MariaDB ColumnStore Python API Usage Documentation

Release 1.2.1-2d5cd1e

MariaDB Corporation

Nov 08, 2018

CONTENTS

CHAPTER ONE

LICENSING

1.1 Documentation Content

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

1.2 MariaDB ColumnStore Python API

The MariaDB ColumnStore Python API (pymcsapi) is licensed under the GNU Lesser General Public License, version 2.1.

CHAPTER

TWO

VERSION HISTORY

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

Version	Changes
1.1.6	• Python documentation added • Windows support added (Alpha)
1.1.5	• Changed the return type of `ColumnStoreBulkInsert.setColumn()` and `ColumnStoreBulkInsert.setNull()` to return a List of ColumnStoreBulkInsert object and return status. Before only the ColumnStoreBulkInsert object was returned.
1.1.4	• Make `ColumnStoreSystemCatalog.getTable()` and `ColumnStoreSystemCatalogTable.getColumn()` case insensitive • Add `ColumnStoreDriver.setDebug()` to enable debugging output to stderr
1.1.1	• Add `ColumnStoreBulkInsert.isActive()` • Make `ColumnStoreBulkInsert.rollback()` fail without error • Add `ColumnStoreBulkInsert.resetRow()` • `pymcsapi.ColumnStoreDateTime()` now uses uint32_t for every parameter • `ColumnStoreSystemCatalog` now uses const for the sub-class strings
1.1.0β	• First beta release

CHAPTER

THREE

USING PYMCSAPI

3.1 Usage Introduction

The Python bulk insert API (pymcsapi) is a wrapper around the C++ bulk insert API (mcsapi) generated by SWIG. As a result Python programmers can utilize the same functions to insert data into ColumnStore tables as C++ developers can do. pymcsapi is available for Python 2 and Python 3.

3.2 Windows 10 (x64) installation

If Python 2.7 or Python 3.7 is detected during pymcsapi's installation, it can be installed automatically.

3.2.1 Manual Installation

If Python 2.7 or Python 3.7 wasn't detected during pymcsapi's installation you can install pymcsapi manually afterwards.

Following variables will be used to represent the program installation paths needed:

Variable	Description	Example
`%PythonInstallDir%`	The base installation directory Python was installed to	C:\Python27
`%Python3InstallDir%`	The base installation directory Python 3 was installed to	C:\Python37
`%mcsapiInstallDir%`	The base installation directory of the Bulk Write SDK	C:\Program Files\MariaDB\ColumnStore Bulk Write SDK

Python 2.7

- copy `libiconv.dll`, `libuv.dll`, `libxml2.dll` and `mcsapi.dll` from `%mcsapiInstallDir%\lib` into `%PythonInstallDir%\DLLs`
- copy `_pymcsapi.pyd` from `%mcsapiInstallDir%\lib\python` into `%PythonInstallDir%\DLLs`
- copy `columnStoreExporter.py`, `mcsapi_reserved_words.txt` and `pymcsapi.py` from `%mcsapiInstallDir%\lib\python2.7` into `%PythonInstallDir%\Lib`

Python 3.7

- copy `libiconv.dll`, `libuv.dll`, `libxml2.dll` and `mcsapi.dll` from `%mcsapiInstallDir%\lib` into `%Python3InstallDir%\DLLs`
- copy `_pymcsapi.pyd` from `%mcsapiInstallDir%\lib\python3` into `%Python3InstallDir%\DLLs`
- copy `columnStoreExporter.py`, `mcsapi_reserved_words.txt` and `pymcsapi.py` from `%mcsapiInstallDir%\lib\python3.7` into `%Python3InstallDir%\Lib`

Note: Unlike the Linux version of pymcsapi3 the Windows version of pymcsapi3 can only be used with the Python 3 release it was compiled with. Swig for Windows compiles against the Python3x.lib and not the main Python3.lib. Therefore, you might have to recompile pymcsapi3 from scratch if you need to use a specific Python 3 release.

3.3 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.py` 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

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

Listing 2: example/basic_bulk_insert.py

```python
import pymcsapi
```

We need to import `pymcsapi` which is the main module to use mcsapi.

Listing 3: example/basic_bulk_insert.py

```python
try:
    driver = pymcsapi.ColumnStoreDriver()
```

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`*.

Listing 4: example/basic_bulk_insert.py

```python
    bulk = driver.createBulkInsert("test", "t1", 0, 0)
```

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.py

```
    for i in range(0,1000):
        bulk.setColumn(0, i)
        bulk.setColumn(1, 1000-i)
        bulk.writeRow()
```

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.

Listing 6: example/basic_bulk_insert.py

```
    bulk.commit()
```

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 7: example/basic_bulk_insert.py

```
except RuntimeError as err:
    print("Error caught: %s" % (err,))
```

If anything fails then we should catch `RuntimeError` to handle it.

3.4 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 8: 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 9: example/advanced_bulk_insert.py

```
import pymcsapi

try:
    driver = pymcsapi.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 10: example/advanced_bulk_insert.py

```
    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 11: example/advanced_bulk_insert.py

```
    bulk.setColumn(0, 2)[0].setColumn(1, "David")[0].setColumn(2, pymcsapi.
↪ColumnStoreDateTime(1972, 5, 23))[0].setColumn(3, pymcsapi.ColumnStoreDateTime(2017,
↪ 7, 7, 15, 20, 18))[0].setColumn(4, pymcsapi.ColumnStoreDecimal(2347623, 2))[0].
↪writeRow()
```

Many of the *`ColumnStoreBulkInsert`* methods return a pointer to the class and a return status which means multiple calls can be chained together in a similar way to ORM APIs. Here we use additional datatypes `ColumnStoreDateTime` and `ColumnStoreDecimal`.

`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.

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.py

```
    bulk.commit()
    summary = bulk.getSummary()
    print("Execution time: %s" % (summary.getExecutionTime(),))
    print("Rows inserted: %s" % (summary.getRowsInsertedCount(),))
    print("Truncation count: %s" %(summary.getTruncationCount(),))
    print("Saturated count: %s" %(summary.getSaturatedCount(),))
    print("Invalid count: %s" %(summary.getInvalidCount(),))
```

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 13: example/advanced_bulk_insert.py

```
except RuntimeError as err:
    print("Error caught: %s" % (err,))
```

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

CHAPTER

FOUR

PYMCSAPI API REFERENCE

4.1 ColumnStoreDriver Class

class ColumnStoreDriver

This is the parent class for pymcsapi. 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.

4.1.1 ColumnStoreDriver()

`pymcsapi.`**`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 `RuntimeError` – When the Columnstore.xml file cannot be found or cannot be parsed

Example

```python
import pymcsapi

try:
    driver = pymcsapi.ColumnStoreDriver()
except RuntimeError as err:
    print("Error caught: %s" % (err,))
```

`pymcsapi.`**`ColumnStoreDriver`**`(`*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 `RuntimeError` – When the Columnstore.xml file cannot be found or cannot be parsed

Example

```python
import pymcsapi

try:
    driver = pymcsapi.ColumnStoreDriver('/usr/local/mariadb/columnstore/etc/
↪Columnstore.xml')
```

(continues on next page)

(continued from previous page)

```
except RuntimeError as err:
    print("Error caught: %s" % (err,))
```

4.1.2 createBulkInsert()

`ColumnStoreDriver.`**`createBulkInsert`**(*db*, *table*, *mode*, *pm*)

Allocates and configures an instance of `ColumnStoreBulkInsert` to be used for bulk inserts with the ColumnStore installation reference by the driver.

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 **`RuntimeError`** – If a table lock cannot be acquired for the desired table

Example

```
import pymcsapi

try:
    driver = pymcsapi.ColumnStoreDriver()
    bulkInsert = driver.createBulkInsert("test", "t1", 0, 0);
except RuntimeError as err:
    print("Error caught: %s" % (err,))
```

4.1.3 getVersion()

`ColumnStoreDriver.`**`getVersion`**()

Returns the version of the mcsapi 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 mcsapi version string

Example

```
import pymcsapi

try:
    driver = pymcsapi.ColumnStoreDriver()
    print("mcsapi version: %s" % (driver.getVersion(),))
except RuntimeError as err:
    print("Error caught: %s" % (err,))
```

4.1.4 setDebug()

`ColumnStoreDriver.`**`setDebug`**`(`*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

```python
import pymcsapi

try:
    driver = pymcsapi.ColumnStoreDriver()
    driver.setDebug(True)
    # Debugging output is now enabled
except RuntimeError as err:
    print("Error caught: %s" % (err,))
```

4.1.5 getSystemCatalog()

`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

```python
import pymcsapi

try:
    driver = pymcsapi.ColumnStoreDriver()
    sysCat = driver.getSystemCatalog()
    table = sysCat.getTable("test", "t1")
    print("t1 has %d columns" % (table.getColumnCount(),))
except RuntimeError as err:
    print("Error caught: %s" % (err,))
```

4.2 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.

4.2.1 getColumnCount()

`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…except blocks in the *`ColumnStoreDriver`* examples.

```
...
driver = pymcsapi.ColumnStoreDriver()
bulkInsert = driver.createBulkInsert(db, table, 0, 0)
# columnCount will now contain the number of columns in the table
columnCount = bulkInsert.getColumnCount()
...
```

4.2.2 setColumn()

`ColumnStoreBulkInsert.`**`setColumn`**`(`*columnNumber*, *value*`)`

Sets a value for a given column.

Parameters

- **`columnNumber`** – The column number to set (starting from `0`)
- **`value`** – The value to set this column

Returns A List of a pointer to the *`ColumnStoreBulkInsert`* class so that calls can be chained, and the return status.

Raises

- **`RuntimeError`** – If there is an error setting the column, such as truncation error when `ColumnStoreBulkInsert.setTruncateIsError()` is used or an invalid column number is supplied
- **`RuntimeError`** – If the transaction has already been closed

Example

This example can be used inside the try…except blocks in the *`ColumnStoreDriver`* examples.

```python
...
driver = pymcsapi.ColumnStoreDriver()
bulkInsert = driver.createBulkInsert(db, table, 0, 0)

# Create a decimal value
decimalVal = pymcsapi.ColumnStoreDecimal("3.14159")

# And an int value
intVal = 123456

# And a string value
strVal = "Hello World"

# Finally a date/time values
dateTime = pymcsapi.ColumnStoreDateTime("1999-01-01 23:23:23", "%Y-%m-%d %H:%M:%S")

nxt, status = bulkInsert.setColumn(0, intVal)
# Check conversion status
if status != pymcsapi.CONVERT_STATUS_NONE:
    exit(1)
nxt, status = bulkInsert.setColumn(1, decimalVal)
# Check conversion status
if status != pymcsapi.CONVERT_STATUS_NONE:
    exit(1)
nxt, status = bulkInsert.setColumn(2, strVal)
# Check conversion status
if status != pymcsapi.CONVERT_STATUS_NONE:
    exit(1)
nxt, status = bulkInsert.setColumn(3, dateTime)
# Check conversion status
if status != pymcsapi.CONVERT_STATUS_NONE:
    exit(1)

# Write this row ready to start another
bulkInsert.writeRow()

decimalVal.set("1.41421")
intVal = 654321
strVal = "dlroW olleH"
dateTime.set("2017-07-05 22:00:43", "%Y-%m-%d %H:%M:%S")

# A chained example
bulkInsert.setColumn(0, intVal)[0].setColumn(1, decimalVal)[0].setColumn(2,␣
↪strVal)[0].setColumn(3, dateTime)[0].writeRow()
...
```

4.2.3 setNull()

`ColumnStoreBulkInsert.`**`setNull`**(*columnNumber*)

Sets a `NULL` for a given column.

Parameters **`columnNumber`** – The column number to set (starting from `0`)

Returns A List of a pointer to the *`ColumnStoreBulkInsert`* class so that calls can be chained, and the return status

Raises

- **RuntimeError** – If there is an error setting the column, such as an invalid column number is supplied
- **RuntimeError** – If the transaction has already been closed

Example

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

```
...
driver = pymcsapi.ColumnStoreDriver()
bulkInsert = driver.createBulkInsert(db, table, 0, 0)

# Set an whole row of NULLs
bulkInsert.setNull(0)[0].setNull(1)[0].setNull(2)[0].setNull(3)[0].writeRow()
...
```

4.2.4 resetRow()

`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()*.

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

Raises **RuntimeError** – If the transaction has already been closed

4.2.5 writeRow()

`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.

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

Raises

- **RuntimeError** – If there has been an error during the write at the network level
- **RuntimeError** – If there has been an error during the write at the remote server level
- **RuntimeError** – If the transaction has already been closed

Example

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

```python
...
driver = pymcsapi.ColumnStoreDriver()
bulkInsert = driver.createBulkInsert(db, table, 0, 0)

# Set values for a 2 int column table
bulkInsert.setColumn(0, 123456)
bulkInsert.setColumn(1, 654321)

# Write the row
bulkInsert.writeRow()
...
```

4.2.6 commit()

`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

- **`RuntimeError`** – If there has been an error during the write at the network level
- **`RuntimeError`** – If there has been an error during the write at the remote server level
- **`RuntimeError`** – If the transaction has already been closed

Example

This example can be used inside the try…except blocks in the *`ColumnStoreDriver`* examples.

```python
...
driver = pymcsapi.ColumnStoreDriver()
bulkInsert = driver.createBulkInsert(db, table, 0, 0)

# Set values for a 2 int column table
bulkInsert.setColumn(0, 123456)
bulkInsert.setColumn(1, 654321)

# Write the row
bulkInsert.writeRow()

# Commit the transaction
bulkInsert.commit()

# This WILL throw an exception if uncommented
```

(continues on next page)

(continued from previous page)

```
# bulkInsert.setColumn(0, 99999)
...
```

4.2.7 rollback()

`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

- **`RuntimeError`** – If there has been an error during the write at the network level
- **`RuntimeError`** – If there has been an error during the write at the remote server level

Example

This example can be used inside the try…except blocks in the *`ColumnStoreDriver`* examples.

```
...
driver = pymcsapi.ColumnStoreDriver()
bulkInsert = driver.createBulkInsert(db, table, 0, 0)

# Set values for a 2 int column table
bulkInsert.setColumn(0, 123456)
bulkInsert.setColumn(1, 654321)

# Write the row
bulkInsert.writeRow()

# Rollback the transaction
bulkInsert.rollback()

# This WILL throw an exception if uncommented
# bulkInsert.setColumn(0, 99999)
...
```

4.2.8 isActive()

`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

4.2.9 getSummary()

`ColumnStoreBulkInsert.getSummary()`

Gets the summary information for this bulk write transaction.

Returns The summary object

Example

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

```
...
driver = pymcsapi.ColumnStoreDriver()
bulkInsert = driver.createBulkInsert(db, table, 0, 0)

# Set values for a 2 int column table
bulkInsert.setColumn(0, 123456)
bulkInsert.setColumn(1, 654321)

# Write the row
bulkInsert.writeRow()

# Rollback the transaction
bulkInsert.rollback()

# Get the summary
summary = bulkInsert.getSummary()

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

4.2.10 setTruncateIsError()

`ColumnStoreBulkInsert::setTruncateIsError(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. . . except blocks in the *ColumnStoreDriver* examples.

```
...
driver = pymcsapi.ColumnStoreDriver()
bulkInsert = driver.createBulkInsert(db, table, 0, 0)

bulkInsert.setTruncateIsError(True)

# A short string that will insert fine
bulkInsert.setColumn(0, "Short string")

# This long string will truncate on my VARCHAR(20) and throw an exception
```

(continues on next page)

(continued from previous page)

```
bulkInsert.setColumn(1, "This is a long string test to demonstrate␣
↪setTruncateIsError()")
...
```

4.2.11 setBatchSize()

`ColumnStoreBulkInsert.`**`setBatchSize`**`(`*batchSize*`)`

Future use, this has not been implemented yet

4.3 ColumnStoreSummary Class

`class ColumnStoreSummary`

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

4.3.1 getExecutionTime()

`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…except blocks in the *`ColumnStoreDriver`* examples.

```
...
driver = pymcsapi.ColumnStoreDriver()
bulkInsert = driver.createBulkInsert(db, table, 0, 0)

# Set values for a 2 int column table
bulkInsert.setColumn(0, 123456)
bulkInsert.setColumn(1, 654321)

# Write the row
bulkInsert.writeRow()

# Rollback the transaction
bulkInsert.rollback()

# Get the summary
summary = bulkInsert.getSummary()

# Get the execution time for the transaction
execTime = summary.getExecutionTime()
...
```

4.3.2 getRowsInsertedCount()

`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. . . except blocks in the *ColumnStoreDriver* examples.

```python
...
driver = pymcsapi.ColumnStoreDriver()
bulkInsert = driver.createBulkInsert(db, table, 0, 0)

# Set values for a 2 int column table
bulkInsert.setColumn(0, 123456)
bulkInsert.setColumn(1, 654321)

# Write the row
bulkInsert.writeRow()

# Rollback the transaction
bulkInsert.rollback()

# Get the summary
summary = bulkInsert.getSummary()

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

4.3.3 getTruncationCount()

`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. . . except blocks in the *ColumnStoreDriver* examples.

```python
...
driver = pymcsapi.ColumnStoreDriver()
bulkInsert = driver.createBulkInsert(db, table, 0, 0)

# Set values for a 2 int column table
bulkInsert.setColumn(0, 123456)
bulkInsert.setColumn(1, 654321)

# Write the row
bulkInsert.writeRow()

```

(continues on next page)

(continued from previous page)

```
# Rollback the transaction
bulkInsert.rollback()

# Get the summary
summary = bulkInsert.getSummary()

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

4.3.4 getSaturatedCount()

`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. . . except blocks in the *`ColumnStoreDriver`* examples.

```
...
driver = pymcsapi.ColumnStoreDriver()
bulkInsert = driver.createBulkInsert(db, table, 0, 0)

# Set values for a 2 int column table
bulkInsert.setColumn(0, 123456)
bulkInsert.setColumn(1, 2147483650)

# Write the row
bulkInsert.writeRow()

# Rollback the transaction
bulkInsert.rollback()

# Get the summary
summary = bulkInsert.getSummary()

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

4.3.5 getInvalidCount()

`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. . . except blocks in the *ColumnStoreDriver* examples.

```
...
driver = pymcsapi.ColumnStoreDriver()
bulkInsert = driver.createBulkInsert(db, table, 0, 0)

# Set values for a 2 int column table
bulkInsert.setColumn(0, 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.setColumn(1, 123456)

# Write the row
bulkInsert.writeRow()

# Rollback the transaction
bulkInsert.rollback()

# Get the summary
summary = bulkInsert.getSummary()

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

4.4 ColumnStoreException Class

Currently pymcsapi hasn't that sophisticated exception handling than mcsapi has. All mcsapi ColumnStoreErrors are parsed by SWIG to `RuntimeErrors`.

INDEX