FPC

DevicePlugin

Database

Revision 2021.1.6

Table of Contents

1 Description

Database access via ODBC. Requires corresponding ODBC driver for required database (such as MySQL ODBC Connector for MySQL databases).

2 Connection strings

Most-common connection strings used for ODBC connection.
Refer to www.connectionstrings.com for other connection strings.

2.1 MySQL

```
Driver={MySQL ODBC 5.1 Driver};
Server=<server-address>;Database=<database-name>;
User=<username>;Password=<password>;
Option=3;
```

Requires MySQL ODBC Connector version 5.1. Can be downloaded from MySQL website.

2.2 MSSQL (Microsoft)

```
Driver={SQL Server Native Client 11.0};Server=<server-address>;
Database=<database-name>;Uid=<username>;Pwd=<password>;
```

Requires Microsoft ODBC Driver 11 for SQL Server. Can be downloaded from Microsoft website.

Version 11 is intended for MS SQL Server 2012. ODBC Connector version 10 was up to SQL Server 2008. Don't miss the server-name syntax in the <server-address>: "<server>**\SQLEXPRESS**" for example, where <server> is server-name or server address and SQLEXPRESS is the name of SQL Server on the target machine.

3 Commands

3.1 *idn? (Identification)

```
*idn?
```

Gets the plug-in identification string.

Parameters

No parameters.

Return value

The identification string in standard format "`<company>,<product/name>,<serial-no>,<version>`".

3.2 open (Open connection to DB)

```
open:{ConnString=[string]}{;Queue=[bool]}
```

Connect to a database using standard ODBC connection strings.

Parameters

`ConnString`	[string]	Specification of connection string to connect to a database **Default:** from the configuration
`Queue`	[bool]	When se to true, on PostRec command records will be first stored to buffer before storing to database. The record will be removed from this buffer after success storing to the database. **Default:** false

Return value

No return value

3.3 close (Close connection)

```
close
```

Closes currenty opened ODBC connection.

Parameters

No parameters.

Return value

No return value

3.4 begin-transaction (Start all commands as transaction until is completed)

```
begin-transaction: failAutoRollback=[ bool]
```

Start transaction on currently opened ODBC connection.
Affects all commands, until is completed by commit/rollback/cancel-transaction or connection disconnect (rollback).

After transaction start, the affected tables will be locked until it is completed.
Precaution 1: After connection lost is the transaction automatically canceled (rollback) on sql server. Plug-in automatically reopen connection, but from safety reason, all following SQL requests will return an error, until transaction is manually canceled !
Precaution 2: *MySQL transactions can be used only on tables which are stored in databases created by engine "InnoDB". MyISAM does not support transactions!*

Parameters

`failAutoRollback`	[bool]	This enable or disable automatic rollback after sql query error. *This does not change the behavior after connection lost!* **True**: do rollback after error **False**: do nothing, rollback or cancel must be called manually

Return value

No return value

3.5 cancel-transaction (Cancel curent transaction without error message)

```
cancel-transaction
```

Cancel (rollback) currently active transaction.
Does not return error, if no transaction was started yet.

Parameters

No parameters.

Return value

No return value

3.6 commit (Commit current transaction)

```
commit
```

Commit current transaction.

Parameters

No parameters.

Return value

No return value

3.7 rollback (Rollback current transaction)

```
rollback
```

Rollback current transaction.

Parameters

No parameters.

Return value

No return value

3.8 Assisted writing

Assisted writing makes easier to put statistics records to the database (typical usage). The programmers defines the target table-name and default values of specified fields (for example Station name, Operator ID, Date, etc..) at the beginning and then only changes values of set of required field (results for example) and calls single method to add a new record to the database. The corresponding SQL query is built internally.

Typical process:

1. set table-name (at the beginning)
2. set default values (at the beginning)
3. change result values (in the process)
4. save record (in the process), goto step 3 and repeat

Steps 3 and 4 are repeated until testing is stopped.

3.8.1 table (Set table)

```
table:<name>
```

Sets currenty used table for assisted writing.

Parameters

`name`	[string]	Table name

Return value

No return value

3.8.2 values (Set values)

values:value$_0$=[string]{:value$_1$=[string]:...:value$_N$=[string]}{:defaults}

Sets values and default values for assisted writing.

Parameters

`value`	[string]	Set of values (name=value)
`defaults`		Switch to make passed values default

Return value

No return value

Examples

```
values:Station="A123":Operator="FPC":defaults
```
Set default values with names"Station" and "Operator" and specified values.

```
values:<clear>
```

Clear values and/or default values for assisted writing.

Parameters

`clear`	[enum]	One of following clear command: • `ClearAll` - clear all values • `Clear` - clear non-default values only • `ClearDef` - clear default values only

Return value

No return value

3.8.3 postrec (Post record)

```
postrect
```

Posts new record to database in assisted writing.

Parameters

No parameters.

Return value

No return value

3.8.4 queue (Queue control)

```
queue:count
queue:clear
```

Controlling the post-record queue (to use queue set the Queue parameter of command open to true):
- Get number of records to be stored
- Cleares all pending records and kill the storing thread

Parameters

No parameters

Return value

When "queue:count", the return value is count of of records to be stored.

3.9 sql (Execute SQL query)

```
sql:<sql>{:reader=[bool]}
```

Performs the SQL query on currently opened DB connection.

Parameters

`sql`	[string]	SQL query to be executed
`reader`	[bool]	Run the reader on executed SQL query (read results) **Default:** false

Return value

When reader=true, the return value is the first single value of the result vector/array. Otherwise the result value is number of affected rows.

Examples

`sql:"SELECT COUNT(*) FROM customers":reader=true`
Execute "SELECT" SQL query and run the reader. Return value will be in this example number of all rows in the table 'customers'.

`sql:"INSERT INTO customers (id,name) VALUES (100,'FPC s.r.o.')"`
Execute "INSERT" query and do not run the reader. Return value will be number of affected rows (1 in this case).

3.10 sql-ret? (Get value from result array)

```
sql-ret?
```

Returns result array size. Any SQL query with reader must be executed first.

Parameters

No parameters.

Return value

Result array size in format "`col,row`", for example "`5,2`" (5 columns, 2 rows).

```
sql-ret?:<col>{:<row>}
```

Returns specified value from result array by specifying column and row index (zero based).

Parameters

`col`	[int]	Column index in range 0 to (max column) - 1
`row`	[int]	Row index in range 0 to (max row) - 1. **Defrault:** 0

Return value

Value from results set of specified column and row.

Examples

`sql-ret?:2`
Return value of third column (zero-based => index 2) of first row.

`sql-ret?:2:1`
Return value of third column of second row.

3.11 acu (Insert accumulator)

Accumulate multiple rows insert for table(s) into memory buffer and commit or rollback when it is required. Create only one query for sql database, therefore is much faster than insert multiple rows sequentially. Assembled query string can be obtained before was committed or after commit fails.

Examples

```
acu:define:table="results":"sn":"time":"who":"count":"retest"
acu:add:table="results":"0001":"@CURRENT_TIMESTAMP":"operatorA":"@415":"2"
acu:add:table="results":"0002":"2019-07-31 15:35:00":"operatorB":"1":"0"
    ..add more rows..
acu:commit / acu:rollback
```

3.11.1 define (Define table and columns)

```
acu:define:{Table=[string]}{;Mysql=[bool]}:<Column0>:<Column1>:...:<ColumnN>
```

Define memory table with specified columns.
Can be defined more tables with different name at once.
Trying to define table which was defined before, old table will be overwritten and all data from it lost without warning.

Parameters

`Table`	[string]	Name of table, where will be stored data. **Default:** "fpc", or any last usage of acu:<cmd> with non-empty table parameter
`Mysql`	[bool]	If is target database MySQL, the table and column names

		will be added to query string with commas at start and the end \`<table/column>\`. **Default:** false
`Column0...ColumnN`	[string]	Definition of table columns where will be stored data.

Return value

No return value

Examples

`acu:define:table="results":"sn":"time":"who":"count":"retest"`
Define a table "results" with columns "sn", "time", "who", "count", "retest".

3.11.2 add (Add row to buffer)

```
acu:add:{Table=[string]}:<Data0>:<Data1>:...:<DataN>
acu:addk:<key>:{Table=[string]}:<Data0>:<Data1>:...:<DataN>
```

Add row into memory buffer. Data must be in exactly same sequence, how was defined in target table.
The "addk" variant uses a key to identify the row, so its possible to overwrite the existing row, when a key already exists in the collection.

Parameters

`Table`	[string]	Name of target table, where will be stored data. **Default:** "fpc", or any last usage of acu:<cmd> with non-empty table parameter
`Data0...DataN`	[string]	Data stored into column in exactly same sequence as target table. Every single <DataN> will be inserted as string in quotation marks '0001'. If is needed add <DataN> as number or use the sql keyword, string must **start with** @ character and second character must not be @. If is needed add string with @ character on start, just double it.
`key`	[string]	Key to identify the row, used by "addk" variant.

Return value

Count of stored rows after insert this row.

Examples

`acu:add:table="results":"0001":"@CURRENT_TIMESTAMP":"operatorA":"@415":"2"`
Add these data as a row into table "results" where columns was "sn", "time", "who", "count", "retest".
Output query will be: `INSERT INTO `results` (`sn`,`time`, `operator`, `who`, `count`, `retest`) values ('0001', CURRENT_TIMESTAMP, 'operatorA', 415, '2');`

3.11.3 replace (Replace string in the buffer)

```
acu:replace:<find>:<replace>{:table=[string]}
```

Add row into memory buffer. Data must be in exactly same sequence, how was defined in target table.
The "addk" variant uses a key to identify the row, so its possible to overwrite the existing row, when a key already exists in the collection.

Parameters

`table`	[string]	Name of target table, where will be stored data. **Default:** "fpc", or any last usage of acu:<cmd> with non-empty table parameter
`find`	[string]	String to be find and replaced in the buffer.
`replace`	[string]	The string which will be used as replacement of found string.

Return value

No return value.

Examples

`acu:replace:"%id":"123"`
Replace all `%id%` strings in the buffer by `123` string.

3.11.4 batch-insert (Insert buffer into database)

```
acu:batch-insert:{Table=[string]}
```

Does insert of one table as one query limited only by SQL setting (Mysql default is 64MB).
When commit succeed, data from memory will be removed, but table definition stays.
When commit fails, table definitions and their data stay in memory.
With command get-data? can be returned query which was unsuccessfully used.

Parameters

`Table`	[string]	Table name to commit. **Default:** "fpc", or any last usage of acu:<cmd> with non-empty table parameter

Return value

Returns the query character count.

3.11.5 batch-insert-many (Insert buffered tables into database as transaction)

```
u:batch-insert-many:{Tables=[string-array]}
```

Does insert of many or all tables as one query via transaction. Query limited only by SQL setting (Mysql default is 64MB).
When succeed, data from memory will be removed, but table definition stays.
When fails, tables definitions and their data stay in memory.

Parameters

`Tables`	[string-array]	Table names, separated by ':' or ';'. Without the parameter will be used all defined tables. **Default:** NONE

Return value

Returns count of used tables.

3.11.6 rollback (Discard buffer)

```
acu:rollback:{Table=[string]}
```

Remove all data rows from selected table memory. Table definition stay untouched.

Parameters

`Table`	[string]	Table name to rollback. **Default:** "fpc", or any last usage of acu:<cmd> with non-empty table parameter

Return value

No return value

3.11.7 clear (Remove buffer and table definition)

```
acu:clear:{Table=[string]}
```

Remove table definition with associated data from memory.

Parameters

`Table`	[string]	Table name to rollback. **Default:** "fpc", or any last usage of acu:<cmd> with non-empty table parameter

Return value

No return value

3.11.8 clear-all (Remove all deifinitions and buffer)

```
acu:clear:{Table=[string]}
```

Remove all tables definition with associated data from memory.

Parameters

`Table`	[string]	Table name to rollback. **Default:** "fpc", or any last usage of acu:<cmd> with non-empty table parameter

Return value

No return value

3.11.9 get-data? (Gets current memory query string)

```
acu:get-data?:{Table=[string]}
```

Return sql query for current table state.

Parameters

`Table`	[string]	Table name to commit. **Default:** "fpc", or any last usage of acu:<cmd> with non-empty table parameter

Return value

Sql query.