** }

Protected Member Functions

- void addArg (char *loBound, char *upBound, const VerticaType &dt, EE::ValueSort sortedness, bool canHaveNulls)
- void setCanHaveNulls (size_t idx, bool u)

  Set a flag to indicate that some values in this range can be NULL.
- void setSortedness (size_t idx, EE::ValueSort s)

  Set the sortedness of values in the range.

Protected Attributes

- size_t **narg**
- std::vector< ValueRange > **ranges**
- std::vector< VString > **svWrappersLo**
- std::vector< VString > **svWrappersUp**
- std::vector< VNumeric > **vnWrappersLo**
- std::vector< VNumeric > **vnWrappersUp**

Friends

- class **EE::VEval**

Detailed Description

This class represents the output value range of a UDSF.

Instances of this class are used to allow UDSF developers specify the output range of UDSFs via the optional ScalarFunction::getOutputRange() function.

Member Function Documentation

void Vertica::VerticaValueRange::addArg (char * *loBound,* char * *upBound,* const VerticaType & *dt,* EE::ValueSort *sortedness,* bool *canHaveNulls*) `[inline],[protected],[inherited]`

Add a value range of a particular function argument

Parameters

loBound	Base location to find the lower bound data
upBound	Base location to find the upper bound data

sortedness	Sortedness of values in the range
dt	The data type of range bounds

bool Vertica::VerticaValueRange::canHaveNulls (size_t *idx*) `[inline], [inherited]`

Indicates if there can be NULL values in the range.

Parameters

idx	the range argument number.

Returns

TRUE if some range values can be NULL, else FALSE.

VNumeric& Vertica::ValueRangeWriter::getNumericRefLo ( ) `[inline]`

Gets a VNumeric object reference to set the range lower bound.

Returns

A VNumeric object reference.

Referenced by setNull().

VNumeric& Vertica::ValueRangeWriter::getNumericRefUp ( ) `[inline]`

Gets a VNumeric object reference to set the range upper bound.

Returns

A VNumeric object reference.

Referenced by setNull().

size_t Vertica::VerticaValueRange::getNumRanges ( ) const `[inline], [inherited]`

Retrieve the number of range arguments.

Returns

the number of range arguments held by this object.

const VerticaType& Vertica::VerticaValueRange::getRangeType (size_t *idx*) const `[inline], [inherited]`

Returns the data type of the values in a range.

Parameters

idx	The index of the range

Returns

a VerticaType object describing the data type of the range values.

EE::ValueSort Vertica::VerticaValueRange::getSortedness (size_t *idx*) `[inline],[inherited]`

Gets the sortedness of values in a range.

Parameters

idx	the range argument number.

Returns

the range sortedness. Possible values are: EE::SORT_UNORDERED - Unsorted EE::SORT_MONOTONIC-_INCREASING - Ascending EE::SORT_MONOTONIC_DECREASING - Descending EE::SORT_CONSTANT - Single value

VString& Vertica::ValueRangeWriter::getStringRefLo ( ) [inline]

Gets a VString object reference to set the range lower bound.

Returns

A VString object reference.

Referenced by setNull().

VString& Vertica::ValueRangeWriter::getStringRefUp ( ) [inline]

Gets a VString object reference to set the range upper bound.

Returns

A VString object reference.

Referenced by setNull().

void Vertica::ValueRangeWriter::setBoolLo (vbool *r*) [inline]

Sets a range bound as a BOOLEAN value.

Parameters

r	The BOOLEAN value to set the range bound.

Referenced by setNull().

void Vertica::VerticaValueRange::setCanHaveNulls (size_t *idx*, bool *u*) [inline], [protected], [inherited]

Set a flag to indicate that some values in this range can be NULL.

Parameters

idx	The argument range number.
u	true if there can be NULL values in the range, else false.

void Vertica::ValueRangeWriter::setDateLo (DateADT *r*) [inline]

Sets a range bound as a DATE value.

Parameters

r	The DATE value to set the range bound.

Referenced by setNull().

void Vertica::ValueRangeWriter::setFloatLo (vfloat *r*) `[inline]`

Sets a range bound as a FLOAT value to the output row.

Parameters

r	The FLOAT value to set the range bound.

Referenced by setNull().

void Vertica::ValueRangeWriter::setIntervalLo (Interval *r*) `[inline]`

Sets a range bound as an INTERVAL value.

Parameters

r	The INTERVAL value to set the range bound.

Referenced by setNull().

void Vertica::ValueRangeWriter::setIntervalYMLo (IntervalYM *r*) `[inline]`

Sets a range bound as an INTERVAL YEAR TO MONTH value.

Parameters

r	The INTERVAL YEAR TO MONTH value to set the range bound.

void Vertica::ValueRangeWriter::setIntLo (vint *r*) `[inline]`

Sets a range bound as an INTEGER value.

Parameters

r	The INTEGER value to set the range bound.

Referenced by setNull().

void Vertica::ValueRangeWriter::setNull ( ) `[inline]`

Sets to NULL all values in this range.

Note

As side effect of this method, this range will be marked as having NULL values and its sortedness set to EE::SORT_CONSTANT.

void Vertica::VerticaValueRange::setSortedness (size_t *idx,* EE::ValueSort *s*) `[inline], [protected], [inherited]`

Set the sortedness of values in the range.

Parameters

idx	The range argument number.
s	The sortedness value to set. Possible values are: EE::SORT_UNORDERED - Unsorted EE::SORT_MONOTONIC_INCREASING - Ascending EE::SORT_MONOTONIC_DECREASING - Descending EE::SORT_CONSTANT - Single value

Example:

```
* range.setSortedness(0, EE:SORT_CONSTANT); // defines a single-valued range
*
```

void Vertica::ValueRangeWriter::setTimeLo (TimeADT *r*) `[inline]`

Sets a range bound as a TIMESTAMP value.

Parameters

r	The TIMESTAMP value to set the range bound.

Referenced by setNull().

void Vertica::ValueRangeWriter::setTimestampLo (Timestamp *r*) `[inline]`

Sets a range bound as a TIMESTAMP value.

Parameters

r	The TIMESTAMP value to set the range bound.

Referenced by setNull().

void Vertica::ValueRangeWriter::setTimestampTzLo (TimestampTz *r*) `[inline]`

Sets a range bound as a TIMESTAMP WITH TIMEZONE value.

Parameters

r	The TIMESTAMP WITH TIMEZONE value to set the range bound.

Referenced by setNull().

void Vertica::ValueRangeWriter::setTimeTzLo (TimeTzADT *r*) `[inline]`

Sets a range bound as a TIMESTAMP WITH TIMEZONE value.

Parameters

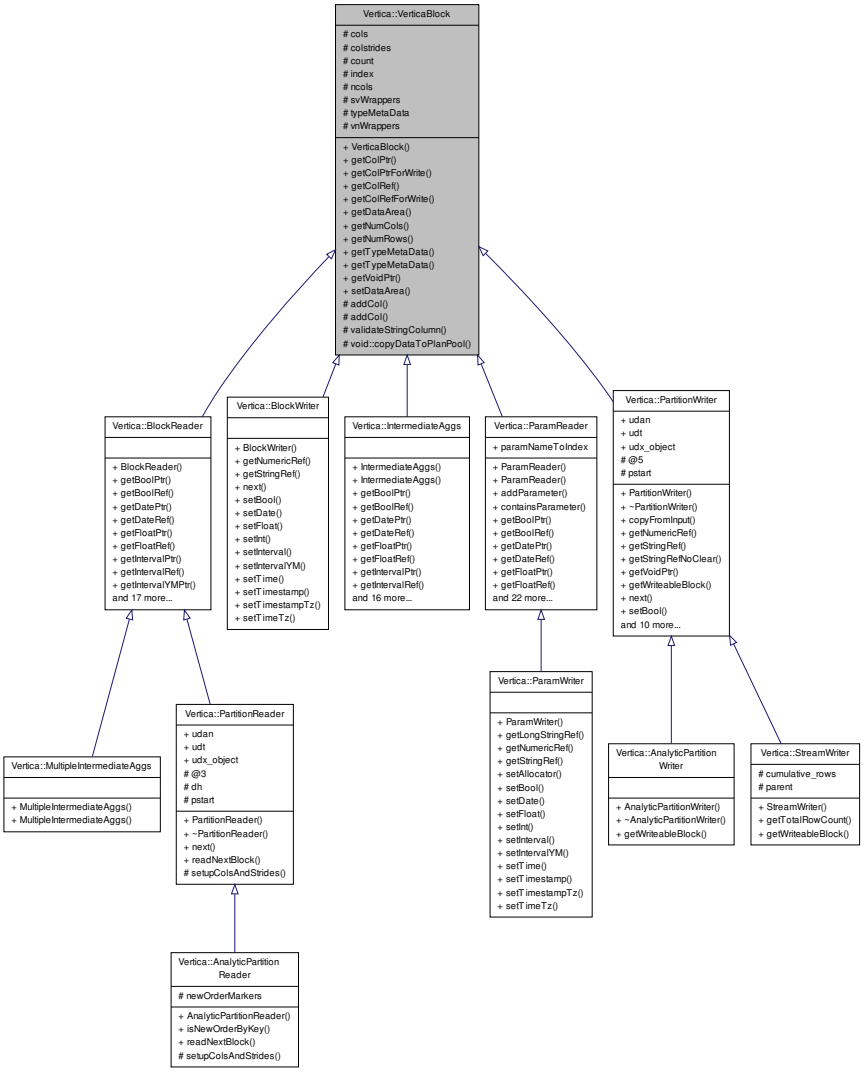
r	The TIMESTAMP WITH TIMEZONE value to set the range bound.

Referenced by setNull().

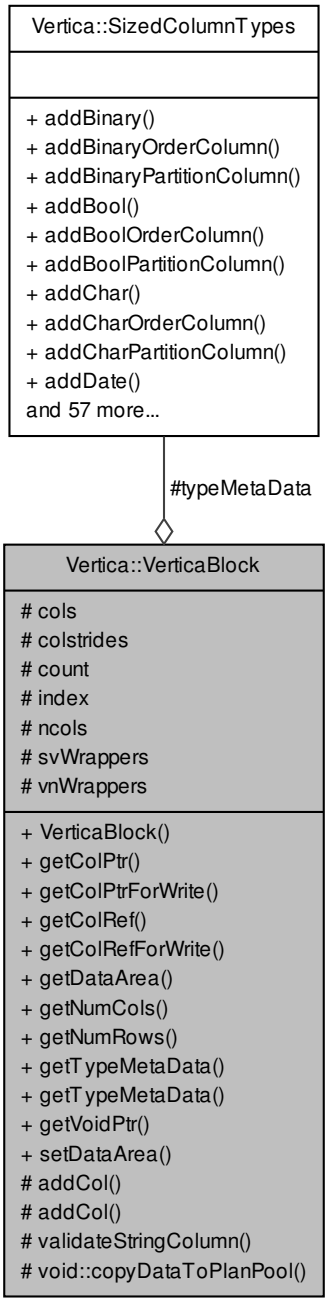
Vertica::VerticaBlock Class Reference

: Represents an in-memory block of tuples

Inheritance diagram for Vertica::VerticaBlock:



Collaboration 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)
- const EE::DataArea * **getDataArea** (size_t idx)
- size_t getNumCols () const
- int getNumRows () const
- const SizedColumnTypes & getTypeMetaData () const
- SizedColumnTypes & getTypeMetaData ()
- void * **getVoidPtr** ()
- void **setDataArea** (size_t idx, void *dataarea)

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)
- friend **void::copyDataToPlanPool** (VerticaBlock *block)

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::UserDefinedAggregate**
- class **::EE::UserDefinedAnalytic**
- class **::EE::UserDefinedProcess**
- class **::EE::UserDefinedTransform**
- class **AggregateFunction**
- struct **CPPExecContext**
- 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

Referenced by Vertica::ParamReader::addParameter().

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);
*
```

Referenced by Vertica::PartitionWriter::copyFromInput().

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.

Referenced by Vertica::BlockReader::isNull().

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

Referenced by Vertica::PartitionWriter::copyFromInput(), Vertica::ParamReader::getType(), Vertica::BlockReader::isNull(), and Vertica::PartitionWriter::setNull().

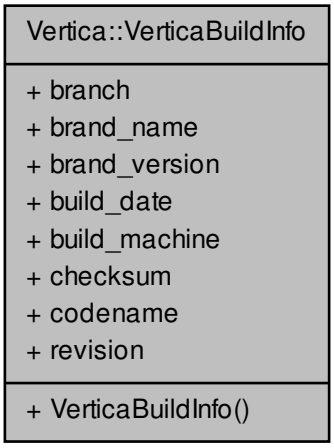
SizedColumnTypes& Vertica::VerticaBlock::getTypeMetaData ( ) `[inline]`

Returns

information about the types and numbers of arguments

Vertica::VerticaBuildInfo Struct Reference

Collaboration diagram for Vertica::VerticaBuildInfo:



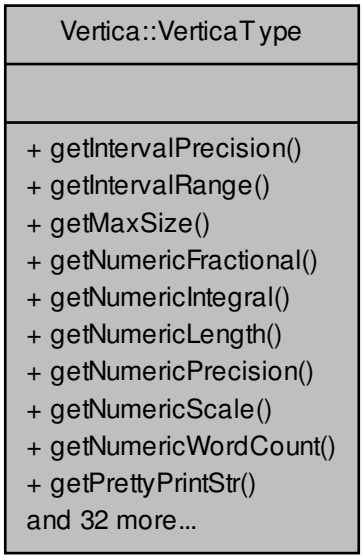
Public Attributes

- 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.

Collaboration diagram for Vertica::VerticaType:



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.

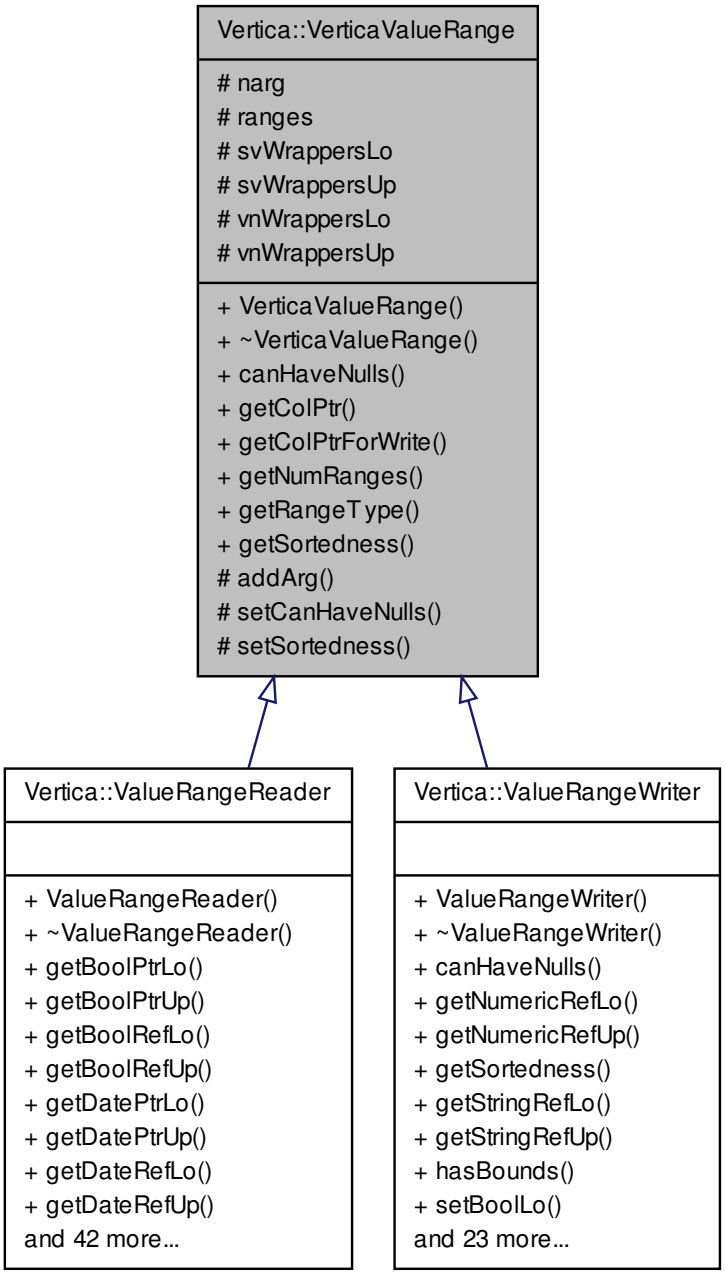
Note: This function is designed so that the common case (aka a built in, non user defined type) is fast – calling isUDType() is a heavyweight operation. – See VER-24673 for an example of when it matters.

Referenced by getMaxSize(), Vertica::ValueRangeReader::isNull(), isStringType(), and Vertica::ValueRangeWriter::setNull().

Vertica::VerticaValueRange Class Reference

This class represents value ranges used in analyzing the output of UDSFs. A range is expressed as a minimum/maximum value (inclusive) pair.

Inheritance diagram for Vertica::VerticaValueRange:



Collaboration diagram for Vertica::VerticaValueRange:

Classes

- struct ValueRange

Public Member Functions

- **VerticaValueRange** (size_t narg)
- bool canHaveNulls (size_t idx)

  Indicates if there can be NULL values in the range.
- template<class T , BoundType b>
  const T * **getColPtr** (size_t idx)
- template<class T , BoundType b>
  T * **getColPtrForWrite** (size_t idx)
- size_t getNumRanges () const

  Retrieve the number of range arguments.
- const VerticaType & getRangeType (size_t idx) const

  Returns the data type of the values in a range.
- EE::ValueSort getSortedness (size_t idx)

  Gets the sortedness of values in a range.

Protected Types

- enum **BoundType** { **LO_BOUND**, **UP_BOUND** }

Protected Member Functions

- void addArg (char *loBound, char *upBound, const VerticaType &dt, EE::ValueSort sortedness, bool canHaveNulls)
- void setCanHaveNulls (size_t idx, bool u)

  Set a flag to indicate that some values in this range can be NULL.
- void setSortedness (size_t idx, EE::ValueSort s)

  Set the sortedness of values in the range.

Protected Attributes

- size_t **narg**
- std::vector< ValueRange > **ranges**
- std::vector< VString > **svWrappersLo**
- std::vector< VString > **svWrappersUp**
- std::vector< VNumeric > **vnWrappersLo**
- std::vector< VNumeric > **vnWrappersUp**

Detailed Description

This class represents value ranges used in analyzing the output of UDSFs. A range is expressed as a minimum/maximum value (inclusive) pair.

Instances of this class are used to allow UDSF developers to specify what the output range of of the function is. The UDSF developer is responsible to override ScalarFunction::getOutputRange() and set the function's output range using the knowledge of the input argument ranges passed as references to ScalarFunction::getOutputRange().

Member Function Documentation

void Vertica::VerticaValueRange::addArg (char * *loBound*, char * *upBound*, const VerticaType & *dt*, EE::ValueSort *sortedness*, bool *canHaveNulls*) `[inline],[protected]`

Add a value range of a particular function argument

Parameters

loBound	Base location to find the lower bound data
upBound	Base location to find the upper bound data
sortedness	Sortedness of values in the range
dt	The data type of range bounds

bool Vertica::VerticaValueRange::canHaveNulls (size_t *idx*) `[inline]`

Indicates if there can be NULL values in the range.

Parameters

idx	the range argument number.

Returns

TRUE if some range values can be NULL, else FALSE.

size_t Vertica::VerticaValueRange::getNumRanges ( ) const `[inline]`

Retrieve the number of range arguments.

Returns

the number of range arguments held by this object.

const VerticaType& Vertica::VerticaValueRange::getRangeType (size_t *idx*) const `[inline]`

Returns the data type of the values in a range.

Parameters

idx	The index of the range

Returns

a VerticaType object describing the data type of the range values.

EE::ValueSort Vertica::VerticaValueRange::getSortedness (size_t *idx*) `[inline]`

Gets the sortedness of values in a range.

Parameters

idx	the range argument number.

Returns

the range sortedness. Possible values are: EE::SORT_UNORDERED - Unsorted EE::SORT_MONOTONIC-_INCREASING - Ascending EE::SORT_MONOTONIC_DECREASING - Descending EE::SORT_CONSTANT - Single value

void Vertica::VerticaValueRange::setCanHaveNulls (size_t *idx,* bool *u*) `[inline],[protected]`

Set a flag to indicate that some values in this range can be NULL.

Parameters

idx	The argument range number.
u	true if there can be NULL values in the range, else false.

void Vertica::VerticaValueRange::setSortedness (size_t *idx,* EE::ValueSort *s*) `[inline],[protected]`

Set the sortedness of values in the range.

Parameters

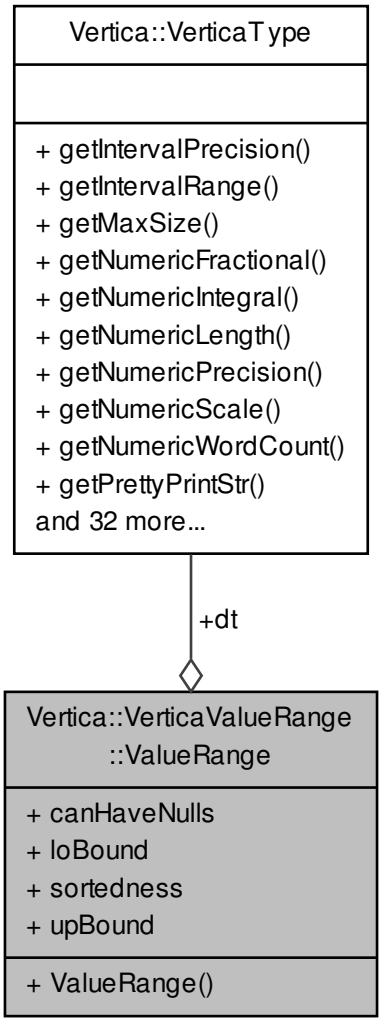
idx	The range argument number.
s	The sortedness value to set. Possible values are: EE::SORT_UNORDERED - Unsorted EE::SORT_MONOTONIC_INCREASING - Ascending EE::SORT_MONOTONIC_DECREASING - Descending EE::SORT_CONSTANT - Single value

Example:

```
* range.setSortedness(0, EE:SORT_CONSTANT); // defines a single-valued range
*
```

Vertica::VerticaValueRange::ValueRange Struct Reference

Collaboration diagram for Vertica::VerticaValueRange::ValueRange:



Public Member Functions

- **ValueRange** (char *loBound, char *upBound, VerticaType dt, EE::ValueSort sortedness, bool canHaveNulls)

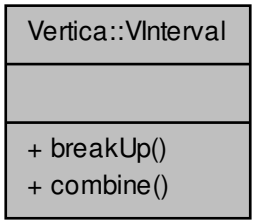
Public Attributes

- bool **canHaveNulls**
- VerticaType **dt**
- char * **loBound**
- EE::ValueSort **sortedness**
- char * **upBound**

Vertica::VInterval Class Reference

Representation of an Interval in Vertica.

Collaboration diagram for Vertica::VInterval:



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.
- static Interval combine (int64 days, int64 hour, int64 min, double sec)

  Compute an Interval from its components.

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.

static Interval Vertica::VInterval::combine (int64 *days,* int64 *hour,* int64 *min,* double *sec*) `[inline],[static]`

Compute an Interval from its components.

Returns

the value of the specified Interval

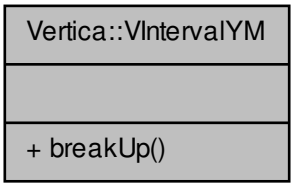
Parameters

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.

Collaboration diagram for Vertica::VIntervalYM:



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

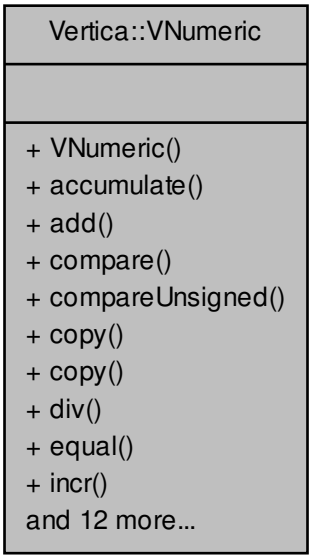
Parameters

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.

Collaboration diagram for Vertica::VNumeric:



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]

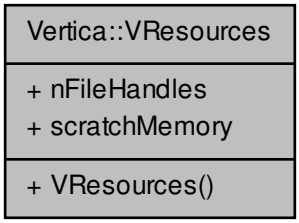
Convert the VNumeric value into floating-point.

Returns

the value in 80-bit floating-point

Vertica::VResources Struct Reference

Collaboration diagram for Vertica::VResources:



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

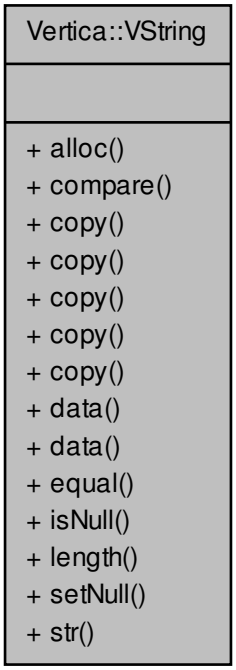
Amount of RAM in bytes used by User defined function

Referenced by Vertica::UDLFactory::getPerInstanceResources().

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.

Collaboration diagram for Vertica::VString:



Public Member Functions

- void alloc (vsize len)

  Allocate the VString's internal buffer and initialize to 'len' 0 bytes.
- 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

void Vertica::VString::alloc (vsize *len*) `[inline]`

Allocate the VString's internal buffer and initialize to 'len' 0 bytes.

The VString is allocated and set to 'len' zero bytes. It is the caller's responsibility to not provide a value of 'len' that is larger than the maximum size of this VString

Parameters

len	length in bytes

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

Referenced by Vertica::PartitionWriter::copyFromInput().

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

Referenced by copy().

void Vertica::VString::copy (const std::string & *s*) `[inline]`

Copy character data from std::string.

Parameters

s	null-terminated character input data

Referenced by copy().

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

Referenced by copy().

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

Referenced by alloc(), and str().

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

Referenced by copy(), Vertica::BlockReader::isNull(), and str().

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.

Referenced by str().

std::string Vertica::VString::str ( ) const `[inline]`

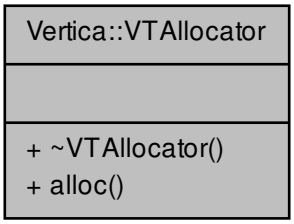
Provides a copy of this VString's data as a std::string.

Returns

a std::string copy of the data in this VString

Vertica::VTAllocator Class Reference

Collaboration diagram for Vertica::VTAllocator:



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]`

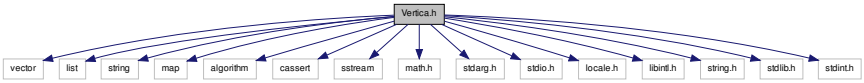
Allocate size_t bytes of memory on a pool. This memory is guaranteed to persist beyond the destroy call but might have been destroyed when the dtor is run.

File Documentation

Vertica.h File Reference

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

Include dependency graph for Vertica.h:



Classes

- struct Vertica::LibraryRegistrar
- struct Vertica::UDxRegistrar

Namespaces

- Vertica

Macros

- #define **__Linux64__**
- #define InlineAggregate()
- #define RegisterFactory(clazz)
- #define **RegisterLibrary**(author, library_build_tag, library_version, library_sdk_version, source_url, description, licenses_required, signature) Vertica::LibraryRegistrar registrar(author, library_build_tag, library_version, library_sdk_version, source_url, description, licenses_required, signature)

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);

Index