merbon
DATABASE

Implementation guide

11/2023

domat
control system

Contents

1 Introduction

1.1 Merbon DB

Merbon DB works as a service or console application in Windows operating systems. It allows storage of data by client applications, such as Merbon SCADA, PLCs, or 3rd party systems, over an open interface (web service, SOAP).

Thanks to several data compression algorithms, minimum file sizes on the disk are achieved. All files can be copied at any time when the application is running, with no risk of data damage or loss.

Merbon DB is designed as a highly scalable application, allowing to store hundreds of thousands of variables (data rows) with no loss of performance. The requirements on system hardware are low comparing to common database systems (SQL, noSQL).

When upgrading the SCADA and data storage system, or transiting from standard history file storage in RcWare Vision to a database solution, it is possible to import the old data. The import may be executed while new data are recorded, so no information is lost.

The database provides web interface for administration, user management, and diagnostics.

1.2 Implementtion guide

This implementation guide describes the process of deploying and configuration of the database.

1.3 Terminology

variable – an entity representing a data point in a control system. Representation of a data point in the database.

data point – source of values for sampling. It is for example a temperature sensor, pump state, or water meter readout. One (physical) meter may contain more data points, e.g. an energy meter provides total consumption, actual power, 3 phase voltages, 3 phase powers, furequency, power factor, etc.

sample – a record of value of a variable at a particular time

data row – a set of samples related to a variable.

2 Installation

The installation is described in the Merbon SCADA Installation manual. The database uses following TCP ports:

- saving data into database and reading is performed over API on port 9876
- web interface for administration is on port 11112
- Merbon Database Adapter: access over port 8686

Hardware requirements are as follows:

- OS Windows Vista and more recent
- memory allocated for Merbon DB is minimum 1 GB RAM (enough for tens of thousands of variables saved every 1 min.; further e.g. for 1 200 000 is 10 GB RAM enough), total memory size must be larger to allow smooth operation of Windows and other programs and services
- Processor: recommended minimum 2 core

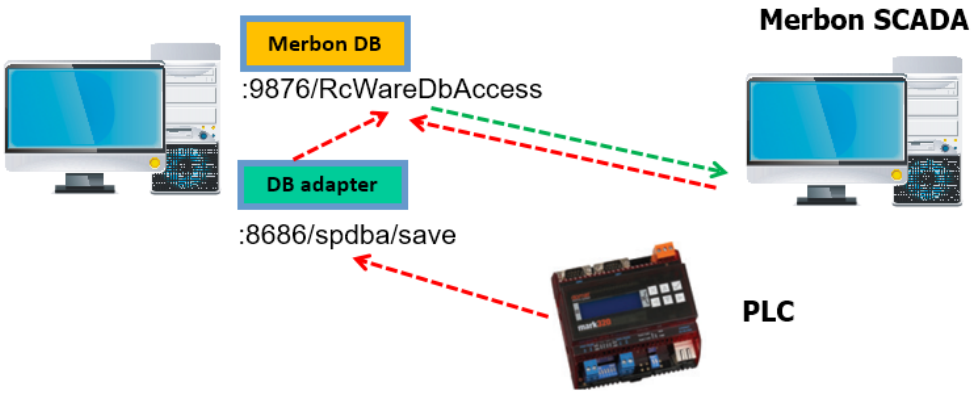
Example of PC configuration for a typical usage:

RAM: 4GB + 1GB for each 50 000 variables (data points).

Disk: 500 GB + 2GB per year per 10 000 variables saved every 3 minutes

2.1 Merbon Database Adapter

Merbon Database Adapter is a service which creates an endpoint for connection of PLCs which have to write to the database. Merbon Database Adapter receives that data from the PLCs, processes, and writes to Merbon DB over its API.



Flow of history data between a PLC, database, and Merbon SCADA. Red: writing, green: reading.

Merbon Database Adapter does not require a licence, and may be installed on another PC than the database is. However, usually there is no reason for that, unless it is required by the IT security policies (the PC with the database shall not be visible for hundreds of PLCs which write the data, there is only a single connection to the database API: that from the Merbon Database Adapter). The default URL of the target database is http://localhost:9876/RcWareDbAccess, can be changed in the configuration file *C:\Apps\Merbon\Database Adapter\Web.config.*

URL of the Merbon Database Adapter is http://[IP address of the server]:8686/spdba/save - and this must be set up in the PLC configuration, such as:

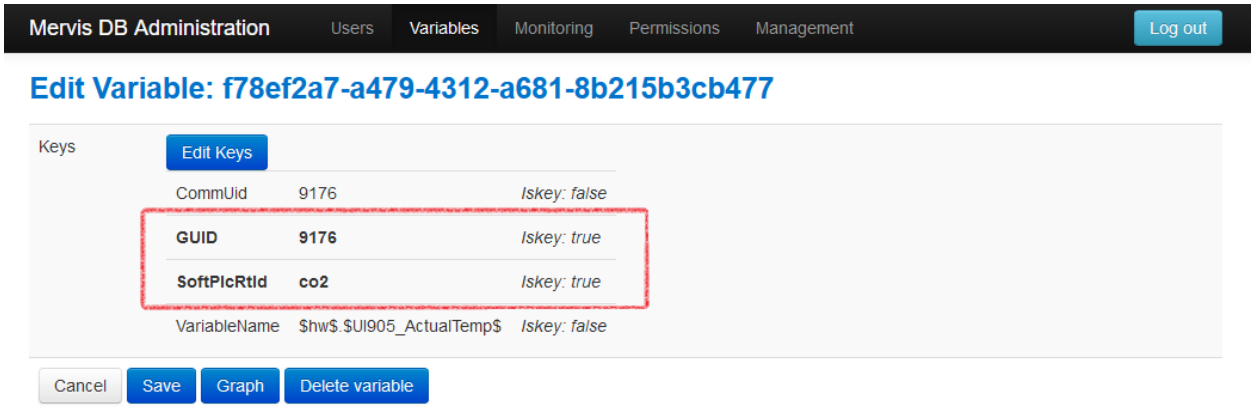
Merbon DB Parameters	
Enabled	True
URL	http://192.168.1.44:8686/spdba/save
Communication Period [min]	1
User Name	rc_user
Password	rc_user
Note	
Proxy Parameters	

User and password (here rc_user / rc_user) must be defined in the database. Remember to set up in the PLC also

- **default gateway** and **DNS server address** in case that the database server (or, server with the Merbon DB Adapter) is outside the local network
- **runtime identification for history** – this string is one of the keys in the database. The Runtime ID length should not be bigger than 31 characters.

3 Keys in the database

A key in the database is an information (or group of informations) which fully identify an item in a database (or in a part of it). A primary key is an unique identifier of an item in a table, in which it is defined. In Merbon DB, an item is unambiguously identified by a set of „key“ keys.



For example, at variables written by a Merbon RT can be seen in the variable properties (the „Edit“ button at the variable) that a variable has two keys: SoftPlcRtId and GUID.

SoftPlcRtId: identification of runtime. Every PLC which has to write into a database must have an unique RTID, or Runtime identification. This identification – a text string – is to be set up in Merbon IDE, in the PLC properties in the *History Runtime Identification* property. Its length should not exceed 31 characters.

GUID: variable identifier within a runtime. When the project is compiled, every variable (or structure, such as function block, array etc.) is assigned a unique GUID (which actually is the position of a variable in the PLC memory).

When the application software is changed (e.g. a sensor is connected to another input), in the PLC the key values may change, including the primary keys. Clients which use the database data must be aware of this, and reconstruct the data rows – typically, use the variable names rather than GUIDs for variable identification. See also below in *Setting of database writing in Merbon IDE a v SoftPLC.*

4 Management and setup

The database is typically managed using its web interface, which is accessible at http://localhost:11112/admin. Default credentials are:

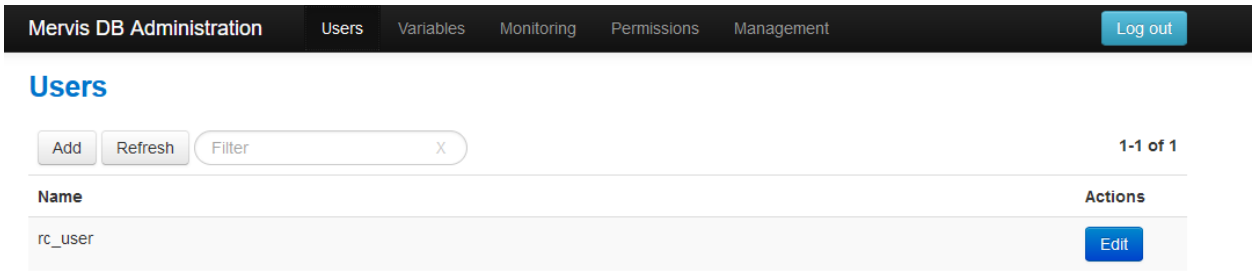
User: **admin**

Password: **rw**

After login, a page opens with menu in its upper part.

4.1 Users

User management is in the *Users* tab:



The first step should be defining of the „working“ users which will be used for data reading and writing. Their credentials are then entered in the client systems, such as PLCs, 3rd party programs, etc. The user named **admin** should only be used for database management. A typical

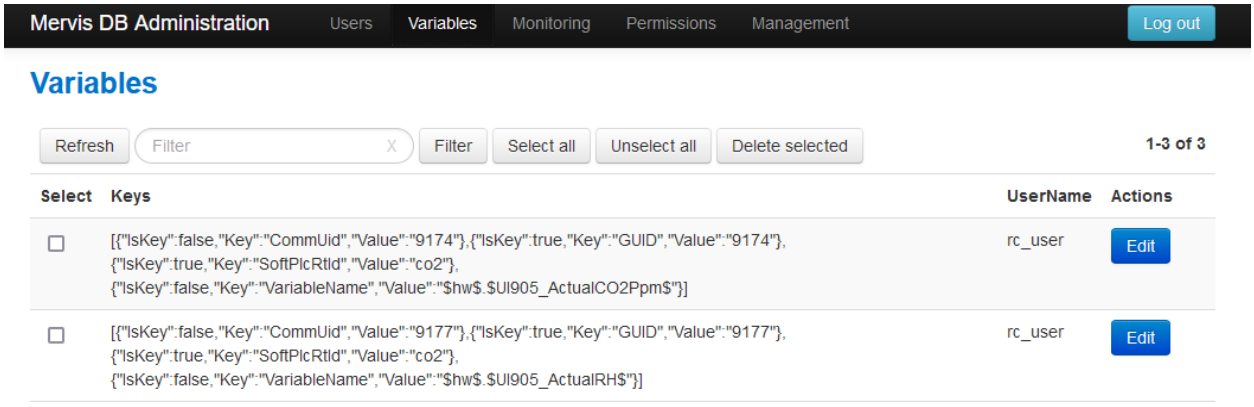
user in the examples below is **rc_user** / **rc_user**. Users should have their password changed from the defaults for security reasons; all common rules for safe password definition apply here.

The credentials for admin are saved in the configuration file as a plain text. Similarly, if a PHP connection string is used, there are SQL database login credentials there. Therefore it is necessary to prevent unauthorised access to the folder with the configuration file.

The *Edit* button allows changing the password.

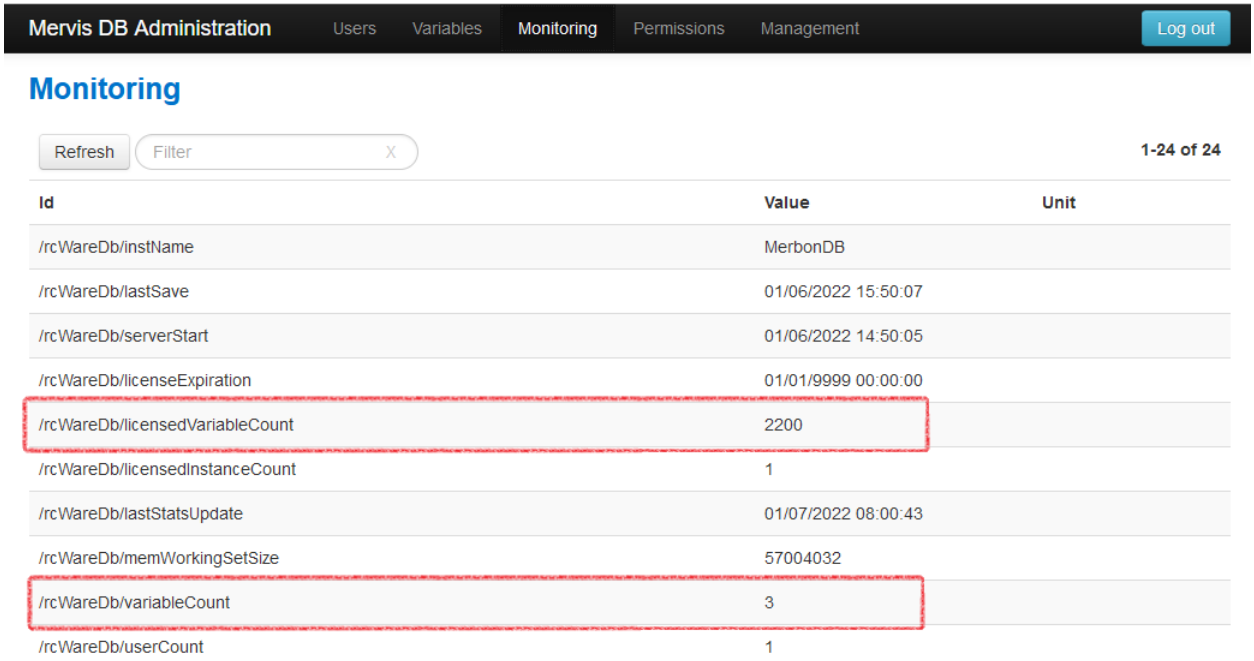
Users can not be deleted, because the user names are (or may be) bound to data rows. Only passwords can be changed.

The *Variables* tab contains a variable browser.



Variables (data rows) may be edited, displayed in a graph, and deleted. Variables may be checked by the *Select* checkbox and then deleted using the *Delete selected* button. Changes can not be taken back, so beware of what you are doing. Larger operations, such as deleting of hundreds of data rows, can be performed by special utilities. Contact technical support at support@domat.cz if necessary.

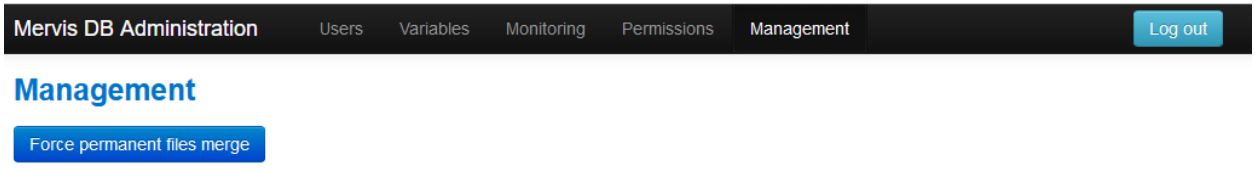
The *Monitoring* tab lists the system variables and other useful data.



The most important items for diagnostics are number of licensed variables, and number of variables in the database. If the licensed variable number is exceeded, the database switches to read-only mode, and no more variables are written! Therefore, regular checks are recommended, especially if there are enhancements in client programs (PLC, SCADA) to be done. The licence may be extended if necessary.

There is no explicit warning if the licensed number of variables is exceeded, as the database has no common user interface. The number of variables should then always be checked before a planned extension of the system (adding more data rows into the database), and also regularly, e.g. after each 6 months of operation.

The *Management* tab contains the *Force permanent files merge* button.



This function invokes merging of temporary files to permanent files. It is a „harmless“ operation which can be executed at any time, typically before the database has to be backed up or the database files moved to another disk. In normal operation, the files are merged as soon as the temporary files reach the maximum size specified in the configuration file.

5 Database configuration

All configuration parameters are in the configuration file, *ESG.Db.Server.Host.exe.config*, located in the folder with the database system files. The default folder is *C:\Apps\Merbon\Merbon Database*. All parameters are read up when the service is started. If there are changes in the configuration file, stop and start the database service (e.g. using the Merbon SCADA Installer) after the file is saved.

Some parameters that can be used for database optimization are described below. The rule of thumb is that if there is no good reason for change, the parameters should be kept on their defaults. A parameter can be easily located by searching of its key text in the configuration file.

5.1 File paths

There are three paths to files in the configuration. At installations up to 40 – 50 000 data points all files can be kept in their default folders. With more data points, the file access can be optimized by moving the files to a storage with faster access:

InfrastructureDataPath

```
<add key="InfrastructureDataPath" value="c:\apps\merbon\Merbon Database Warehouse" />
```

Path to infrastructure files (list of variables, list of users, status information). The files are not accessed frequently and are of middle size, however, it is good to have fast sequential read and write access to this storage. It will make the database start faster.

LogsPath

```
<add key="LogsPath" value="c:\apps\merbon\Merbon Database Warehouse" />
```

Here, the data are stored in an unoptimized form – the most recent data before moving into permanent files. The files are not large, about tens of MB. They are read and written frequently. It is good if this folder has a fast read and write access when the database is started and stopped.

This is not the path to database system event logs – see below.

PermanentFilesPath

Permanently stored data, mostly read-only. Big files, which are written only seldom. This folder should have fast random access for reading.

```
<add key="PermanentFilesPath" value="c:\apps\merbon\Merbon Database Warehouse" />
```

Database log files

Records on system events of the database, listings of errors. Useful for diagnostics if e.g. the database can not be started or stops operation unexpectedly.

```
<add name="circular" type="ESG.Core.Logging.CircularTraceListener, ESG.Core" initializeData="c:\apps\merbon\Merbon Database
```

```
Logs\rcwaredbserver.log" maxFileSizeKB="100"
traceOutputOptions="DateTime"></add>
```

5.2 Ports

TCP ports for the services should not be changed deliberately. If necessary, the port numbers are to be edited in these lines of the configuration file:

9876 – API for client programs, data reading and writing

- `<serviceMetadata httpGetEnabled="true" httpGetUrl=http://localhost:9876/RcWareDbAccess`
- `<endpoint contract="ESG.Db.Server.Shared.IHistoryAccess" address="http://localhost:9876/RcWareDbAccess" binding="basicHttpBinding" name="HistoryAccess" bindingConfiguration="HistoryAccessHttpBinding" bindingNamespace="http://dev.rcware.eu/esgdb_HTTP" />`
- `<endpoint contract="ESG.Db.Server.Shared.IHistoryAccess" address="http://localhost:9876/RcWareDbGZipAccess" binding="customBinding" name="HistoryAccessGZip" bindingConfiguration="HistoryAccessGZipBinding" bindingNamespace="http://dev.rcware.eu/esgdb_HTTP_GZIP" />`

(This port number has to be changed at three lines of the configuration file.)

11111 – monitoring

```
<endpoint contract="ESG.Core.ServerMonitor.Shared.IWebDataRetrieval"
behaviorConfiguration="ServerMonitorWebBehavior"
bindingNamespace="http://dev.rcware.eu/ServerMonitor_REST"
binding="webHttpBinding" bindingConfiguration="WebHttpsBindingConfig"
address="http://localhost:11111/" />
```

11112 – web interface for administration

```
<endpoint
contract="ESG.Db.Server.Engine.Administration.IAdministrationApi"
behaviorConfiguration="ServerMonitorWebBehavior"
bindingNamespace="http://dev.rcware.eu/Administration_REST"
binding="webHttpBinding" bindingConfiguration="WebHttpsBindingConfig"
address="http://localhost:11112/" />
```

5.3 Depth of event logging

```
<add name="sourceSwitch" value="All" />
```

Possible values are `Information, Warning, Error, All`. If the database runs without problems, it is advised to set this parameter to Error, which prevents from creating unnecesary large log files. All log files can be deleted manually at any time.

5.4 File sizes

An important parameter for optimisation.

The file sizes are specified in powers of two, e.g.

20 ... 1 048 576 (1 MB)

...

30 ... 1 073 741 824 (1 GB)

Maximum is

32 ... 4 294 967 296 (4 GB)

Every file type has two parameters:

...FileSize

If this size is exceeded, a new file is created. The new file has increased number in its file name.

...FileGrowth

Steps in size for allocation data into files.

The file types are as follows:

Users... – files with user definitions, hashed passwords, etc.

Variables... – lists of variables and their metadata

PersLog... – files for fast temporary data storage, before saving into permanent files

PermFileSet... – files for permanent data storage

Perm... – At databases with 100 000 variables and more, it is good to keep these files large, because this keeps the number of indices low and saves memory. For example, for 1 000 000 variables, a 4 GB file should be defined (value of 32 in the configuration file). On the other hand, increasing of permanent files slows down the merging of permanent files with logs (buffers of received values): e.g. a large file (4 GB) is extended by a small piece of data (64 MB) and the result has to be saved, which is consuming PC resources.

Index... – index files. If indices like .s0, .s1 etc. appear in the folders, it is advisable to increase the index file size.

5.5 Others

MaxDistributionQueueLength

Do not change this parameter, the database response may slow down by up to 99.9 % if the queue length is decreased.

Other parameters are described in the comments in the configuration file. The usually need not to be changed.

6 API

API (Application Programming Interface) is an interface which the database provides for 3rd party applications. In case of Merbon DB, it is a description of communication between a 3rd party system, such as a metering readout system or SCADA, and the Merbon DB. The complete documentation to the Merbon DB API is provided on a separate basis.

It reads and saves values of those types: Double, Boolean, Integer, DateTime, String, and Blob. Each value is saved together with its time stamp, or with a ”Good Through” value, which is date and time, by which the value has not changed and is still valid. (This saves database space.) The client should also provide information about the saving interval with each saved value. The basis of API are methods **GetData** and **SaveData**, together with a special method **GetParticularData**.

6.1 Basic principles

When processing text strings, letters are case-sensitive.

Time stamps, Good Through, and time parameters of all functions are given in UTC.

The complete API description is available as a WSDL file.

6.2 Variables identification

Every client which saves data using the **SaveData** method must decide how to uniquely identify a value, or its belonging to a variable.

A Variable is identified by a set of keys and values, where there is a set of keys which represents an unique combination which identifies a variable unambiguously.

Example:

ValueA

- key1: value1, isKey: **true**
- key2: value2, isKey: false
- key3: **value3**: isKey: **true**
- value: 200
- time stamp: UTC 12.01.2009 10:00:00

ValueB

- key1: **value10**, isKey: **true**
- key2: value20, isKey: false
- key3: **value30**, isKey: **true**

- time stamp: UTC 12.01.2009 10:00:00

ValueC

- key1: **value10**, isKey: **true**
- key2: value60, isKey: false
- key3: **value30**, isKey: **true**
- time stamp: UTC 12.01.2009 11:00:00

In this example, the recorded values belong to 2 different variables. Values B and C belong to the same variable, because the ”key” keys (isKey = true) have the same values (key1 = 10, key3 = 30).

6.3 Reading of data

As there may be many millions of values saved for a particular variable in the Merbon DB, the requests for values are limited by size and they can be enlarged by increasing of the offset parameter (as returned in the reply of the server) to read more data.

It is possible to request for more variables at the same time, so for iteration ”over the variables” there is another offset parameter, too.

The ValueItem class description

- `"Hvt" – HistoryValueType – value type (Double, Blob, String, Int64, NotDefined, ISODateTime, Boolean)`
- `"Ivl" – Interval – interval in which the value was saved`
- `"Ts" – UtcTimeStamp – time at which the value was valid`
- `"Gt" – GoodThrough – time by which the value was valid`
- `"Bv" – BooleanValue`
- `"Dv" – DoubleValue`
- `"Iv" – Int64Value`
- `"Sv" – StringValue`
- `"BinV" – BlobValue`
- `"Dtv" – DateTimeValue`

Example in PHP

The example is for illustration only. For implementation, more documentation is needed, which will be provided by support@domat.cz.

```
<?php /*Example showing values readout from RcWare DB over API v2.0.*/
ini_set("soap.wsdl_cache_enabled", "1"); //caching of wsdl – better set in php.ini
try
{//URL – copy from RcWare and add ?wsdl
//if we use file wsdl – replace all strings db.rcware.eu:9877 by real address
  // and port of the application
  $soapClient = new SoapClient('./wsdl/RcWareDbAccess.wsdl.xml', array('trace' => 1,
```

```
'features' => SOAP_SINGLE_ELEMENT_ARRAYS));//trace = 1 - for debug

  $credentials = array('Name'=>'username', 'Password'=>'password'); //Copy from RcWare

  $utcTZ = new DateTimezone('UTC'); //all time stamps should be in UTC

  $utcFrom =  new DateTime('-6months', $utcTZ ); //starting point of the data we want
  $utcTo =  new DateTime('now', $utcTZ ); //ending point of the requested period
  $valOffset = 0; //let us start from the first value
  $valCount = 10; //and step by XX values (optimum value is about 3000)
  $varOffset = 0;
  $varCount = 10;

//RcWare DB identifies each variable by a set of keys and values.
//The complete set must be unique for each variable (necessary when saving data),
// but when reading data, the values can be requested also by a non-unique subset of keys and
values.
//Then, all values of all variables which meet the specification, are returned.
//Therefore I can ask for more variables at the same time, they just must be specified here
//DPGuid is enough for identification – it is unique within a RcWare project
  $variablesKey = array(array(array('IsKey'=>true, 'Key'=>'DPGuid',
'Value'=>"2C7F333A-85BA-465B-A7AD-8858B4EBB2F2")));

  echo "We ask for values from: {$utcFrom->format('c')} until: {$utcTo->format('c')}\n\n";

do//readout of data from the server
  {
    $response = null;
do
    {
//the other party unfortunately does not interpret the time zones correctly if only the 'c'
parameter is used...
      $response = $soapClient->GetData(array('credentials'=>$credentials,
'variablesKey'=>$variablesKey,
'utcFrom'=>$utcFrom->format('Y-m-d\TH:i:s\Z'),
'utcTo'=>$utcTo->format('Y-m-d\TH:i:s\Z'),
'valueOffset'=>$valOffset,
'valueCount'=>$valCount,
'variableCount'=>$varCount,
'variableOffset'=>$varOffset));
      $valOffset = $response->nextValueOffset;

      foreach($response->GetDataResult->Mvr as $varArray)
      {
       foreach($varArray->Vals->I as $val)
        {
          echo print_r($val);
        }
      }
    } while($response->nextValueOffset != -1);

    $varOffset = $response->nextVariableOffset;

  } while ($response->nextVariableOffset != -1);
}
catch (Exception $e)
{
```

```
//print_r($soapClient->__getLastRequest()); //needs "trace = 1"
  print_r($e);
}
?>
```

Writing of data and other functions are described in the electronic API documentation.

7 Data pump EC – one-off data transfer

The data pump is a simple program which allows to copy data between various sources. A source may be

- a Merbon DB database
- a RcWare Vision nebo Merbon SCADA folder with history files (.txl)
- a general SQL database.

The program is called from a command line. Its parameters specify the activity to be executed, and endpoints for the data transfer. It is advised to create a .bat file with the complete command, and run it in a command line window. All parameters are to be entered in a single command line. Some parameters are introduced by a „ - “ character.

Even if the data pump would be executed several times with the same (or overlapping) parameters over the same databases, the data are not duplicated in the target database, they are written only once. During the data transfer process, which may last for hours or days, the target database is able to accept data from all clients. This allows to recover or move a living database on another machine without any interruption of the services.

7.1 Example of history data import from RcWare Vision into Merbon DB

The most frequent situation. It is used if a RcWare Vision project with history saving into text files is moved to a RcWare Vision with saving data in a Merbon DB (or, in other words, adding of a Merbon DB into an existing RcWare Vision), or when an old RcWare Vision project is migrated to Merbon SCADA using a Merbon DB.

```
EsgDbDataPump.exe importType:file directory:"c:\temp\safranka\DATA"
endPointName:"HistoryAccess" destination:http://localhost:9876/RcWareDbAccess
destinationUser:rc_user destinationPsw:rc_user from:"2010/1/1 00:00"
to:"2010/12/31 00:00" historyFileType:all -skipError
```

7.2 Example of moving data from a SQL database to Merbon DB

It is used when an existing RcWare Vision installation used a general SQL database to save data. From this database, the data are now to be copied to Merbon DB.

```
EsgDbDataPump.exe importType:sql directory:"c:\RcData\DATA"
connectionString:"Data Source=local;Integrated
Security=No;user=rc_user;password=rc_user;" endPointName:"HistoryAccess"
destination:http://localhost:9876/EsgDbAccess destinationUser:root
destinationPsw:root from:"1970/1/1" to:"2009/10/8 12:30" -skipError
```

7.3 Parameter list and usage

importType: esgdb, sql, file, datalogger – data import type; according to it, the other parameters are used.

- esgdb – another Merbon DB database
- sql – SQL database into which RcWare Vision wrote data (rare)
- file –RcWare Vision or Merbon SCADA history files (most frequent)

source:http://172.16.0.45:9876/EsgDbAccess – data source URL
sourceUser:rc_user – data source user
sourcePsw:rc_user - data source password
destination:http://localhost:9876/EsgDbAccess – data target URL
destinationUser:root – data target user
destinationPsw:root – data target password
from:1970/1/1 – start date (format yyyy/m/d)
to:2009/8/1 – end date (format yyyy/m/d) – may be in the future
readBlockCount:10000 – how many values to read in one request
writeBlockCount:1000 – how many values to write in one SaveData operation
skipError – continue if the server saves less data then it was sent to it by the pump; if this parameter is not active, the import ends
forceRepeat – repeatedly tries to save data which could not be saved, until successful. Waits 30 s between tries
readVariableFromSource: - The data pump reads (using the same user name and password) the list of variables from the target database, and imports only values of those variables which already exist. Used to avoid copying unnecessary data (which may increase the number of used datapoints).
useGetData – uses the method *GetData* to read data from the source, this is only supported up to database version 3.2 – the newer versions support *GetVariableArrayData*

directory: - a RcWare Vision data folder, where the history files and *EsgDbImport.inf* containing import information will be searched for (when only certain variables shall be imported, see Selection of variable subset)

connectionString: - connection string to the SQL database, where the RcWare data will be imported from

historyFileType:all,long,short,hystereze – specifies which files are to be imported:

- all – all .tx* files
- long – .txl files
- short – .txs files
- hystereze – .txh files

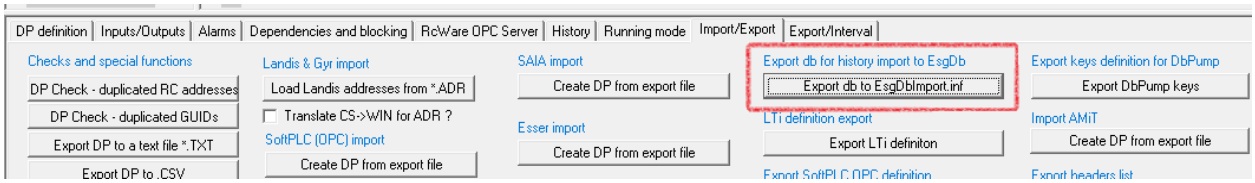
endPointName:"name of endpoint configuration in the .config file" – to identify the connection type to communicate (basicHttpBinding for SOAP services, netTcpBinding for permanent TCP connection to binary data), default is basicHttpBinding. HistoryAccess is a reference to an AppConfig file, which specifies the connection details.

Another example of a data pump command:

```
D:\_pom\DataPump>EsgDbDataPump.exe importType:file
directory:"c:\data\FVE_KOSORIN" endPointName:"HistoryAccess"
destination:http://localhost:9876/RcWareDbAccess destinationUser:rc_user
destinationPsw:rc_user from:"2010/1/1 00:00" to:"2011/8/15 00:00"
historyFileType:all -skipError
```

7.4 Selection of variable subset

When importing from text files, all variables contained in the text files (with samples within the specified time span) are imported. For some applications, it needs to be specified which data points have to be imported (if only some of the data series have to be imported rather than the complete contents of the history files). For this, the definition file *EsgDbImport.inf* must be generated in RcWare Vision in the data point table, in the Import/Export tab.



The proceeding is as follows:

- unlock the data file – in RcWare Vision go to the the data point table, click the padlock icon
- tag (the T key, or in the *Tagging* menu) all data points to be imported
- down in the Import/Export tab click the *Export db to EsgDbImport.inf* button. The file is created in the project folder, where are also the history data files. The data pump reads the file after start, and processes only the variables specified in the file. If the history data (.tx... files) are located in another folder, the *EsgDbImport.inf* file must be copied to the same location.

8 Deleting data

The data should be deleted from a Merbon DB only after thoughtful consideration, and if necessary, e.g. to remove unnecessary data rows to decrease the number of variables if there is a danger of exceeding the number of licensed variables.

8.1 Complete deletion of all data

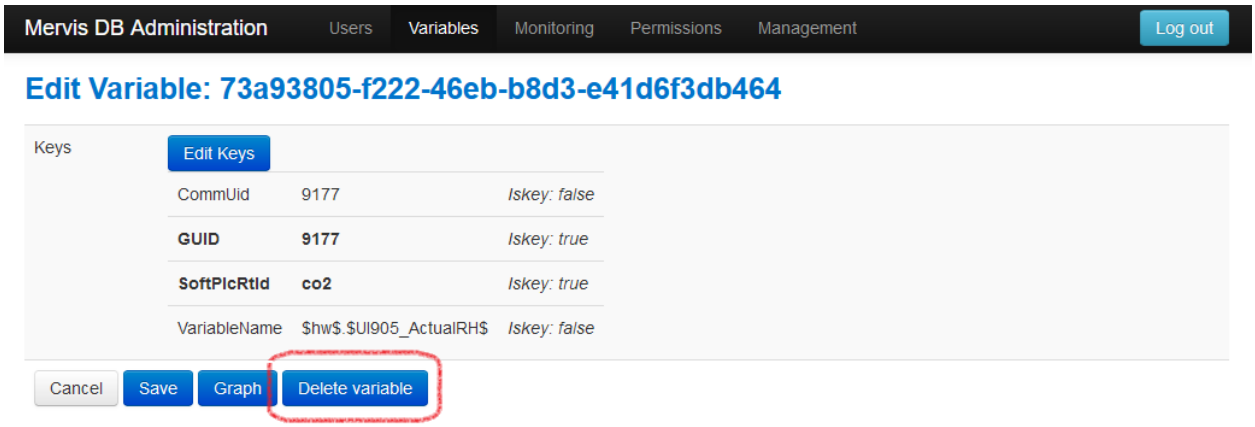
All data of all variables can be easily deleted by removing the files

- variables*.*
- plog_*.*
- pfs*.*

in the *C:\Apps\Merbon\Merbon Database Warehouse* folder (or another folder, if the default folder was changed). Users and their rights will be maintained. The database creates new, empty files after start.

8.2 Deletion of one variable over the web

Simple deletion of a single data row can be executed over the web interface, in the *Variables – Edit* menu by clicking the *Delete variable* button. This function deletes all samples related to a particular variable.



8.3 Selective deletion - DbTool

To delete more variables selectively, use the **DbTool** utility. To read data of all variables, run the utility with the administration endpoint of the selected database (which is TCP port 11112, unless changed in the configuration file) and credentials **for web access to the administration (i.e. not with those of a standard database user who reads and writes data)**. The program must be run with the parameter *csvStatsOutput*, which creates a file with the list of all variables in the database and their properties.

Note.: Open the PowerShell window in the program folder by pressing Shift and right mouse click, then *Open PowerShell window here*.

```
.\ConsoleDBDotNet.exe --db http://localhost:11112/ --user user --
password password --csvStatsOutput out.csv
```

For larger databases with tens of thousands of data rows the export may take several hours. Run the export on a PC which may run uninterruptedly until the process ends, preferably on that one where the database is installed.

The resulting file (here *out.csv*) is then used to specify which variables have to be deleted from the database. Open the file in Microsoft Excel (not by direct clicking on the .csv file; create a blank sheet in Excel and import the data in the *Data – From text/CSV* menu), and **delete the rows with variables which have to be kept in the database.**

ATTENTION, users tend to delete variables from the table which have to be deleted. This is wrong. The variables which have to be kept must be put away. A definition for deleting is created.

The variables which remain in the file will be deleted. Save the file under another name, e.g. *delete.csv*.

In the new file, make the following operations:

- take over the heading from the original file
- if necessary, replace the separating semicolons „;“ by commas „,“

The *delete.csv* file then should look like this in *notepad.exe*:

```
http://localhost:11112/,InnerId,NewestTimestamp,OldestTimestamp,Keys
http://localhost:11112/,73a93805-f222-46eb-b8d3-e41d6f3db464,06.01.2022 15:50:00,06.01.2022 15:38:09,Comml
http://localhost:11112/,f78ef2a7-a479-4312-a681-8b215b3cb477,06.01.2022 15:50:00,06.01.2022 15:38:09,Comml
```

All variables which remain in the file which is used as a parameter for the application will be deleted from the database. The deletion is executed by launching the program with a parameter *csvDeleteInput*:

```
.\ConsoleDBDotNet.exe --db http://localhost:11112/ --user user --
password password --csvDeleteInput  delete.csv
```

The program writes out the IDs of the deleted variables. After it is finished, all variables, whose IDs were in the parameter file, are deleted from the database.

List of parameters:

`--db`	Management URL of the database
`--user`	Database user with admin rights
`--password`	Password of this user

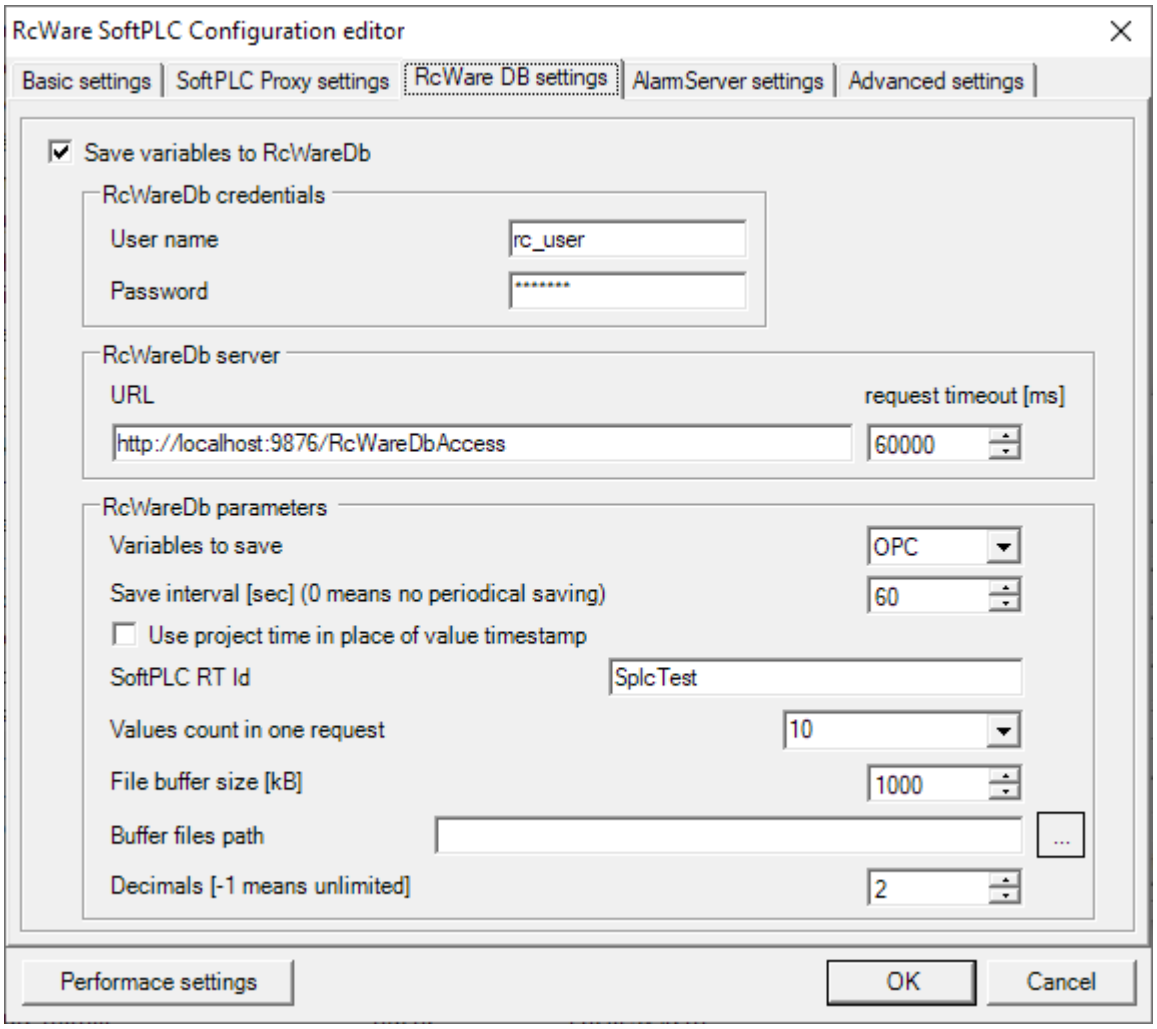
and one of the parameters

`--csvStatsOutput`	lists of all the database variables into a file
`--csvDeleteInput`	deletes the variables defined in the file from the database.

9 Setting of database writing in SoftPLC

Supported platforms are Windows Runtime and IPLC510. Saving data from the PLC into a database is configured as follows:

At **Windows Runtime** the settings is done using the service configurator.



Save variables to RcWare DB: Check to save the data from PLC into a database.

RcWare DB Credentials: User name and password of a database user.

request timeout: How long the PLC shall wait for the reply from the Merbon DB; after the timeout expires, the connection is considered as lost, and the data are buffered locally.

Variables to save:

- All = all variables in the runtime – do not use;
- Used = „nearly“ all variables (e.g. unused block inputs are not sampled) – do not use;
- OPC = this is OK, but remember to tag all relevant variables in SoftPLC IDE as OPC!

Save interval: Interval to sample the values.

Use project time in place of value timestamp: Uses timestamp from the RTC of the PLC rather than the variable update time (which is displayed in the Variables table).

SoftPLC RTID: Thic string is one of the database primary keys, and must be unique for every PLC.

Values count in one request: Number of variables depends on the throughput of the network. Decrease this value if connection is slow.

File buffer size (kB): PLC has about 10 MB free space, thus 1MB is OK (buffering allows to save about 20 000 samples per 1 MB, so the total buffering time depends on the number of samples and sampling interval).

Buffer files path: Location of the buffer files on the disk.

Decimals: Number of decimal places for all Real (Double) values.

At **IPLC510** the writing into database is configured by editing of the configuration file, `/var/opt/rcware/WinServiceRT/soft_plc.config`. The part containing parameters regarding Merbon DB communication looks as follows (the most important parameters are in bold letters):

```
        <!-- Whether or not to save data to RcWareDb. -->
<add key="SAVE_DATA_TO_RCWARE_DB" value="true"/>
```

True = data will be written into the database.

```
<!-- RcWareDb configuration mode (manual/automatic) -->
<add key="RCWARE_DB_AUTOMATIC_CONFIGURATION_MODE" value="false"/>
```

Keep the default settings – do not change.

```
        <!-- Host of the RcWareDb. -->
<add key="RCWARE_DB_HOST" value="klecany.domat.cz"/>
```

Name or IP address of the server where Merbon DB is running. (If a server name is entered rather than IP address, check the DNS settings in PLC network parameters.)

```
        <!-- Port of the RcWareDb. -->
<add key="RCWARE_DB_PORT" value="9876"/>
```

TCP port of the Merbon DB server. 9876 is a default value.

```
        <!-- Protocol of the RcWareDb. -->
<add key="RCWARE_DB_PROTOCOL" value="http"/>
```

Keep the default settings – do not change; another option is https, but the database would have to be configured accordingly.

```
        <!-- In case of automatic RcWareDb configuration host list is used -->
<add key="RCWARE_DB_URL_LIST" value=""/>
```

Keep the default settings – do not change.

```
        <!-- Save interval in seconds. -->
<add key="RCWARE_DB_SAVE_INTERVAL" value="300"/>
```

Interval of values sampling.

```
        <!-- Size of file buffer (in kB's) used when there is no connection to
RcWareDb. -->
        <add key="RCWARE_DB_FILE_BUFFER_SIZE" value="1000"/>
```

A PLC has about 10 MB free storage, so 1MB is OK (buffering allows to store about 20 000 samples per 1 MB, thus the total buffering time depends on the number of samples and sampling interval).

```
        <!-- SoftPlc Rt Id for RcWareDb data identification. -->
<add key="RCWARE_DB_SOFTPLC_RT_ID" value="FVE_DUBNO"/>
```

This string is one of the primary keys and must be unique for every PLC. See below.

```
<!-- Variables that will be saved to RcWareDb. Available choice: All, OPC,
Used -->
<add key="RCWARE_DB_VARIABLES_TO_SAVE" value="OPC"/>
```

- All = all variables in the runtime – do not use;
- Used = „nearly“ all variables (e.g. unused block inputs are not sampled) – do not use;
- OPC = this is OK, but remember to tag all relevant variables in SoftPLC IDE as OPC!

```
<!-- Access user name for connecting to RcWareDb. -->
<add key="RCWARE_DB_USER_NAME" value="rc_user"/>
```

User name with data write access defined in the database.

```
<!-- Access password for connecting to RcWareDb. -->
<add key="RCWARE_DB_PASSWORD" value="rc_user"/>
```

Password of the above user.

```
<!-- Values count in one request.       -->
<add key="RCWARE_DB_REQUEST_VARIABLES_COUNT" value="100"/>
```

Number of variables depends on the throughput of the network. Decrease this value if connection is slow.

```
<!-- Buffer files path. -->
<add key="RCWARE_DB_BUFFER_FILES_PATH" value="/tmp"/>
```

Location of the buffer files on the PLC file system.

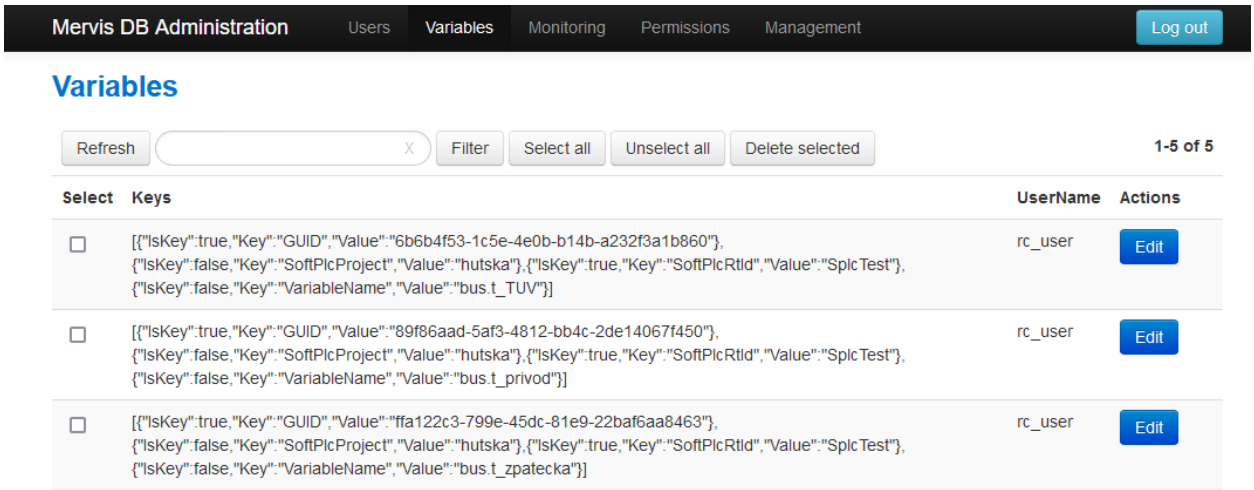
```
<!-- RcWareDb request timeout in ms. -->
<add key="RCWARE_DB_REQUEST_TIMEOUT" value="120000"/>
```

How long the PLC shall wait for the reply from the Merbon DB; after the timeout expires, the connection is considered as lost, and the data are buffered locally.

```
<!-- RcWareDb use RTC time of RT instead of value timestamp.-->
<add key="RCWARE_DB_USE_RTC_FOR_VALUE_TIMESTAMP" value="false"/>
```

If a value is not changed (e.g. it is a constant) – which time stamp has to be used.

The variable records in the database then look as follows:

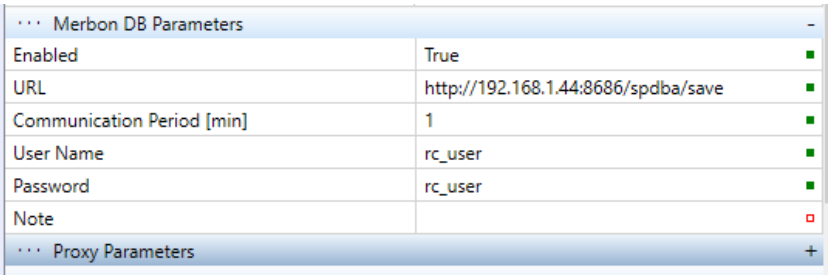


The variables have two primary keys: **GUID** (unique variable identifier within a runtime) and **SoftPLCRtId** (unique identification of runtime (PLC) within the database).

When a project is edited in SoftPLC IDE, the primary keys usually do not change, GUID is constant, and the variable exists for the whole lifetime of the project, even after its name or type has been changed. To generate a new GUID, a variable would have to be deleted and created again.

10 Setting of database writing in Merbon IDE

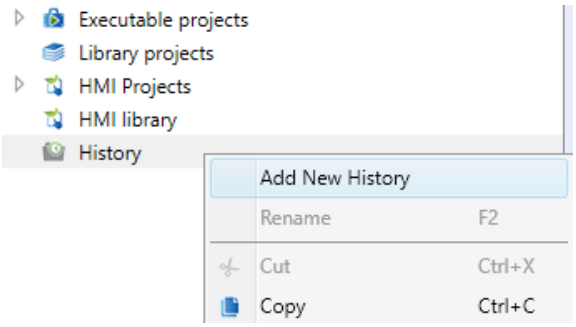
In Merbon IDE, the general parameters for writing into the database must be configured first:



Enabled: True = data will be written to the database
URL: Name or IP address of the server where the Merbon Database Adapter is running (mostly the same PC where Merbon DB is installed). The URL must be entered as complete, *http://... :8686/spdba/save*
Communication Period: How often the PLC writes data to the database. This is not the sampling interval! Data are temporarily stored in the PLC and sent to the database only time after time to keep the Merbon Database Adapter load not too high.
User name / Password: Credentials of database user with write rights (the user must be defined in the database).

Remember to set up in the PLC properties the **Runtime ID** – this is an unique string for the PLC that writes into the database. Usually, this is a combination of project name and PLC designation, such as "Uni_Vienna_Boilers“. **This is one of the primary keys**. The string length must not exceed 31 characters.

Next, specify what values shall be recorded, and the sampling period. This is what the History in the PLC is for. Remember that **to define History, the Solution must be switched to Full mode**.



In the History block properties, define the data sampling period. Variables to be sampled are then assigned to the block. There may be more blocks with different sampling periods in the PLC. The total history length, buffered in PLC, depends on the total number of sampled variables, and their sampling periods. The interval of communication with the database thus must be several times shorter than the length of buffered history so that no data are lost. See details in the Merbon IDE help, *History, saving data in a database*.

The setting is part of the PLC configuration. Finally, the configuration must be uploaded to the PLC.

When a name or type of variable is changed, the **CommUid** changes, and most probably the **GUID** as well (GUID is a primary key), which breaks the data row in the database. The data row reconstruction must be done by the client. Variables may be identified e.g. according to the **VariableName** key.