



# MariaDB

**MariaDB ColumnStore C++ API Usage  
Documentation**  
*Release 1.2.3-3d1ab30*

**MariaDB Corporation**

Mar 07, 2019

# CONTENTS

<b>1</b>	<b>Licensing</b>	<b>1</b>
1.1	Documentation Content . . . . .	1
1.2	MariaDB ColumnStore C++ API . . . . .	1
<b>2</b>	<b>Version History</b>	<b>2</b>
<b>3</b>	<b>Using mcsapi</b>	<b>3</b>
3.1	Usage Introduction . . . . .	3
3.2	Basic Bulk Insert . . . . .	3
3.3	Advanced Bulk Insert . . . . .	5
<b>4</b>	<b>Compiling with mcsapi</b>	<b>7</b>
4.1	Pre-requisites . . . . .	7
4.2	Compiling . . . . .	7
<b>5</b>	<b>mcsapi API Reference</b>	<b>9</b>
5.1	ColumnStoreDriver Class . . . . .	9
5.2	ColumnStoreBulkInsert Class . . . . .	16
5.3	ColumnStoreSummary Class . . . . .	23
5.4	ColumnStoreException Class . . . . .	27
5.5	columnstore_data_convert_status_t Type . . . . .	27
5.6	columnstore_data_types_t Type . . . . .	28
5.7	columnstore_lock_types_t Type . . . . .	29
5.8	TableLockInfo Struct . . . . .	29
5.9	ColumnStoreDateTime Class . . . . .	30
5.10	ColumnStoreDecimal Class . . . . .	31
5.11	ColumnStoreTime Class . . . . .	32
5.12	ColumnStoreSystemCatalog Class . . . . .	33
5.13	ColumnStoreSystemCatalogTable Class . . . . .	34
5.14	ColumnStoreSystemCatalogColumn Class . . . . .	35
	<b>Index</b>	<b>38</b>

## LICENSING

### 1.1 Documentation Content



The mcsapi documentation is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

### 1.2 MariaDB ColumnStore C++ API

The MariaDB ColumnStore C++ API (mcsapi) is licensed under the [GNU Lesser General Public License, version 2.1](https://www.gnu.org/licenses/lesser.html).

## VERSION HISTORY

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

Version	Changes
1.2.2	<ul style="list-style-type: none"><li>• Add <code>ColumnStoreDriver::listTableLocks()</code></li><li>• Add <code>ColumnStoreDriver::isTableLocked()</code></li><li>• Add <code>ColumnStoreDriver::clearTableLock()</code></li></ul>
1.2.1	<ul style="list-style-type: none"><li>• <code>ColumnStoreDriver::setDebug()</code> is now an integer setting for debug level instead of boolean</li><li>• <code>ColumnStoreBulkInsert::setColumn()</code> now has <code>bool</code> and <code>char*</code> methods</li></ul>
1.2.0	<ul style="list-style-type: none"><li>• Add support for <code>TIME</code> datatype with <code>ColumnStoreTime</code></li></ul>
1.1.6	<ul style="list-style-type: none"><li>• Windows support added (Alpha)</li></ul>
1.1.4	<ul style="list-style-type: none"><li>• Make <code>ColumnStoreSystemCatalog::getTable()</code> and <code>ColumnStoreSystemCatalogTable::getColumn()</code> case insensitive</li><li>• Add <code>ColumnStoreDriver::setDebug()</code> to enable debugging output to <code>stderr</code></li></ul>
1.1.1	<ul style="list-style-type: none"><li>• Add <code>ColumnStoreBulkInsert::isActive()</code></li><li>• Make <code>ColumnStoreBulkInsert::rollback()</code> fail without error</li><li>• Add <code>ColumnStoreBulkInsert::resetRow()</code></li><li>• <code>ColumnStoreDateTime::ColumnStoreDateTime()</code> now uses <code>uint32_t</code> for every parameter</li><li>• <code>ColumnStoreSystemCatalog</code> now uses <code>const</code> for the sub-class strings</li></ul>
1.1.0 $\beta$	<ul style="list-style-type: none"><li>• First beta release</li></ul>

## 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 be 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`

```
1 CREATE TABLE `t1` (  
2   `a` INT(11) DEFAULT NULL,  
3   `b` INT(11) DEFAULT NULL  
4 ) ENGINE=Columnstore;
```

Listing 2: `example/basic_bulk_insert.cpp`

```
25 #include <libmcsapi/mcsapi.h>  
26 #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, chained 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

```

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

```

Listing 8: example/advanced\_bulk\_insert.cpp

```

25 #include <libmcsapi/mcsapi.h>
26 #include <iostream>
27
28 int main(void)
29 {
30     mcsapi::ColumnStoreDriver* driver = nullptr;
31     mcsapi::ColumnStoreBulkInsert* bulk = nullptr;
32     try {
33         driver = new mcsapi::ColumnStoreDriver();
34         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

```

35     bulk->setColumn(0, 1);
36     bulk->setColumn(1, "Andrew");
37     bulk->setColumn(2, "1936-12-24");
38     bulk->setColumn(3, "2017-07-07 15:14:12");
39     bulk->setColumn(4, "15239.45");
40     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

```

41     mcsapi::ColumnStoreDateTime dob;
42     dob.set("1972-05-23", "%Y-%m-%d");
43     mcsapi::ColumnStoreDateTime added;
44     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 way as the basic bulk insert example.



## 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 libboost-dev
```

#### 4.1.2 CentOS 7

```
sudo yum install snappy libuv libxml2 pkgconfig boost-devel gcc gcc-c++
```

#### 4.1.3 Windows 10 (x64)

On Windows you need to install the [Visual C++ Redistributable for Visual Studio 2015](#) in the x64 version in order to use mcsapi. You have to install Visual Studio 2017 to compile .cpp files that use mcsapi.

### 4.2 Compiling

#### 4.2.1 Linux

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`
```

#### 4.2.2 Windows 10 (x64)

Here is an basic example using Visual Studio's x64 Native Tools Command Prompt:

```
cl example.cpp %mcsapiInstallDir%\lib\mcsapi.lib /I %mcsapiInstallDir%\include
```

The variable `%mcsapiInstallDir%` represents the base installation directory of the Bulk Write SDK. (e.g. `C:\Program Files\MariaDB\ColumnStore Bulk Write SDK`)

To run the built executable you have to copy `mcapi`'s DLLs `libiconv.dll`, `libuv.dll`, `libxml2.dll` and `mcsapi.dll` from `%mcsapiInstallDir%\lib` to the directory of the built executable.

## 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

```
1 #include <iostream>
2 #include <libmcsapi/mcsapi.h>
3
4 int main(void)
5 {
6     mcsapi::ColumnStoreDriver* driver = nullptr;
7     try {
8         driver = new mcsapi::ColumnStoreDriver();
9     } catch (mcsapi::ColumnStoreError &e) {
10         std::cout << "Error caught " << e.what() << std::endl;
11     }
12     delete driver;
13     return 0;
14 }
```

*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

```
1 #include <iostream>
2 #include <libmcsapi/mcsapi.h>
3
4 int main(void)
5 {
6     mcsapi::ColumnStoreDriver* driver = nullptr;
7     try {
8         driver = new mcsapi::ColumnStoreDriver("/usr/local/mariadb/columnstore/etc/
↪Columnstore.xml");
9     } catch (mcsapi::ColumnStoreError &e) {
10         std::cout << "Error caught " << e.what() << std::endl;
11     }
12     delete driver;
13     return 0;
14 }
```

### 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 table 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

```
1 #include <iostream>
2 #include <libmcsapi/mcsapi.h>
3
```

(continues on next page)

(continued from previous page)

```

4  int main(void)
5  {
6      std::string table("t1");
7      std::string db("test");
8      mcsapi::ColumnStoreDriver* driver = nullptr;
9      mcsapi::ColumnStoreBulkInsert* bulkInsert = nullptr;
10     try {
11         driver = new mcsapi::ColumnStoreDriver();
12         bulkInsert = driver->createBulkInsert(db, table, 0, 0);
13     } catch (mcsapi::ColumnStoreError &e) {
14         std::cout << "Error caught " << e.what() << std::endl;
15     }
16     delete bulkInsert;
17     delete driver;
18     return 0;
19 }

```

### 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

```

1  #include <iostream>
2  #include <libmcsapi/mcsapi.h>
3
4  int main(void)
5  {
6      try {
7          mcsapi::ColumnStoreDriver* driver = new mcsapi::ColumnStoreDriver();
8          const char* version = driver->getVersion();
9          std::cout << version << std::endl;
10     } catch (mcsapi::ColumnStoreError &e) {
11         std::cout << "Error caught: " << e.what() << std::endl;
12     }
13     return 0;
14 }

```

### 5.1.4 setDebug()

void *ColumnStoreDriver::setDebug* (uint8\_t level)

Enables/disables verbose debugging output which is sent to stderr upon execution. Levels are as follows:

- 0 - Off
- 1 - Show messages and binary packets truncated at 1000 bytes
- 2 - Show full messages, full length binary packets and ASCII translations

**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** `level` – Set to the log level required, 0 = off.

### Example

```
1 #include <iostream>
2 #include <libmcsapi/mcsapi.h>
3
4 int main(void)
5 {
6     try {
7         mcsapi::ColumnStoreDriver* driver = new mcsapi::ColumnStoreDriver();
8         driver->setDebug(true);
9         // Debugging output is now enabled
10    } catch (mcsapi::ColumnStoreError &e) {
11        std::cout << "Error caught: " << e.what() << std::endl;
12    }
13    return 0;
14 }
```

### 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

```
1 #include <iostream>
2 #include <libmcsapi/mcsapi.h>
3
4 int main(void)
5 {
6     try {
7         mcsapi::ColumnStoreDriver* driver = new mcsapi::ColumnStoreDriver();
8         mcsapi::ColumnStoreSystemCatalog sysCat = driver->getSystemCatalog();
9
10        mcsapi::ColumnStoreSystemCatalogTable tbl = sysCat.getTable("test", "t1");
11        std::cout << "t1 has " << tbl.getColumnCount() << " columns" << endl;
12
13        mcsapi::ColumnStoreSystemCatalogColumn col1 = tbl.getColumn(0);
14        std::cout << "The first column in t1 is " << col1.getColumnName() << endl;
15    } catch (mcsapi::ColumnStoreError &e) {
16        std::cout << "Error caught: " << e.what() << std::endl;
17    }
18    return 0;
19 }
```

### 5.1.6 listTableLocks()

`std::vector<mcsapi::TableLockInfo> ColumnStoreDriver::listTableLocks()`

Returns a vector of TableLockInfo objects that contains information about the current table locks in the ColumnStore system.

**Returns** A vector of mcsapi::TableLockInfo objects

#### Example

```

1 #include <iostream>
2 #include <libmcsapi/mcsapi.h>
3
4 int main(void)
5 {
6     try{
7         mcsapi::ColumnStoreDriver* driver = new mcsapi::ColumnStoreDriver();
8         std::vector<mcsapi::TableLockInfo> tliv = driver.listTableLocks();
9     } catch (mcsapi::ColumnStoreError &e) {
10        std::cout << "Error caught: " << e.what() << std::endl;
11    }
12    return 0;
13 }
```

### 5.1.7 isTableLocked()

`bool ColumnStoreDriver::isTableLocked(const std::string &db, const std::string &table)`

Returns true if the specified table is locked and false if it is not locked.

#### Parameters

- **db** – The database name for the table to check
- **table** – The table name to check

**Returns** true if the specified table is locked, otherwise false

**Raises ColumnStoreServerError** If the specified table is not existent

---

**Note:** Only locks of tables that have been existent when ColumnStoreDriver was created can be detected.

---

#### Example

```

1 #include <iostream>
2 #include <libmcsapi/mcsapi.h>
3
4 int main(void)
5 {
6     try{
7         mcsapi::ColumnStoreDriver* driver = new mcsapi::ColumnStoreDriver();
8         bool locked = driver.isTableLocked("test", "tmp1");
9     } catch (mcsapi::ColumnStoreError &e) {
10        std::cout << "Error caught: " << e.what() << std::endl;
11    }
12 }
```

(continues on next page)

(continued from previous page)

```

11     }
12     return 0;
13 }

```

`bool ColumnStoreDriver::isTableLocked(const std::string &db, const std::string &table, TableLockInfo &rtn)`

Returns `true` if the specified table is locked and `false` if it is not locked. Further information about the table lock can be accessed through the referenced `TableLockInfo`.

#### Parameters

- **db** – The database name for the table to check
- **table** – The table name to check
- **rtn** – The `TableLockInfo` object reference to store further information about the table lock into.

**Returns** `true` if the specified table is locked, otherwise `false`

**Raises ColumnStoreServerError** If the specified table is not existent

---

**Note:** Only locks of tables that have been existent when `ColumnStoreDriver` was created can be detected.

---

### Example

```

1  #include <iostream>
2  #include <libmcsapi/mcsapi.h>
3
4  int main(void)
5  {
6      try{
7          mcsapi::ColumnStoreDriver* driver = new mcsapi::ColumnStoreDriver();
8          mcsapi::TableLockInfo tli;
9          bool locked = driver.isTableLocked("test", "tmp1", tli);
10         } catch (mcsapi::ColumnStoreError &e) {
11             std::cout << "Error caught: " << e.what() << std::endl;
12         }
13         return 0;
14     }

```

### 5.1.8 clearTableLock()

`void ColumnStoreDriver::clearTableLock(uint64_t lockId)`

Clears a table lock with given id

**Parameters** **lockId** – The id of the table lock to clear

#### Example



```

1 #include <iostream>
2 int main(void)
3 {
4     try{
5         mcsapi::ColumnStoreDriver* driver = new mcsapi::ColumnStoreDriver();
6         std::vector<mcsapi::TableLockInfo> tliv = driver.listTableLocks();
7         for( auto& tli : tliv ){
8             driver.clearTableLock(tli.id);
9         }
10    } catch (mcsapi::ColumnStoreError &e) {
11        std::cout << "Error caught: " << e.what() << std::endl;
12    }
13    return 0;
14 }

```

void *ColumnStoreDriver::clearTableLock* (mcsapi::TableLockInfo *tli*)

Clears a table lock with given TableLockInfo element using its lock id

**Parameters** *tli* – The TableLockInfo object whose id will be used to clear the lock

### Example

```

1 #include <iostream>
2 #include <libmcsapi/mcsapi.h>
3
4 int main(void)
5 {
6     try{
7         mcsapi::ColumnStoreDriver* driver = new mcsapi::ColumnStoreDriver();
8         std::vector<mcsapi::TableLockInfo> tliv = driver.listTableLocks();
9         for( auto& tli : tliv ){
10            driver.clearTableLock(tli);
11        }
12    } catch (mcsapi::ColumnStoreError &e) {
13        std::cout << "Error caught: " << e.what() << std::endl;
14    }
15    return 0;
16 }

```

void *ColumnStoreDriver::clearTableLock* (const std::string &*db*, const std::string &*table*)

Clears a table lock of given database table combinationd

#### Parameters

- **db** – The database name for the table to clear
- **table** – The table name to clear

### Example

```

1 #include <iostream>
2 #include <libmcsapi/mcsapi.h>
3
4 int main(void)
5 {
6     try{

```

(continues on next page)

(continued from previous page)

```

7     mcsapi::ColumnStoreDriver* driver = new mcsapi::ColumnStoreDriver();
8     driver.clearTableLock("test", "tmp1");
9     } catch (mcsapi::ColumnStoreError &e) {
10        std::cout << "Error caught: " << e.what() << std::endl;
11    }
12    return 0;
13 }

```

## 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.

```

1 ...
2 driver = new mcsapi::ColumnStoreDriver();
3 bulkInsert = driver->createBulkInsert(db, table, 0, 0);
4 // columnCount will now contain the number of columns in the table
5 uint16_t columnCount = bulkInsert->getColumnCount();
6 ...

```

### 5.2.2 setColumn()

```
ColumnStoreBulkInsert *ColumnStoreBulkInsert::setColumn (uint16_t columnNumber, const
std::string &value, column-
store_data_convert_status_t *status
= nullptr)
```

```

ColumnStoreBulkInsert *ColumnStoreBulkInsert::setColumn (uint16_t      columnNumber,
                                                         const char *value, column-
                                                         store_data_convert_status_t *status
                                                         = nullptr)

ColumnStoreBulkInsert *ColumnStoreBulkInsert::setColumn (uint16_t columnNumber, const
                                                         char *value, size_t length, column-
                                                         store_data_convert_status_t *status
                                                         = nullptr)

ColumnStoreBulkInsert *ColumnStoreBulkInsert::setColumn (uint16_t      columnNumber,
                                                         uint64_t      value,      column-
                                                         store_data_convert_status_t *status
                                                         = nullptr)

ColumnStoreBulkInsert *ColumnStoreBulkInsert::setColumn (uint16_t      columnNumber,
                                                         int64_t      value,      column-
                                                         store_data_convert_status_t *status
                                                         = nullptr)

ColumnStoreBulkInsert *ColumnStoreBulkInsert::setColumn (uint16_t      columnNumber,
                                                         uint32_t      value,      column-
                                                         store_data_convert_status_t *status
                                                         = nullptr)

ColumnStoreBulkInsert *ColumnStoreBulkInsert::setColumn (uint16_t      columnNumber,
                                                         int32_t      value,      column-
                                                         store_data_convert_status_t *status
                                                         = nullptr)

ColumnStoreBulkInsert *ColumnStoreBulkInsert::setColumn (uint16_t      columnNumber,
                                                         uint16_t      value,      column-
                                                         store_data_convert_status_t *status
                                                         = nullptr)

ColumnStoreBulkInsert *ColumnStoreBulkInsert::setColumn (uint16_t      columnNumber,
                                                         int16_t      value,      column-
                                                         store_data_convert_status_t *status
                                                         = nullptr)

ColumnStoreBulkInsert *ColumnStoreBulkInsert::setColumn (uint16_t      columnNumber,
                                                         uint8_t      value,      column-
                                                         store_data_convert_status_t *status
                                                         = nullptr)

ColumnStoreBulkInsert *ColumnStoreBulkInsert::setColumn (uint16_t      columnNum-
                                                         ber,  int8_t  value,      column-
                                                         store_data_convert_status_t *status
                                                         = nullptr)

ColumnStoreBulkInsert *ColumnStoreBulkInsert::setColumn (uint16_t      columnNum-
                                                         ber,  bool  value,      column-
                                                         store_data_convert_status_t *status
                                                         = nullptr)

ColumnStoreBulkInsert *ColumnStoreBulkInsert::setColumn (uint16_t      columnNumber,
                                                         double      value,      column-
                                                         store_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.

```

1  ...
2  driver = new mcsapi::ColumnStoreDriver();
3  bulkInsert = driver->createBulkInsert(db, table, 0, 0);
4
5  // Create a decimal value
6  ColumnStoreDecimal decimalVal;
7  decimalVal.set("3.14159");
8
9  // And an int value
10 uint32_t intVal = 123456;
11
12 // And a string value
13 std::string strVal("Hello World");
14
15 // Finally a date/time values
16 ColumnStoreDateTime dateTime;
17 std::string newTime("1999-01-01 23:23:23");
18 std::string tFormat("%Y-%m-%d %H:%M:%S");
19 dateTime.set(newTime, tFormat);
20
21 // A status variable so we can check all is good
22 mcsapi::columnstore_data_convert_status_t status;
23
24 bulkInsert->setColumn(0, intVal, &status);
25 // Check conversion status
26 if (status != CONVERT_STATUS_NONE)

```

(continues on next page)

(continued from previous page)

```

27     return 1;
28 bulkInsert->setColumn(1, decimalVal, &status);
29 // Check conversion status
30 if (status != CONVERT_STATUS_NONE)
31     return 1;
32 bulkInsert->setColumn(2, strVal, &status);
33 // Check conversion status
34 if (status != CONVERT_STATUS_NONE)
35     return 1;
36 bulkInsert->setColumn(3, dateTime, &status);
37 // Check conversion status
38 if (status != CONVERT_STATUS_NONE)
39     return 1;
40
41 // Write this row ready to start another
42 bulkInsert->writeRow();
43
44 decimalVal.set("1.41421");
45 intVal = 654321;
46 strVal = "dlroW olleH";
47 newTime = "2017-07-05 22:00:43";
48 dateTime.set(newTime, tFormat);
49
50 // A chained example
51 bulkInsert->setColumn(0, intVal)->setColumn(1, decimalVal)->setColumn(2, strVal)->
52   ↳setColumn(3, dateTime)->writeRow();
53 ...

```

### 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.

```
1 ...
2 driver = new mcsapi::ColumnStoreDriver();
3 bulkInsert = driver->createBulkInsert(db, table, 0, 0);
4
5 // Set an whole row of NULLs
6 bulkInsert->setNull(0)->setNull(1)->setNull(2)->setNull(3)->writeRow();
7 ...
```

## 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.

```
1 ...
2 driver = new mcsapi::ColumnStoreDriver();
3 bulkInsert = driver->createBulkInsert(db, table, 0, 0);
4
5 // Set values for a 2 int column table
6 bulkInsert->setValue(0, (uint32_t) 123456);
7 bulkInsert->setValue(1, (uint32_t) 654321);
8
9 // Write the row
10 bulkInsert->writeRow();
11 ...
```

## 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.

```

1  ...
2  driver = new mcsapi::ColumnStoreDriver();
3  bulkInsert = driver->createBulkInsert(db, table, 0, 0);
4
5  // Set values for a 2 int column table
6  bulkInsert->setValue(0, (uint32_t) 123456);
7  bulkInsert->setValue(1, (uint32_t) 654321);
8
9  // Write the row
10 bulkInsert->writeRow();
11
12 // Commit the transaction
13 bulkInsert->commit();
14
15 // This WILL throw an exception if uncommented
16 // bulkInsert->setValue(0, (uint32_t) 99999);
17 ...

```

## 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.

```
1  ...
2  driver = new mcsapi::ColumnStoreDriver();
3  bulkInsert = driver->createBulkInsert(db, table, 0, 0);
4
5  // Set values for a 2 int column table
6  bulkInsert->setValue(0, (uint32_t) 123456);
7  bulkInsert->setValue(1, (uint32_t) 654321);
8
9  // Write the row
10 bulkInsert->writeRow();
11
12 // Rollback the transaction
13 bulkInsert->rollback();
14
15 // This WILL throw an exception if uncommented
16 // bulkInsert->setValue(0, (uint32_t) 99999);
17 ...
```

## 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.

```
1  ...
2  driver = new mcsapi::ColumnStoreDriver();
3  bulkInsert = driver->createBulkInsert(db, table, 0, 0);
4
5  // Set values for a 2 int column table
6  bulkInsert->setValue(0, (uint32_t) 123456);
7  bulkInsert->setValue(1, (uint32_t) 654321);
8
9  // Write the row
```

(continues on next page)



(continued from previous page)

```

10 bulkInsert->writeRow();
11
12 // Rollback the transaction
13 bulkInsert->rollback();
14
15 // Get the summary
16 ColumnStoreSummary summary = bulkInsert->getSummary();
17
18 // Get the number of inserted rows before they were rolled back
19 uint64_t rows = summary.getRowsInsertedCount();
20 ...

```

### 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.

```

1 ...
2 driver = new mcsapi::ColumnStoreDriver();
3 bulkInsert = driver->createBulkInsert(db, table, 0, 0);
4
5 bulkInsert->setTruncateIsError(true);
6 std::string strVal("Short string");
7
8 // A short string that will insert fine
9 bulkInsert->setValue(0, strVal);
10
11 // This long string will truncate on my VARCHAR(20) and throw an exception
12 strVal = "This is a long string test to demonstrate setTruncateIsError()";
13 bulkInsert->setValue(1, strVal);
14 ...

```

### 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.

```
1  ...
2  driver = new mcsapi::ColumnStoreDriver();
3  bulkInsert = driver->createBulkInsert(db, table, 0, 0);
4
5  // Set values for a 2 int column table
6  bulkInsert->setValue(0, (uint32_t) 123456);
7  bulkInsert->setValue(1, (uint32_t) 654321);
8
9  // Write the row
10 bulkInsert->writeRow();
11
12 // Rollback the transaction
13 bulkInsert->rollback();
14
15 // Get the summary
16 ColumnStoreSummary summary = bulkInsert->getSummary();
17
18 // Get the execution time for the transaction
19 double execTime = summary.getExecutionTime();
20 ...
```

### 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.

```
1  ...
2  driver = new mcsapi::ColumnStoreDriver();
3  bulkInsert = driver->createBulkInsert(db, table, 0, 0);
4
5  // Set values for a 2 int column table
6  bulkInsert->setValue(0, (uint32_t) 123456);
7  bulkInsert->setValue(1, (uint32_t) 654321);
8
9  // Write the row
10 bulkInsert->writeRow();
```

(continues on next page)

(continued from previous page)

```

11
12 // Rollback the transaction
13 bulkInsert->rollback();
14
15 // Get the summary
16 ColumnStoreSummary summary = bulkInsert->getSummary();
17
18 // Get the number of inserted rows before they were rolled back
19 uint64_t rows = summary.getRowsInsertedCount();
20 ...

```

### 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.

```

1 ...
2 driver = new mcsapi::ColumnStoreDriver();
3 bulkInsert = driver->createBulkInsert(db, table, 0, 0);
4
5 std::string strVal("Short string");
6
7 // A short string that will insert fine
8 bulkInsert->setValue(0, strVal);
9
10 // This long string will truncate on my VARCHAR(20)
11 strVal = "This is a long string test to demonstrate a truncation";
12 bulkInsert->setValue(1, strVal);
13
14 // Get the summary
15 ColumnStoreSummary summary = bulkInsert->getSummary();
16
17 // Get the number of truncated values before they were rolled back
18 uint64_t truncateCount = summary.getTruncationCount();
19 ...

```

### 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.

```
1 ...
2 driver = new mcsapi::ColumnStoreDriver();
3 bulkInsert = driver->createBulkInsert(db, table, 0, 0);
4
5 // Set values for a 2 int column table
6 bulkInsert->setValue(0, (uint32_t) 123456);
7 // Slightly higher than a signed int max, this will saturate
8 bulkInsert->setValue(1, (uint32_t) 2147483650);
9
10 // Write the row
11 bulkInsert->writeRow();
12
13 // Rollback the transaction
14 bulkInsert->rollback();
15
16 // Get the summary
17 ColumnStoreSummary summary = bulkInsert->getSummary();
18
19 // Get the number of saturated values before they were rolled back
20 uint64_t saturatedCount = summary.getSaturatedCount();
21 ...
```

### 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.

```
1 ...
2 driver = new mcsapi::ColumnStoreDriver();
3 bulkInsert = driver->createBulkInsert(db, table, 0, 0);
4
5 // Set values for a 2 int column table
6 bulkInsert->setValue(0, (uint32_t) 123456);
7 // This is a DATE column, which is invalid to set as a date.
8 // The result will be the date set to '0000-00-00'
9 // and a invalid counter increment
10 bulkInsert->setValue(1, (uint32_t) 123456);
11
12 // Write the row
13 bulkInsert->writeRow();
14
15 // Rollback the transaction
16 bulkInsert->rollback();
```

(continues on next page)

(continued from previous page)

```

17
18 // Get the summary
19 ColumnStoreSummary summary = bulkInsert->getSummary();
20
21 // Get the number of invalid values before they were rolled back
22 uint64_t invalidCount = summary.getInvalidCount();
23 ...

```

## 5.4 ColumnStoreException Class

**class ColumnStoreError** : public `std::runtime_error`

A general exception 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 exception class covering problems with the driver’s internals

**class ColumnStoreBufferError** : public *ColumnStoreError*

An exception class covering problems with the driver’s buffers

**class ColumnStoreServerError** : public *ColumnStoreError*

An exception class that contains errors received from the server

**class ColumnStoreNetworkError** : public *ColumnStoreError*

An exception class covering problems with network connection and communication

**class ColumnStoreDataError** : public *ColumnStoreError*

An exception class covering problems with setting column data and data conversion

**class ColumnStoreUsageError** : public *ColumnStoreError*

An exception class covering problems with usage of the driver

**class ColumnStoreConfigError** : public *ColumnStoreError*

An exception class covering problems with reading the XML configuration

**class ColumnStoreVersionError** : public *ColumnStoreError*

An exception class for an incompatible ColumnStore version number

**class ColumnStoreNotFound** : public *ColumnStoreError*

An exception 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 columnstore\_lock\_types\_t Type

These are constants for the table locks as returned by *TableLockInfo::state()*

**type LOCK\_TYPE\_CLEANUP**

**type LOCK\_TYPE\_LOADING**

## 5.8 TableLockInfo Struct

**type TableLockInfo**  
A struct containing table lock information

**type TableLockInfo::id**  
UNSIGNED int64 lock id

**type TableLockInfo::ownerName**  
std::string name of the owner of the table lock

**type TableLockInfo::ownerPID**  
UNSIGNED int32 process id of the lock owner

**type TableLockInfo::ownerSessionID**  
UNSIGNED int32 session id of the lock owner

**type TableLockInfo::ownerTxnID**  
UNSIGNED int32 transaction id of the lock owner

**type TableLockInfo::state**  
columnstore\_lock\_types\_t state of the lock

**type TableLockInfo::creationTime**  
time\_t creation time of the lock

**type TableLockInfo::dbrootList**  
std::vector<uint32\_t> list of PMs involved in this lock

## 5.9 ColumnStoreDateTime Class

### class ColumnStoreDateTime

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

### 5.9.1 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 integers

---

**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

### 5.9.2 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

## 5.10 ColumnStoreDecimal Class

**class ColumnStoreDecimal**

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

### 5.10.1 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

## 5.10.2 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

## 5.11 ColumnStoreTime Class

**class ColumnStoreTime**

A class which is used to contain a date/time used to set TIME columns using *ColumnStoreBulkInsert::setColumn()*

### 5.11.1 ColumnStoreTime()

*ColumnStoreTime*::ColumnStoreTime ()

Sets the date/time to 00:00:00.

*ColumnStoreTime*::ColumnStoreTime (tm &time)

Sets the time value of the tm struct.

**Parameters** *time* – The time to set

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

`ColumnStoreTime::ColumnStoreTime (const std::string &time, const std::string &format)`

Sets the time based on a given string and format.

**Parameters**

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

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

`ColumnStoreTime::ColumnStoreTime (int32_t hour, uint32_t minute, uint32_t second, uint32_t microsecond = 0, bool is_negative = false)`

Sets the time based on a given set of integers

---

**Note:** If the the time is a negative and the hours are zero then `is_negative` should be set. Otherwise the driver will automatically set this for you.

---

**Parameters**

- **hour** – The hour
- **minute** – The minute
- **second** – The second
- **microsecond** – The microseconds
- **is\_negative** – A zero hour time that is negative

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

## 5.11.2 set()

`bool ColumnStoreTime::set (tm &time)`

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

**Parameters** **time** – The time to set

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

`bool ColumnStoreTime::set (const std::string &time, const std::string &format)`

Sets the time based on a given string and format.

**Parameters**

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

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

## 5.12 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.

---

### 5.12.1 `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

## 5.13 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.

---

### 5.13.1 `getSchemaName()`

```
const std::string &ColumnStoreSystemCatalogTable::getSchemaName ()
```

Retrieves the database schema name for the table

**Returns** The schema name

### 5.13.2 `getTableName()`

```
const std::string &ColumnStoreSystemCatalogTable::getTableName ()
```

Retrieves the table name for the table

**Returns** The table name

### 5.13.3 `getOID()`

```
uint32_t ColumnStoreSystemCatalogTable::getOID ()
```

Retrieves the ColumnStore object ID for the table.

**Returns** The object ID for the table

### 5.13.4 getColumnCount()

`uint16_t ColumnStoreSystemCatalogTable::getColumnCount ()`

Retrieves the number of columns in the table

**Returns** The number of columns in the table

### 5.13.5 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

## 5.14 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.

---

### 5.14.1 getOID()

`uint32_t ColumnStoreSystemCatalogColumn::getOID ()`

Retrieves the ColumnStore object ID for the column

**Returns** The column object ID

### 5.14.2 getColumnName()

`const std::string &ColumnStoreSystemCatalogColumn::getColumnName ()`

Retrieves the name of the column

**Returns** The column name

### 5.14.3 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

### 5.14.4 getType()

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

**Returns** The data type for the column

### 5.14.5 getWidth()

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

**Returns** The width in bytes

### 5.14.6 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

### 5.14.7 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

### 5.14.8 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

### 5.14.9 getPrecision()

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

**Returns** The decimal precision

### 5.14.10 `getScale()`

`uint32_t ColumnStoreSystemCatalogColumn::getScale()`

Retrieves the decimal scale for the column.

**Returns** The decimal scale

### 5.14.11 `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

### 5.14.12 `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

## C

- columnstore\_data\_convert\_status\_t (C++ type), 27
- columnstore\_data\_types\_t (C++ type), 28
- COLUMNSTORE\_INSTALL\_DIR, 4, 9
- ColumnStoreBufferError (C++ class), 27
- ColumnStoreBulkInsert (C++ class), 16
- ColumnStoreBulkInsert::commit (C++ function), 21
- ColumnStoreBulkInsert::getColumnCount (C++ function), 16
- ColumnStoreBulkInsert::getSummary (C++ function), 22
- ColumnStoreBulkInsert::isActive (C++ function), 22
- ColumnStoreBulkInsert::resetRow (C++ function), 20
- ColumnStoreBulkInsert::rollback (C++ function), 21
- ColumnStoreBulkInsert::setBatchSize (C++ function), 23
- ColumnStoreBulkInsert::setColumn (C++ function), 16–18
- ColumnStoreBulkInsert::setNull (C++ function), 19
- ColumnStoreBulkInsert::setTruncateIsError (C++ function), 23
- ColumnStoreBulkInsert::writeRow (C++ function), 20
- ColumnStoreConfigError (C++ class), 27
- ColumnStoreDataError (C++ class), 27
- ColumnStoreDateTime (C++ class), 30
- ColumnStoreDateTime::ColumnStoreDateTime (C++ function), 30
- ColumnStoreDateTime::set (C++ function), 30, 31
- ColumnStoreDecimal (C++ class), 31
- ColumnStoreDecimal::ColumnStoreDecimal (C++ function), 31
- ColumnStoreDecimal::set (C++ function), 32
- ColumnStoreDriver (C++ class), 9
- ColumnStoreDriver::clearTableLock (C++ function), 14, 15
- ColumnStoreDriver::ColumnStoreDriver (C++ function), 9
- ColumnStoreDriver::createBulkInsert (C++ function), 10
- ColumnStoreDriver::getSystemCatalog (C++ function), 12
- ColumnStoreDriver::getVersion (C++ function), 11
- ColumnStoreDriver::isTableLocked (C++ function), 13, 14
- ColumnStoreDriver::listTableLocks (C++ function), 13
- ColumnStoreDriver::setDebug (C++ function), 11
- ColumnStoreError (C++ class), 27
- ColumnStoreInternalError (C++ class), 27
- ColumnStoreNetworkError (C++ class), 27
- ColumnStoreNotFound (C++ class), 27
- ColumnStoreServerError (C++ class), 27
- ColumnStoreSummary (C++ class), 23
- ColumnStoreSummary::getExecutionTime (C++ function), 24
- ColumnStoreSummary::getInvalidCount (C++ function), 26
- ColumnStoreSummary::getRowsInsertedCount (C++ function), 24
- ColumnStoreSummary::getSaturatedCount (C++ function), 25
- ColumnStoreSummary::getTruncationCount (C++ function), 25
- ColumnStoreSystemCatalog (C++ class), 33
- ColumnStoreSystemCatalog::getTable (C++ function), 34
- ColumnStoreSystemCatalogColumn (C++ class), 35
- ColumnStoreSystemCatalogColumn::compressionType (C++ function), 37
- ColumnStoreSystemCatalogColumn::getColumnName (C++ function), 35
- ColumnStoreSystemCatalogColumn::getDefaultValue (C++ function), 36



ColumnStoreSystemCatalogColumn::getDictionaryType (C++ function), 36  
 DATA\_TYPE\_SMALLINT (C++ type), 28  
 DATA\_TYPE\_TEXT (C++ type), 29

ColumnStoreSystemCatalogColumn::getOID (C++ function), 35  
 DATA\_TYPE\_TINYINT (C++ type), 28  
 DATA\_TYPE\_UBIGINT (C++ type), 29

ColumnStoreSystemCatalogColumn::getPosition (C++ function), 36  
 DATA\_TYPE\_UDECIMAL (C++ type), 28  
 DATA\_TYPE\_UDOUBLE (C++ type), 29

ColumnStoreSystemCatalogColumn::getPrecision (C++ function), 36  
 DATA\_TYPE\_UFLOAT (C++ type), 29  
 DATA\_TYPE\_UINT (C++ type), 29

ColumnStoreSystemCatalogColumn::getScale (C++ function), 37  
 DATA\_TYPE\_UMEDINT (C++ type), 29  
 DATA\_TYPE\_USMALLINT (C++ type), 28

ColumnStoreSystemCatalogColumn::getType (C++ function), 36  
 DATA\_TYPE\_UTINYINT (C++ type), 28  
 DATA\_TYPE\_VARBINARY (C++ type), 28

ColumnStoreSystemCatalogColumn::getWidth (C++ function), 36  
 DATA\_TYPE\_VARCHAR (C++ type), 28

ColumnStoreSystemCatalogColumn::isAutoincrement (C++ function), 36  
 E  
 environment variable

ColumnStoreSystemCatalogColumn::isNullable (C++ function), 37  
 COLUMNSTORE\_INSTALL\_DIR, 4, 9

ColumnStoreSystemCatalogTable (C++ class), 34  
 L  
 LOCK\_TYPE\_CLEANUP (C++ type), 29

ColumnStoreSystemCatalogTable::getColumnLock (C++ function), 35  
 LOCK\_TYPE\_LOADING (C++ type), 29

ColumnStoreSystemCatalogTable::getColumnRowCount (C++ function), 35  
 T  
 TableLockInfo::creationTime (C++ type), 29

ColumnStoreSystemCatalogTable::getOID (C++ function), 34  
 TableLockInfo::dbrootList (C++ type), 29

ColumnStoreSystemCatalogTable::getSchemaName (C++ function), 34  
 TableLockInfo::id (C++ type), 29

ColumnStoreSystemCatalogTable::getTableName (C++ function), 34  
 TableLockInfo::ownerName (C++ type), 29  
 TableLockInfo::ownerPID (C++ type), 29  
 TableLockInfo::ownerSessionID (C++ type), 29

ColumnStoreTime (C++ class), 32  
 TableLockInfo::ownerTxnID (C++ type), 29

ColumnStoreTime::ColumnStoreTime (C++ function), 32, 33  
 TableLockInfo::state (C++ type), 29  
 TableLockInfo (C++ type), 29

ColumnStoreTime::set (C++ function), 33

ColumnStoreUsageError (C++ class), 27

ColumnStoreVersionError (C++ class), 27

CONVERT\_STATUS\_INVALID (C++ type), 27

CONVERT\_STATUS\_NONE (C++ type), 27

CONVERT\_STATUS\_SATURATED (C++ type), 27

CONVERT\_STATUS\_TRUNCATED (C++ type), 28

## D

DATA\_TYPE\_BIGINT (C++ type), 28

DATA\_TYPE\_BIT (C++ type), 28

DATA\_TYPE\_BLOB (C++ type), 28

DATA\_TYPE\_CHAR (C++ type), 28

DATA\_TYPE\_CLOB (C++ type), 28

DATA\_TYPE\_DATE (C++ type), 28

DATA\_TYPE\_DATETIME (C++ type), 28

DATA\_TYPE\_DECIMAL (C++ type), 28

DATA\_TYPE\_DOUBLE (C++ type), 28

DATA\_TYPE\_FLOAT (C++ type), 28

DATA\_TYPE\_INT (C++ type), 28

DATA\_TYPE\_MEDINT (C++ type), 28