Warning!
This module exposes advanced functions, use only if you know what you are doing.

system.db.storeHistory

Description

Insert a record to a database table, using the Store&Forward system. With the optional timeout parameter set to a value > 0, this function will behave similar to a Block Transaction Group. Subsequent calls with the same datasource and table will create a block of rows, which is forwarded to the database in a single statement.

The function does not check if the given table name exists, or if the used columns are present or have the correct datatype. Errors will result in exceptions thrown by the Store&Forward system.

Unlike the Transaction Groups, this function stores timestamps (value.timestamp) as UTC with millisecond resolution. Convert the timestamp to a string to store it to a DATETIME column.

When using timeouts, all projects share one instance of the script module, so if storeHistory is called form different projects for the same table, records from both projects will be grouped together. A project save will not cause the timeout to expire, but a gateway shutdown will.

Syntax

system.db.**storeHistory(**database, tableName, columnNames, values[, timeout[, retrigger]]**)**

Parameters

String `database` - The name of the database connection to execute against.
String `tableName` - The name of the database table to INSERT a record.
String[] `columnNames` - A list of column names. When using timeout, the column order has to be the same in subsequent calls.
Object[] `values` - A list of values to write to each column specified. Timestamp values are stored as UTC milliseconds.
long `timeout` - If this parameter is >0, subsequent calls to this functions will be aggregated into a single database update. Timeout starts when the first row is stored with timeout > 0.[optional]
boolean `retrigger` - Retrigger timeout. Be careful, when records are stored continously with retrigger=true, they will never be forwarded to the database.[optional]

Returns

booleanTrue if the record has been stored. This is no indication of a succesful forwarding to the database.

Scope

Gateway

Examples

This code will store a value in a tag change script. To be of practical use, the id has to be determined, e.g by evaluating the tag path.

```
columns = ["id", "t_stamp", "value"]
values = ["1000", newValue.timestamp, newValue.value]
system.tag.storeHistory("DB","MyTable",columns, values, 5000)
```

system.db.forwardHistory

Description

Cancels a previously started timeout and stores the given record group immediately.

Syntax

system.db.**forwardHistory(**datebase, tableName, columnNames**)**

Parameters

String `database` - The name of the database connection to execute against.
String `tableName` - The name of the database table.
String[] `columnNames` - A list of column names. The column order has to match the storeHistory call.

Returns

booleanTrue if the record has been forwarded, false if no record was found. This is no indication of a succesful forwarding to the database.

Scope

Gateway

system.util.addGatewayScriptModule

Description

Adds a script module to the gateway context and makes it available to use with the runScript() expression in SQLTags. This function should be called once in a gateway startup script.
The module is registered in the 'app' package with its simple name, packages are ignored.

Warning: This functions links modules defined in a project to tag evaluation in gateway scope. This might result in total chaos and may prevent overall tag execution!

Syntax

system.util.**addGatewayScriptModule(**module**)**

Parameters

ScriptModule `module` - The module to add to the gateway context.

Returns

Nothing

Scope

Gateway

Examples

This code will register the script module app.myPackage.myModule. Note that the package definition gets lost.

```
import app.myPackage.myModule
system.tag.addGatewayScriptModule(app.myPackage.myModule)
#use with runScript("app.myModule.func1(myParam)")
```