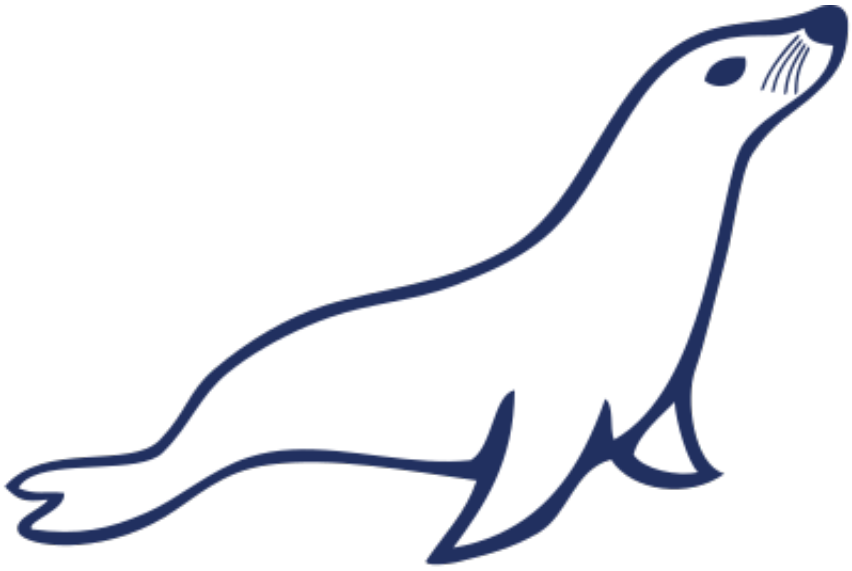

MariaDB ColumnStore C++ API Usage Documentation

Release 1.1.4-0cf0875

MariaDB Corporation

May 31, 2018

CONTENTS

CHAPTER

ONE

LICENSING

1.1 Documentation Content

The mcsapi documentation is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

1.2 MariaDB ColumnStore C++ API

The MariaDB ColumnStore C++ API (mcsapi) is licensed under the GNU Lesser General Public License, version 2.1.

CHAPTER

TWO

VERSION HISTORY

This is a version history of C++ API interface changes. It does not include internal fixes and changes.

CHAPTER

THREE

USING MCSAPI

3.1 Usage Introduction

The bulk insert API is designed to provide a very rapid way to insert data into ColumnStore tables. It can be seen as a large transaction which appends data to the end of the column extent files. The commit is atomic by moving the high water mark for the column extents. A rollback is also very fast as it will undo any uncommitted high water mark changes and remove the new extents.

Whilst a bulk insert API is happening a table lock is gained on the table being written to. During this time the table is effectively read-only. Any DML statements are likely to timeout waiting.

It can be run from any point in the ColumnStore installation as it reads the Columnstore.xml file to discover where the PMs are. It can also be run remotely if the Columnstore.xml file is present and the ColumnStore installation was configured using hostnames/IPs accessible by the machine running the API.

Methods in *ColumnStoreBulkInsert* are not guaranteed to thread safe so an instance of that class should stay in a single thread. But each instance of the class can be run in separate threads.

3.2 Basic Bulk Insert

In this example we will insert 1000 rows of two integer values into table `test.t1`. The full code for this can be found in the `example/basic_bulk_insert.cpp` file in the mcsapi codebase.

You will need the following table in the test database to execute this:

Listing 1: example/basic_bulk_insert.sql

```
CREATE TABLE `t1` (
  `a` int(11) DEFAULT NULL,
  `b` int(11) DEFAULT NULL
) ENGINE=Columnstore;
```

Listing 2: example/basic_bulk_insert.cpp

```
#include <libmcsapi/mcsapi.h>
#include <iostream>
```

We need to include `libmcsapi/mcsapi.h` which is the main include file for mcsapi. This will pull in all the required includes to use mcsapi.

Listing 3: example/basic_bulk_insert.cpp

```
28 int main(void)
29 {
30     mcsapi::ColumnStoreDriver* driver = nullptr;
31     mcsapi::ColumnStoreBulkInsert* bulk = nullptr;
```

A pointer is needed to *ColumnStoreDriver* to get the configuration information about the ColumnStore installation and one to `ColumnStoreBulk` to hold the class for the bulk insert.

Listing 4: example/basic_bulk_insert.cpp

```
32     try {
33         driver = new mcsapi::ColumnStoreDriver();
34         bulk = driver->createBulkInsert("test", "t1", 0, 0);
```

A new instance of *ColumnStoreDriver* is created which will attempt to find the `Columnstore.xml` configuration file by first searching for the environment variable *COLUMNSTORE_INSTALL_DIR* and then the default path of `/usr/local/mariadb/columnstore/etc/Columnstore.xml`. Alternatively we could provide a path as a parameter to *ColumnStoreDriver*.

Once we have the ColumnStore installation's configuration in the driver we use this to initiate a bulk insert using *ColumnStoreDriver::createBulkInsert()*. We are using the `test` database and the `t1` table. The remaining two parameters are unused for now and set to `0`.

Listing 5: example/basic_bulk_insert.cpp

```
35         for (int i = 0; i < 1000; i++)
36         {
37                 bulk->setColumn(0, (uint32_t)i);
38                 bulk->setColumn(1, (uint32_t)1000 - i);
39                 bulk->writeRow();
40         }
41         bulk->commit();
```

A "for" loop is used to loop over 1000 arbitrary inserts in this example. We use *ColumnStoreBulkInsert::setColumn()* to specify that column `0` (column `a`) should be set to the integer from the "for" loop and column `1` (column `b`) is set to `1000` minus the integer from the "for" loop.

When we have added something to every column *ColumnStoreBulkInsert::writeRow()* is used to indicate we are finished with the row. The library won't necessarily write the row at this stage, it buffers up to 100,000 rows by default.

At the end of the loop we execute *ColumnStoreBulkInsert::commit()* which will send any final rows and initiate the commit of the data. If we do not do this the transaction will be implicitly rolled back instead.

Listing 6: example/basic_bulk_insert.cpp

```
42     } catch (mcsapi::ColumnStoreError &e) {
43         std::cout << "Error caught: " << e.what() << std::endl;
44     }
45     delete bulk;
46     delete driver;
47 }
```

If anything fails then we should catch `ColumnStoreException` to handle it. Finally we should delete our bulk insert class and driver class instances.

3.3 Advanced Bulk Insert

In this example we will insert 2 rows in a more complex table. This will demonstrate using different kinds of data types, chanined methods and getting the summary information at the end of a transaction.

You will need the following table in the test database to execute this:

Listing 7: example/advanced_bulk_insert.sql

```
CREATE TABLE `t2` (
  `id` int(11) DEFAULT NULL,
  `name` varchar(40) DEFAULT NULL,
  `dob` date DEFAULT NULL,
  `added` datetime DEFAULT NULL,
  `salary` decimal(9,2) DEFAULT NULL
) ENGINE=Columnstore;
```

Listing 8: example/advanced_bulk_insert.cpp

```
#include <libmcsapi/mcsapi.h>
#include <iostream>

int main(void)
{
    mcsapi::ColumnStoreDriver* driver = nullptr;
    mcsapi::ColumnStoreBulkInsert* bulk = nullptr;
    try {
        driver = new mcsapi::ColumnStoreDriver();
        bulk = driver->createBulkInsert("test", "t2", 0, 0);
```

As with the basic example we create an instance of the driver and use it to create a bulk insert instance.

Listing 9: example/advanced_bulk_insert.cpp

```
                bulk->setColumn(0, 1);
                bulk->setColumn(1, "Andrew");
        bulk->setColumn(2, "1936-12-24");
        bulk->setColumn(3, "2017-07-07 15:14:12");
        bulk->setColumn(4, "15239.45");
                bulk->writeRow();
```

This demonstrates setting several different data types using strings of data. The second column (column `1`) is a `VARCHAR(40)` and is set to "Andrew". The third column is a `DATE` column and the API will automatically convert this into a binary date format before transmitting it to ColumnStore. The fourth is a `DATETIME` and the fifth a `DECIMAL` column which again the API will convert from the strings into the binary format.

Listing 10: example/advanced_bulk_insert.cpp

```
        mcsapi::ColumnStoreDateTime dob;
        dob.set("1972-05-23", "%Y-%m-%d");
        mcsapi::ColumnStoreDateTime added;
        added.set("2017-07-07 15:20:18", "%Y-%m-%d %H:%M:%S");
```

`ColumnStoreDateTime` is used to create an entry for a `DATE` or `DATETIME` column. It can be used to define custom formats for dates and times using the strptime format.

Listing 11: example/advanced_bulk_insert.cpp

```
45          mcsapi::ColumnStoreDecimal salary;
46          salary.set(2347623, 2);
```

A decimal is created using the *ColumnStoreDecimal* class. It can be set using a string, `double` or a pair of integers. The first integer is the precision and the second integer is the scale. So this number becomes `23476.23`.

Listing 12: example/advanced_bulk_insert.cpp

```
47          bulk->setColumn(0, 2)->setColumn(1, "David")->setColumn(2, dob)
48              ->setColumn(3, added)->setColumn(4, salary)->writeRow();
49          bulk->commit();
```

Many of the *ColumnStoreBulkInsert* methods return a pointer to the class which means multiple calls can be chained together in a similar way to ORM APIs. Here we can also see the dates and decimal we set earlier are applied.

Listing 13: example/advanced_bulk_insert.cpp

```
50          mcsapi::ColumnStoreSummary summary = bulk->getSummary();
51          std::cout << summary.getRowsInsertedCount() << " inserted in " <<
52              summary.getExecutionTime() << " seconds" << std::endl;
```

After a commit or rollback we can obtain summary information from the bulk insert class. This is done using the *ColumnStoreBulkInsert::getSummary()* method which will return a reference *ColumnStoreSummary* class. In this example we get the number of rows inserted (or would be inserted if there was a rollback) and the execution time from the moment the bulk insert class is created until the commit or rollback is complete.

Listing 14: example/advanced_bulk_insert.cpp

```
53      } catch (mcsapi::ColumnStoreError &e) {
54          std::cout << "Error caught: " << e.what() << std::endl;
55      }
56      delete bulk;
57      delete driver;
58  }
```

At the end we clean up in the same was as the basic bulk insert example.

CHAPTER

FOUR

COMPILING WITH MCSAPI

4.1 Pre-requisites

To link mcsapi to your application you first need install the following pre-requisites:

4.1.1 Ubuntu

```
sudo apt-get install libsnappy1v5 libuv1 libxml2 g++ gcc pkg-config libbost-dev
```

4.1.2 CentOS 7

```
sudo yum install snappy libuv libxml2 pkgconfig boost-devel
sudo yum install centos-release-scl
sudo yum install devtoolset-4-gcc*
scl enable devtoolset-4 bash
```

4.2 Compiling

The easiest way to compile is to use pkg-config to provide the required compile options.

The following is a basic example of how to do compile an example c++ application with mcsapi:

```
g++ example.cpp -o example -std=c++11 `pkg-config libmcsapi --cflags --libs`
```

CHAPTER

FIVE

MCSAPI API REFERENCE

All API calls are with the `mcsapi` namespace.

5.1 ColumnStoreDriver Class

COLUMNSTORE_INSTALL_DIR
: The optional environment variable containing the path to the ColumnStore installation. Used by *ColumnStoreDriver*

class ColumnStoreDriver
: This is the parent class for mcsapi. It uses the `Columnstore.xml` file to discover the layout of the ColumnStore cluster. It therefore needs to be able to discover the path to the ColumnStore installation.

5.1.1 ColumnStoreDriver()

ColumnStoreDriver`::ColumnStoreDriver()`
: Creates an instance of the ColumnStoreDriver. This will search for the environment variable *COLUMNSTORE_INSTALL_DIR*, if this isn't found then the default path of `/usr/local/mariadb/columnstore/` is used.

Raises ColumnStoreConfigError When the Columnstore.xml file cannot be found or cannot be parsed

Example

```
#include <iostream>
#include <libmcsapi/mcsapi.h>

int main(void)
{
    mcsapi::ColumnStoreDriver* driver = nullptr;
    try {
        driver = new mcsapi::ColumnStoreDriver();
    } catch (mcsapi::ColumnStoreError &e) {
        std::cout << "Error caught " << e.what() << std::endl;
    }
    delete driver;
    return 0;
}
```

ColumnStoreDriver`::ColumnStoreDriver`(`const` std::string &*path*)

Creates an instance of `ColumnStoreDriver` using the specified path to the Columnstore.xml file (including filename).

Parameters **`path`** – The path to the Columnstore.xml (including filename)

Raises **ColumnStoreConfigError** When the Columnstore.xml file cannot be found or cannot be parsed

Example

```
#include <iostream>
#include <libmcsapi/mcsapi.h>

int main(void)
{
    mcsapi::ColumnStoreDriver* driver = nullptr;
    try {
        driver = new mcsapi::ColumnStoreDriver("/usr/local/mariadb/columnstore/etc/
↪Columnstore.xml");
    } catch (mcsapi::ColumnStoreError &e) {
        std::cout << "Error caught " << e.what() << std::endl;
    }
    delete driver;
    return 0;
}
```

5.1.2 createBulkInsert()

ColumnStoreBulkInsert **ColumnStoreDriver*`::createBulkInsert`(`const` std::string &*db*, `const` std::string &*table*, uint8_t *mode*, uint16_t *pm*)

Allocates and configures an instance of *`ColumnStoreBulkInsert`* to be used for bulk inserts with the ColumnStore installation reference by the driver. The resulting object should be freed by the application using the library.

Parameters

- **`db`** – The database name for the table to insert into
- **`table`** – The tabe name to insert into
- **`mode`** – Future use, must be set to `0`
- **`pm`** – Future use, must be set to `0`. For now batches of inserts use a round-robin between the PM servers.

Returns An instance of *`ColumnStoreBulkInsert`*

Raises **ColumnStoreServerError** If a table lock cannot be acquired for the desired table

Example

```
#include <iostream>
#include <libmcsapi/mcsapi.h>

```

(continues on next page)

(continued from previous page)

```
int main(void)
{
    std::string table("t1");
    std::string db("test");
    mcsapi::ColumnStoreDriver* driver = nullptr;
    mcsapi::ColumnStoreBulkInsert* bulkInsert = nullptr;
    try {
        driver = new mcsapi::ColumnStoreDriver();
        bulkInsert = driver->createBulkInsert(db, table, 0, 0);
    } catch (mcsapi::ColumnStoreError &e) {
        std::cout << "Error caught " << e.what() << std::endl;
    }
    delete bulkInsert;
    delete driver;
    return 0;
}
```

5.1.3 getVersion()

const char ***ColumnStoreDriver*`::`**`getVersion`**`()`

Returns the version of the library in the format `1.0.0-0393456-dirty` where `1.0.0` is the version number, `0393456` is the short git tag and `dirty` signifies there is uncommitted code making up this build.

Returns The version string

Example

```
#include <iostream>
#include <libmcsapi/mcsapi.h>

int main(void)
{
    try {
        mcsapi::ColumnStoreDriver* driver = new mcsapi::ColumnStoreDriver();
        const char* version = driver->getVersion();
        std::cout << version << std::endl;
    } catch (mcsapi::ColumnStoreError &e) {
        std::cout << "Error caught: " << e.what() << std::endl;
    }
    return 0;
}
```

5.1.4 setDebug()

void *ColumnStoreDriver*`::`**`setDebug`**`(`bool *enabled*`)`

Enables/disables verbose debugging output which is sent to stderr upon execution.

Note: This is a global setting which will apply to all instances of all of the API's classes after it is set until it is turned off.

Parameters **`enabled`** – Set to `true` to enable and `false` to disable.

Example

```
#include <iostream>
#include <libmcsapi/mcsapi.h>

int main(void)
{
    try {
        mcsapi::ColumnStoreDriver* driver = new mcsapi::ColumnStoreDriver();
        driver->setDebug(true);
        // Debugging output is now enabled
    } catch (mcsapi::ColumnStoreError &e) {
        std::cout << "Error caught: " << e.what() << std::endl;
    }
    return 0;
}
```

5.1.5 getSystemCatalog()

ColumnStoreSystemCatalog &*ColumnStoreDriver*`::getSystemCatalog()`

Returns an instance of the ColumnStore system catalog which contains all of the ColumnStore table and column details

Returns The system catalog

Example

```
#include <iostream>
#include <libmcsapi/mcsapi.h>

int main(void)
{
    try {
        mcsapi::ColumnStoreDriver* driver = new mcsapi::ColumnStoreDriver();
        mcsapi::ColumnStoreSystemCatalog sysCat = driver->getSystemCatalog();

        mcsapi::ColumnStoreSystemCatalogTable tbl = sysCat.getTable("test", "t1");
        std::cout << "t1 has " << tbl.getColumnCount() << " columns" << endl;

        mcsapi::ColumnStoreSystemCatalogColumn col1 = tbl.getColumn(0);
        std::cout << "The first column in t1 is " << col1.getColumnName() << endl;
    } catch (mcsapi::ColumnStoreError &e) {
        std::cout << "Error caught: " << e.what() << std::endl;
    }
    return 0;
}
```

5.2 ColumnStoreBulkInsert Class

`class ColumnStoreBulkInsert`

The bulk insert class is designed to rapidly insert data into a ColumnStore installation.

Note: An instance of this class should only be created from *ColumnStoreDriver*

Note: If an explicit commit is not given before the class is destroyed then an implicit rollback will be executed

Note: This class should be viewed as a single transaction. Once committed or rolled back the class cannot be used for any more operations beyond getting the summary. Further usage attempts will result in an exception being thrown.

5.2.1 getColumnCount()

uint16_t *ColumnStoreBulkInsert*::**getColumnCount**()

Gets the number of columns in the table to be inserted into.

Returns A count of the number of columns

Example

This example can be used inside the try...catch blocks in the *ColumnStoreDriver* examples.

```
...
driver = new mcsapi::ColumnStoreDriver();
bulkInsert = driver->createBulkInsert(db, table, 0, 0);
// columnCount will now contain the number of columns in the table
uint16_t columnCount = bulkInsert->getColumnCount();
...
```

5.2.2 setColumn()

ColumnStoreBulkInsert **setColumn**(uint16_t *columnNumber*, **const** std::string &*value*, *columnstore_data_convert_status_t* *status = nullptr)

ColumnStoreBulkInsert **setColumn**(uint16_t *columnNumber*, uint64_t *value*, *columnstore_data_convert_status_t* *status = nullptr)

ColumnStoreBulkInsert **setColumn**(uint16_t *columnNumber*, int64_t *value*, *columnstore_data_convert_status_t* *status = nullptr)

ColumnStoreBulkInsert **setColumn**(uint16_t *columnNumber*, uint32_t *value*, *columnstore_data_convert_status_t* *status = nullptr)

ColumnStoreBulkInsert **setColumn**(uint16_t *columnNumber*, int32_t *value*, *columnstore_data_convert_status_t* *status = nullptr)

ColumnStoreBulkInsert **setColumn**(uint16_t *columnNumber*, uint16_t *value*, *columnstore_data_convert_status_t* *status = nullptr)

ColumnStoreBulkInsert ***ColumnStoreBulkInsert::setColumn**(uint16_t *columnNumber*, int16_t *value*, *columnstore_data_convert_status_t* **status* = nullptr)

ColumnStoreBulkInsert ***ColumnStoreBulkInsert::setColumn**(uint16_t *columnNumber*, uint8_t *value*, *columnstore_data_convert_status_t* **status* = nullptr)

ColumnStoreBulkInsert ***ColumnStoreBulkInsert::setColumn**(uint16_t *columnNumber*, int8_t *value*, *columnstore_data_convert_status_t* **status* = nullptr)

ColumnStoreBulkInsert ***ColumnStoreBulkInsert::setColumn**(uint16_t *columnNumber*, double *value*, *columnstore_data_convert_status_t* **status* = nullptr)

ColumnStoreBulkInsert ***ColumnStoreBulkInsert::setColumn**(uint16_t *columnNumber*, *ColumnStoreDateTime* &*value*, *columnstore_data_convert_status_t* **status* = nullptr)

ColumnStoreBulkInsert ***ColumnStoreBulkInsert::setColumn**(uint16_t *columnNumber*, *ColumnStoreDecimal* &*value*, *columnstore_data_convert_status_t* **status* = nullptr)

Sets a value for a given column.

Parameters

- **columnNumber** – The column number to set (starting from `0`)
- **value** – The value to set this column
- **status** – An optional pointer to a user supplied *columnstore_data_convert_status_t* type. If supplied this will be set to the resulting status of any data conversion required.

Returns A pointer to the *ColumnStoreBulkInsert* class so that calls can be chained

Raises ColumnStoreDataError If there is an error setting the column, such as truncation error when *ColumnStoreBulkInsert::setTruncateIsError()* is used or an invalid column number is supplied

Raises ColumnStoreUsageError If the transaction has already been closed

Example

This example can be used inside the try...catch blocks in the *ColumnStoreDriver* examples.

```
...
driver = new mcsapi::ColumnStoreDriver();
bulkInsert = driver->createBulkInsert(db, table, 0, 0);

// Create a decimal value
ColumnStoreDecimal decimalVal;
decimalVal.set("3.14159");

// And an int value
uint32_t intVal = 123456;

// And a string value
```

(continues on next page)

(continued from previous page)

```
std::string strVal("Hello World");

// Finally a date/time values
ColumnStoreDateTime dateTime;
std::string newTime("1999-01-01 23:23:23");
std::string tFormat("%Y-%m-%d %H:%M:%S");
dateTime.set(newTime, tFormat);

// A status variable so we can check all is good
mcsapi::columnstore_data_convert_status_t status;

bulkInsert->setColumn(0, intVal, &status);
// Check conversion status
if (status != CONVERT_STATUS_NONE)
     return 1;
bulkInsert->setColumn(1, decimalVal, &status);
// Check conversion status
if (status != CONVERT_STATUS_NONE)
     return 1;
bulkInsert->setColumn(2, strVal, &status);
// Check conversion status
if (status != CONVERT_STATUS_NONE)
     return 1;
bulkInsert->setColumn(3, dateTime, &status);
// Check conversion status
if (status != CONVERT_STATUS_NONE)
     return 1;

// Write this row ready to start another
bulkInsert->writeRow();

decimalVal.set("1.41421");
intVal = 654321;
strVal = "dlroW olleH";
newTime = "2017-07-05 22:00:43";
dateTime.set(newTime, tFormat);

// A chained example
bulkInsert->setColumn(0, intVal)->setColumn(1, decimalVal)->setColumn(2, strVal)->
↪setColumn(3, dateTime)->writeRow();
...
```

5.2.3 setNull()

ColumnStoreBulkInsert ***ColumnStoreBulkInsert**::**setNull**(uint16_t *columnNumber*, *columnstore_data_convert_status_t* **status* = nullptr)

Sets a `NULL` for a given column.

Parameters

- **columnNumber** – The column number to set (starting from `0`)
- **status** – An optional pointer to a user supplied *columnstore_data_convert_status_t* type. If supplied this will be set to the resulting status of any data conversion required.

Returns A pointer to the *ColumnStoreBulkInsert* class so that calls can be chained

Raises ColumnStoreDataError If there is an error setting the column, such as an invalid column number is supplied

Raises ColumnStoreUsageError If the transaction has already been closed

Example

This example can be used inside the try...catch blocks in the *ColumnStoreDriver* examples.

```
...
driver = new mcsapi::ColumnStoreDriver();
bulkInsert = driver->createBulkInsert(db, table, 0, 0);

// Set an whole row of NULLs
bulkInsert->setNull(0)->setNull(1)->setNull(2)->setNull(3)->writeRow();
...
```

5.2.4 resetRow()

ColumnStoreBulkInsert ***ColumnStoreBulkInsert*::resetRow()**

Resets everything that has been set for the current row. This method should be used to clear the row memory without using *ColumnStoreBulkInsert::writeRow()*.

Raises ColumnStoreUsageError If the transaction has already been closed

5.2.5 writeRow()

ColumnStoreBulkInsert ***ColumnStoreBulkInsert*::writeRow()**

States that a row is ready to be written.

Note: The row may not be written at this stage. The library will batch an amount of rows together before sending them, by default data is only sent to the server every 100,000 rows or *ColumnStoreBulkInsert::commit()* is called. Data is not committed with `writeRow()`, it has to be explicitly committed at the end of the transaction.

Raises ColumnStoreNetworkError If there has been an error during the write at the network level

Raises ColumnStoreServerError If there has been an error during the write at the remote server level

Raises ColumnStoreUsageError If the transaction has already been closed

Example

This example can be used inside the try...catch blocks in the *ColumnStoreDriver* examples.

```
...
driver = new mcsapi::ColumnStoreDriver();
bulkInsert = driver->createBulkInsert(db, table, 0, 0);

// Set values for a 2 int column table
```

(continues on next page)

(continued from previous page)

```
bulkInsert->setValue(0, (uint32_t) 123456);
bulkInsert->setValue(1, (uint32_t) 654321);

// Write the row
bulkInsert->writeRow();
...
```

5.2.6 commit()

void *ColumnStoreBulkInsert*::**commit**()

Commits the data to the table.

Note: After making this call the transaction is completed and the class should not be used for anything but *ColumnStoreBulkInsert::getSummary()* or *ColumnStoreBulkInsert::isActive()*. Attempts to use it again will trigger an exception.

Note: If the commit fails a rollback will be executed automatically upon deletion of the *ColumnStoreBulkInsert* object.

Raises ColumnStoreNetworkError If there has been an error during the write at the network level

Raises ColumnStoreServerError If there has been an error during the write at the remote server level

Raises ColumnStoreUsageError If the transaction has already been closed

Example

This example can be used inside the try...catch blocks in the *ColumnStoreDriver* examples.

```
...
driver = new mcsapi::ColumnStoreDriver();
bulkInsert = driver->createBulkInsert(db, table, 0, 0);

// Set values for a 2 int column table
bulkInsert->setValue(0, (uint32_t) 123456);
bulkInsert->setValue(1, (uint32_t) 654321);

// Write the row
bulkInsert->writeRow();

// Commit the transaction
bulkInsert->commit();

// This WILL throw an exception if uncommented
// bulkInsert->setValue(0, (uint32_t) 99999);
...
```

5.2.7 rollback()

void *ColumnStoreBulkInsert*::**rollback**()

Rolls back the data written to the table. If the transaction has already been committed or rolled back this will just return without error.

Note: After making this call the transaction is completed and the class should not be used for anything but *ColumnStoreBulkInsert::getSummary()* or *ColumnStoreBulkInsert::isActive()*. Attempts to use it again will trigger an exception.

Raises ColumnStoreNetworkError If there has been an error during the write at the network level

Raises ColumnStoreServerError If there has been an error during the write at the remote server level

Example

This example can be used inside the try...catch blocks in the *ColumnStoreDriver* examples.

```
...
driver = new mcsapi::ColumnStoreDriver();
bulkInsert = driver->createBulkInsert(db, table, 0, 0);

// Set values for a 2 int column table
bulkInsert->setValue(0, (uint32_t) 123456);
bulkInsert->setValue(1, (uint32_t) 654321);

// Write the row
bulkInsert->writeRow();

// Rollback the transaction
bulkInsert->rollback();

// This WILL throw an exception if uncommented
// bulkInsert->setValue(0, (uint32_t) 99999);
...
```

5.2.8 isActive()

bool *ColumnStoreBulkInsert*::**isActive**()

Returns whether or not the bulk insert transaction is still active.

Returns true if the transaction is still active, false if it has been committed or rolled back

5.2.9 getSummary()

ColumnStoreSummary &*ColumnStoreBulkInsert*::**getSummary**()

Gets the summary information for this bulk write transaction.

Returns The summary object

Example

This example can be used inside the try... catch blocks in the *ColumnStoreDriver* examples.

```
...
driver = new mcsapi::ColumnStoreDriver();
bulkInsert = driver->createBulkInsert(db, table, 0, 0);

// Set values for a 2 int column table
bulkInsert->setValue(0, (uint32_t) 123456);
bulkInsert->setValue(1, (uint32_t) 654321);

// Write the row
bulkInsert->writeRow();

// Rollback the transaction
bulkInsert->rollback();

// Get the summary
ColumnStoreSummary summary = bulkInsert->getSummary();

// Get the number of inserted rows before they were rolled back
uint64_t rows = summary.getRowsInsertedCount();
...
```

5.2.10 setTruncateIsError()

void *ColumnStoreBulkInsert*::**setTruncateIsError**(bool *set*)

Sets whether or not a truncation of CHAR/VARCHAR data is an error. It is disabled by default.

Parameters **set** – true to enable, false to disable

Example

This example can be used inside the try... catch blocks in the *ColumnStoreDriver* examples.

```
...
driver = new mcsapi::ColumnStoreDriver();
bulkInsert = driver->createBulkInsert(db, table, 0, 0);

bulkInsert->setTruncateIsError(true);
std::string strVal("Short string");

// A short string that will insert fine
bulkInsert->setValue(0, strVal);

// This long string will truncate on my VARCHAR(20) and throw an exception
strVal = "This is a long string test to demonstrate setTruncateIsError()";
bulkInsert->setValue(1, strVal);
...
```

5.2.11 setBatchSize()

void *ColumnStoreBulkInsert*::**setBatchSize**(uint32_t *batchSize*)
Future use, this has not been implemented yet

5.3 ColumnStoreSummary Class

class ColumnStoreSummary
A class containing the summary information for a transaction. An instance of this should be obtained from *ColumnStoreBulkInsert::getSummary()*.

5.3.1 getExecutionTime()

double *ColumnStoreSummary*::**getExecutionTime**()
Returns the total time for the transaction in seconds, from creation of the *ColumnStoreBulkInsert* class until commit or rollback.

Returns The total execution time in seconds

Example

This example can be used inside the try...catch blocks in the *ColumnStoreDriver* examples.

```
...
driver = new mcsapi::ColumnStoreDriver();
bulkInsert = driver->createBulkInsert(db, table, 0, 0);

// Set values for a 2 int column table
bulkInsert->setValue(0, (uint32_t) 123456);
bulkInsert->setValue(1, (uint32_t) 654321);

// Write the row
bulkInsert->writeRow();

// Rollback the transaction
bulkInsert->rollback();

// Get the summary
ColumnStoreSummary summary = bulkInsert->getSummary();

// Get the execution time for the transaction
double execTime = summary.getExecutionTime();
...
```

5.3.2 getRowsInsertedCount()

uint64_t *ColumnStoreSummary*::**getRowsInsertedCount**()
Returns the number of rows inserted during the transaction or failed to insert for a rollback.

Returns The total number of rows

Example

This example can be used inside the try... catch blocks in the *ColumnStoreDriver* examples.

```
...
driver = new mcsapi::ColumnStoreDriver();
bulkInsert = driver->createBulkInsert(db, table, 0, 0);

// Set values for a 2 int column table
bulkInsert->setValue(0, (uint32_t) 123456);
bulkInsert->setValue(1, (uint32_t) 654321);

// Write the row
bulkInsert->writeRow();

// Rollback the transaction
bulkInsert->rollback();

// Get the summary
ColumnStoreSummary summary = bulkInsert->getSummary();

// Get the number of inserted rows before they were rolled back
uint64_t rows = summary.getRowsInsertedCount();
...
```

5.3.3 getTruncationCount()

uint64_t *ColumnStoreSummary*::**getTruncationCount**()

Returns the number of truncated CHAR/VARCHAR values during the transaction.

Returns The total number of truncated values

Example

This example can be used inside the try... catch blocks in the *ColumnStoreDriver* examples.

```
...
driver = new mcsapi::ColumnStoreDriver();
bulkInsert = driver->createBulkInsert(db, table, 0, 0);

std::string strVal("Short string");

// A short string that will insert fine
bulkInsert->setValue(0, strVal);

// This long string will truncate on my VARCHAR(20)
strVal = "This is a long string test to demonstrate a truncation";
bulkInsert->setValue(1, strVal);

// Get the summary
ColumnStoreSummary summary = bulkInsert->getSummary();

// Get the number of truncated values before they were rolled back
uint64_t truncateCount = summary.getTruncationCount();
...
```

5.3.4 getSaturatedCount()

uint64_t *ColumnStoreSummary*`::getSaturatedCount()`
Returns the number of saturated values during the transaction.

Returns The total number of saturated values

Example

This example can be used inside the try...catch blocks in the *ColumnStoreDriver* examples.

```
...
driver = new mcsapi::ColumnStoreDriver();
bulkInsert = driver->createBulkInsert(db, table, 0, 0);

// Set values for a 2 int column table
bulkInsert->setValue(0, (uint32_t) 123456);
// Slightly higher than a signed int max, this will saturate
bulkInsert->setValue(1, (uint32_t) 2147483650);

// Write the row
bulkInsert->writeRow();

// Rollback the transaction
bulkInsert->rollback();

// Get the summary
ColumnStoreSummary summary = bulkInsert->getSummary();

// Get the number of saturated values before they were rolled back
uint64_t saturatedCount = summary.getSaturatedCount();
...
```

5.3.5 getInvalidCount()

uint64_t *ColumnStoreSummary*`::getInvalidCount()`
Returns the number of invalid values during the transaction.

Note: An invalid value is one where a data conversion during `ColumnStoreBulkInsert::setValue()` was not possible. When this happens a `0` or empty string is used instead and the status value set accordingly.

Returns The total number of invalid values

Example

This example can be used inside the try...catch blocks in the *ColumnStoreDriver* examples.

```
...
driver = new mcsapi::ColumnStoreDriver();
bulkInsert = driver->createBulkInsert(db, table, 0, 0);

// Set values for a 2 int column table
```

(continues on next page)

(continued from previous page)

```
bulkInsert->setValue(0, (uint32_t) 123456);
// This is a DATE column, which is invalid to set as a date.
// The result will be the date set to '0000-00-00'
// and a invalid counter increment
bulkInsert->setValue(1, (uint32_t) 123456);

// Write the row
bulkInsert->writeRow();

// Rollback the transaction
bulkInsert->rollback();

// Get the summary
ColumnStoreSummary summary = bulkInsert->getSummary();

// Get the number of invalid values before they were rolled back
uint64_t invalidCount = summary.getInvalidCount();
...
```

5.4 ColumnStoreException Class

class ColumnStoreError : **public** std::runtime_error
A general execption for the mcsapi classes. It should be used just as the `std::runtime_error` exception is used. It can be used as a "catchall" for all ColumnStore exceptions

class ColumnStoreInternalError : **public** *ColumnStoreError*
An execption class covering problems with the driver's internals

class ColumnStoreBufferError : **public** *ColumnStoreError*
An execption class covering problems with the driver's buffers

class ColumnStoreServerError : **public** *ColumnStoreError*
An execption class that contains errors received from the server

class ColumnStoreNetworkError : **public** *ColumnStoreError*
An execption class covering problems with network connection and communication

class ColumnStoreDataError : **public** *ColumnStoreError*
An execption class covering problems with setting column data and data conversion

class ColumnStoreUsageError : **public** *ColumnStoreError*
An execption class covering problems with usage of the driver

class ColumnStoreConfigError : **public** *ColumnStoreError*
An execption class covering problems with reading the XML configuration

class ColumnStoreVersionError : **public** *ColumnStoreError*
An execption class for an incompatible ColumnStore version number

class ColumnStoreNotFound : **public** *ColumnStoreError*
An execption class to signify a table or column was not found

5.5 columnstore_data_convert_status_t Type

type columnstore_data_convert_status_t
The status value used in *ColumnStoreBulkInsert::setColumn()* and *ColumnStoreBulkInsert::setNull()* to signify the status of any data conversion that occurred during setting.

type CONVERT_STATUS_NONE
There was no problems during the conversion or no conversion.

type CONVERT_STATUS_SATURATED
The value was saturated during the conversion, the maximum/minimum was used instead.

type CONVERT_STATUS_INVALID
The value was invalid during the conversion, 0 or empty string was used instead.

type CONVERT_STATUS_TRUNCATED
The value was truncated.

5.6 columnstore_data_types_t Type

type columnstore_data_types_t
The data type as returned by *ColumnStoreSystemCatalogColumn::getType()*.

type DATA_TYPE_BIT
BIT data type

type DATA_TYPE_TINYINT
TINYINT data type

type DATA_TYPE_CHAR
CHAR data type

type DATA_TYPE_SMALLINT
SMALLINT data type

type DATA_TYPE_DECIMAL
DECIMAL data type

type DATA_TYPE_MEDINT
MEDIUMINT data type

type DATA_TYPE_INT
INT data type

type DATA_TYPE_FLOAT
FLOAT data type

type DATA_TYPE_DATE
DATE data type

type DATA_TYPE_BIGINT
BIGINT data type

type DATA_TYPE_DOUBLE
DOUBLE data type

type DATA_TYPE_DATETIME
DATETIME data type

type DATA_TYPE_VARCHAR
VARCHAR data type

type DATA_TYPE_VARBINARY
VARBINARY data type

type DATA_TYPE_CLOB
Unused

type DATA_TYPE_BLOB
BLOB data type

type DATA_TYPE_UTINYINT
UNSIGNED TINYINT data type

type DATA_TYPE_USMALLINT
UNSIGNED SMALLINT data type

type DATA_TYPE_UDECIMAL
UNSIGNED DECIMAL data type

type DATA_TYPE_UMEDINT
UNSIGNED MEDIUMINT data type

type DATA_TYPE_UINT
UNSIGNED INT data type

type DATA_TYPE_UFLOAT
UNSIGNED FLOAT data type

type DATA_TYPE_UBIGINT
UNSIGNED BIGINT data type

type DATA_TYPE_UDOUBLE
UNSIGNED DOUBLE data type

type DATA_TYPE_TEXT
TEXT data type

5.7 ColumnStoreDateTime Class

class ColumnStoreDateTime
A class which is used to contain a date/time used to set `DATE` or `DATETIME` columns using *`ColumnStoreBulkInsert::setColumn()`*

5.7.1 ColumnStoreDateTime()

ColumnStoreDateTime`::`**`ColumnStoreDateTime`**`()`
Sets the date/time to `0000-00-00 00:00:00`.

ColumnStoreDateTime`::`**`ColumnStoreDateTime`**`(`tm *&time*`)`
Sets the date/time the value of the `tm` struct.

Parameters **`time`** – The date/time to set

Raises **ColumnStoreDataError** When an invalid date or time is supplied

ColumnStoreDateTime`::`**`ColumnStoreDateTime`**`(`**`const`** std::string *&dateTime*, **`const`** std::string *&format*`)`
Sets the date/time based on a given string and format.

Parameters

- **dateTime** – A string containing the date/time to set
- **format** – The format specifier for the date/time string. This uses the strptime format.

Raises ColumnStoreDataError When an invalid date or time is supplied

ColumnStoreDateTime::**ColumnStoreDateTime**(uint32_t *year*, uint32_t *month*, uint32_t *day*, uint32_t *hour*, uint32_t *minute*, uint32_t *second*, uint32_t *microsecond*)

Sets the date/time based on a given set of intergers

Note: Microseconds are for future usage, they are not currently supported in ColumnStore.

Parameters

- **year** – The year
- **month** – The month of year
- **day** – The day of month
- **hour** – The hour
- **minute** – The minute
- **second** – The second
- **microsecond** – The microseconds

Raises ColumnStoreDataError When an invalid date or time is supplied

5.7.2 set()

bool *ColumnStoreDateTime*::**set**(tm &*time*)

Sets the date/time using the value of the `tm` struct.

Parameters **time** – The date/time to set

Returns `true` if the date/time is valid, `false` if it is not

bool *ColumnStoreDateTime*::**set**(**const** std::string &*dateTime*, **const** std::string &*format*)

Sets the date/time based on a given string and format.

Parameters

- **dateTime** – A string containing the date/time to set
- **format** – The format specifier for the date/time string. This uses the strptime format.

Returns `true` if the date/time is valid, `false` if it is not

5.8 ColumnStoreDecimal Class

class ColumnStoreDecimal

A class which is used to contain a non-lossy decimal format used to set `DECIMAL` columns using *ColumnStoreBulkInsert::setColumn()*.

5.8.1 ColumnStoreDecimal()

ColumnStoreDecimal`::ColumnStoreDecimal()`
Sets the decimal to `0`.

ColumnStoreDecimal`::ColumnStoreDecimal` (int64_t *value*)
Sets the decimal to an supplied integer value.

Parameters **`value`** – The value to set

Raises **ColumnStoreDataError** When an invalid value is supplied

ColumnStoreDecimal`::ColumnStoreDecimal` (**`const`** std::string &*value*)
Sets the decimal to the contents of a supplied `std::string` value (such as `"3.14159"`).

Parameters **`value`** – The value to set

Raises **ColumnStoreDataError** When an invalid value is supplied

ColumnStoreDecimal`::ColumnStoreDecimal` (double *value*)
Sets the decimal to the contents of a supplied `double` value.

Note: The internally this uses the `std::string` method so the performance may be lower than expected.

Parameters **`value`** – The value to set

Raises **ColumnStoreDataError** When an invalid value is supplied

ColumnStoreDecimal`::ColumnStoreDecimal` (int64_t *number*, uint8_t *scale*)
Sets the decimal to a given number and scale. For example for the value 3.14159 you would set the number to `314159` and the scale to `5`.

Parameters

- **`number`** – The number to set
- **`scale`** – The scale for the number

Raises **ColumnStoreDataError** When an invalid number/scale is supplied

5.8.2 set()

bool *ColumnStoreDecimal*`::set` (int64_t *value*)
Sets the decimal to an supplied integer value.

Parameters **`value`** – The value to set

Returns Always returns `true`

bool *ColumnStoreDecimal*`::set` (**`const`** std::string &*value*)
Sets the decimal to the contents of a supplied `std::string` value (such as `"3.14159"`).

Parameters **`value`** – The value to set

Returns `true` if the conversion was successful or `false` if it failed

bool *ColumnStoreDecimal*`::set` (double *value*)
Sets the decimal to the contents of a supplied `std::string` value (such as `"3.14159"`).

Note: The internally this uses the `std::string` method so the performance may be lower than expected.

Parameters **value** – The value to set

Returns `true` if the conversion was successful or `false` if it failed

bool *ColumnStoreDecimal*::**set**(int64_t *number*, uint8_t *scale*)

Sets the decimal to a given number and scale. For example for the value 3.14159 you would set the number to `314159` and the scale to `5`.

Parameters

- **number** – The number to set
- **scale** – The scale for the number

Returns `true` if the conversion was successful or `false` if it failed

5.9 ColumnStoreSystemCatalog Class

class ColumnStoreSystemCatalog

A class which contains the ColumnStore system catalog of tables and columns. It should be instantiated using *ColumnStoreDriver::getSystemCatalog()*.

Note: The system catalog stores schema, table and column names as lower case and therefore the functions only return lower case names. Since version 1.1.4 we make case insensitive matches.

5.9.1 getTable()

ColumnStoreSystemCatalogTable &*ColumnStoreSystemCatalog*::**getTable**(**const** std::string &*schemaName*, **const** std::string &*tableName*)

Gets the table information for a specific table.

Parameters

- **schemaName** – The schema the table is in
- **tableName** – The name of the table

Returns The table information

Raises ColumnStoreNotFound If the table is not found in the system catalog

5.10 ColumnStoreSystemCatalogTable Class

class ColumnStoreSystemCatalogTable

A class which contains the system catalog information for a specific table. It should be instantiated using *ColumnStoreSystemCatalog::getTable()*.

Note: The system catalog stores schema, table and column names as lower case and therefore the functions only return lower case names. Since version 1.1.4 we make case insensitive matches.

5.10.1 getSchemaName()

const std::string &*ColumnStoreSystemCatalogTable*::**getSchemaName**()
Retrieves the database schema name for the table

Returns The schema name

5.10.2 getTableName()

const std::string &*ColumnStoreSystemCatalogTable*::**getTableName**()
Retrieves the table name for the table

Returns The table name

5.10.3 getOID()

uint32_t *ColumnStoreSystemCatalogTable*::**getOID**()
Retrieves the ColumnStore object ID for the table.

Returns The object ID for the table

5.10.4 getColumnCount()

uint16_t *ColumnStoreSystemCatalogTable*::**getColumnCount**()
Retrieves the number of columns in the table

Returns The number of columns in the table

5.10.5 getColumn()

ColumnStoreSystemCatalogColumn &*ColumnStoreSystemCatalogTable*::**getColumn**(**const** std::string &*columnName*)
Retrieves the column information for a specified column by name

Parameters **columnName** – The name of the column to retrieve

Returns The column information

Raises **ColumnStoreNotFound** If the column is not found

ColumnStoreSystemCatalogColumn &*ColumnStoreSystemCatalogTable*::**getColumn**(uint16_t *columnNumber*)
Retrieves the column information for a specified column by number starting at zero

Parameters **columnNumber** – The number of the column to retrieve starting at 0

Returns The column information

Raises **ColumnStoreNotFound** If the column is not found

5.11 ColumnStoreSystemCatalogColumn Class

class ColumnStoreSystemCatalogColumn
A class containing information about a specific column in the system catalog. Should be instantiated using *ColumnStoreSystemCatalogTable::getColumn()*.

Note: The system catalog stores schema, table and column names as lower case and therefore the functions only return lower case names. Since version 1.1.4 we make case insensitive matches.

5.11.1 getOID()

uint32_t *ColumnStoreSystemCatalogColumn*::**getOID**()
Retrieves the ColumnStore object ID for the column

Returns The column object ID

5.11.2 getColumnName()

const std::string &*ColumnStoreSystemCatalogColumn*::**getColumnName**()
Retrieves the name of the column

Returns The column name

5.11.3 getDictionaryOID()

uint32_t *ColumnStoreSystemCatalogColumn*::**getDictionaryOID**()
Retrieves the dictionary object ID for the column (or `0` if there is no dictionary)

Returns The dictionary object ID or `0` for no dictionary

5.11.4 getType()

columnstore_data_types_t ColumnStoreSystemCatalogColumn::**getType**()
Retrieves the data type for the column

Returns The data type for the column

5.11.5 getWidth()

uint32_t *ColumnStoreSystemCatalogColumn*::**getWidth**()
Retrieves the width in bytes for the column

Returns The width in bytes

5.11.6 getPosition()

uint32_t *ColumnStoreSystemCatalogColumn*::**getPosition**()
Retrieves the column's position in the table. The sequence of columns in the table is sorted on object ID, columns may be out-of-order if an ALTER TABLE has inserted one in the middle of the table.

Returns The column's position in the table

5.11.7 getDefaultValue()

const std::string &*ColumnStoreSystemCatalogColumn*::**getDefaultValue**()
Retrieves the default value for the column in text. The value is empty for no default.

Returns The column's default value

5.11.8 isAutoincrement()

bool *ColumnStoreSystemCatalogColumn*::**isAutoincrement**()
Retrieves whether or not this column is an autoincrement column.

Returns `true` if this column is autoincrement, `false` if it isn't

5.11.9 getPrecision()

uint32_t *ColumnStoreSystemCatalogColumn*::**getPrecision**()
Retrieves the decimal precision for the column.

Returns The decimal precision

5.11.10 getScale()

uint32_t *ColumnStoreSystemCatalogColumn*::**getScale**()
Retrieves the decimal scale for the column.

Returns The decimal scale

5.11.11 isNullable()

bool *ColumnStoreSystemCatalogColumn*::**isNullable**()
Retrieves whether or not the column can be set to `NULL`

Returns `true` if the column can be `NULL` or `false` if it can not

5.11.12 compressionType()

uint8_t *ColumnStoreSystemCatalogColumn*::**compressionType**()
Retrieves the compression type for the column. `0` means no compression and `2` means Snappy compression

Returns The compression type for the column

INDEX

C

D

E