SAE – Automation, s.r.o., Nová Dubnica

Solid **A**nd **E**ffective partner for development of your products and industry automation.

Programming of OpcDbGateway and SAEAUT UNIVERSAL OPC Server

*OpcDbGateway is universal platform enabling integration of complex or easy applications preferably by configuring instead of more laborious programming. SAEAUT UNIVERSAL OPC Server is tool for easy development of dedicated OPC servers. Both products use **common configuring and programming practices** which are described in this document. There is also explained that it is easy to work with them for engineers using languages according to the IEC61131-3 as well as for programmers using languages as C++, C#.NET or Visual BASIC .NET.*

Introduction

The **IEC 61131-3** standard defines 5 programming languages and a software model for the programming of control systems. OpcDbGateway and SAEAUT UNIVERSAL OPC Server can be used to create control systems, and so there were an endeavour to enable usage of application creation practices alike to approach described in IEC1131-3. To implement probably all tasks, easy or complex, three of those 5 languages: **FBD** – Function Block Diagram, **ST** – structured text and **SFC** – sequential function control are always sufficient. Application creation practices used in OpcDbGateway and SAEAUT UNIVERSAL OPC Server resembles using of those three languages. There is not exact match with FBD, ST and SFC languages because the products are supposed to be used not only for creating of automation control systems but to integrate other types of applications as well. This resemblance enables easy adaptation on working with OpcDbGateway those who are used to work with above mentioned languages.

OpcDbGateway enables using either to configure or to program different tasks within function blocks. To program *Function blocks* content, to the **ST equivalent approach** using different higher level universal programming languages as C++, C# , Visual Basic .NET can be used. SFC can be implemented using triggers of the type value representing actual status together with associated function blocks providing to the actual status related actions and transition to a next status.

User can choose implementing of an application (1) by using **configuring** (for easy applications), (2) by **using of programming** or (3) by **combination of both methods** (for complex applications).

OpcDbGateway provides also the functionality of the **communication middleware** enabling access to data from different data sources as devices, databases, files, e-mails, SMS. It provides data exchange between different data sources using shared memory called ***Project Image Memory (PIM)*** containing **Memory Operands (MO)** which are able to contain different non-structured data types.[1]. Data exchange is provided by copying of data from MO associated with one data source to another data source.

[1] To exchange structured data types, an array of MO can be used

Running of programs in OpcDbGateway is **event-driven**. Events are triggered by triggers. Events have defined priority. There is **synchronous** and **asynchronous processing of tasks**. Synchronous Tasks are triggered by special implicit periodical trigger and processed **by way typical for programmable logical controllers (PLC)**. At the beginning of every synchronous cycle, data from different external data sources are (1) read implicitly (without need to program or configure this reading), (2) processed according to the configured or programmed functionality, (3) written implicitly to external data sources. Within every synchronous period **one implicit Function block – *Main*** which has the lowest priority is processed. Other function blocks within synchronous processing can be either (1) called[2] from the function block *Main* or (2) started as events by triggers and executed always with higher priority.[3]

Asynchronous processing lacks this implicit reading/writing from/to external data sources. Working with external data sources must be explicitly programmed. As this implicit reading/writing can be time consuming, asynchronous processing with **selective reading and writing from/to external data sources can be faster**.

Triggers are either of **the type time** (using cyclical or one shot timers) or of the **type value** (if a value of a MO changes from FALSE to TRUE an event is initiated). Asynchronous events are also put to the ordered priority front if cannot be processed immediately. Contrary to synchronous processing, the evaluation of priority front is not synchronised by the special implicit periodical trigger. [4]

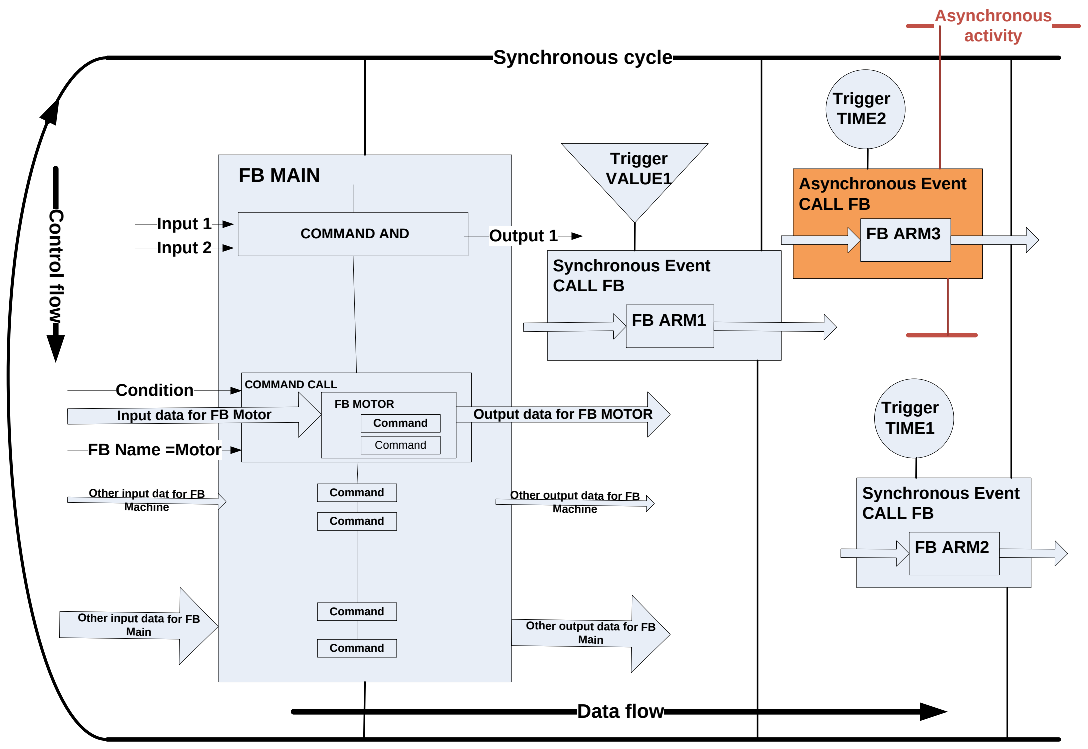


Figure 1 Cyclical processing of function block MAIN, synchronised processing of synchronous events and asynchronous processing of asynchronous events

[2] conditionally or unconditionally

[3] The synchronous events put to the priority front in previous synchronous are processed in next period. This is the reason why the Main function block has the lowest priority.

[4] In fact, as triggers can be cyclical, also this way, it is possible to provide periodic calling of tasks. But synchronous processing provides also some implicit functionality related to external data sources.

Within Figure 1, it is possible to see that function block *MAIN* is called cyclically. Control flow in the function block MAIN is provided by configurable commands. Every command can have maximally 2 inputs and maximally 1 output. Special command ***CALL*** enables calling of **nested function blocks**. (Functional block *MOTOR* in the Figure 1 is called from the function block *MAIN* using command *CALL*). Function blocks can be called also within events which are started by triggers. There can be events **synchronised within synchronous cycle** like event where *FB ARM1* is processed or **asynchronous** like event where *FB ARM3* is processed. One trigger can start one or more events. There are also special function blocks ***Start***, ***Restart*** and ***End*** (not shown in Figure 1) which are processed only one time at the starting or ending of OpcDbGateway runtime application and that are not started by triggers.

Sequential function control (SFC) with status automats can be implemented using triggers of the type value. Every status has associated one MO used as triggering variable. For every status, one Function block providing actions which have to be done in given step and second Function block providing evaluation of transition conditions to a next status is connected.

Configuring and programming of function blocks

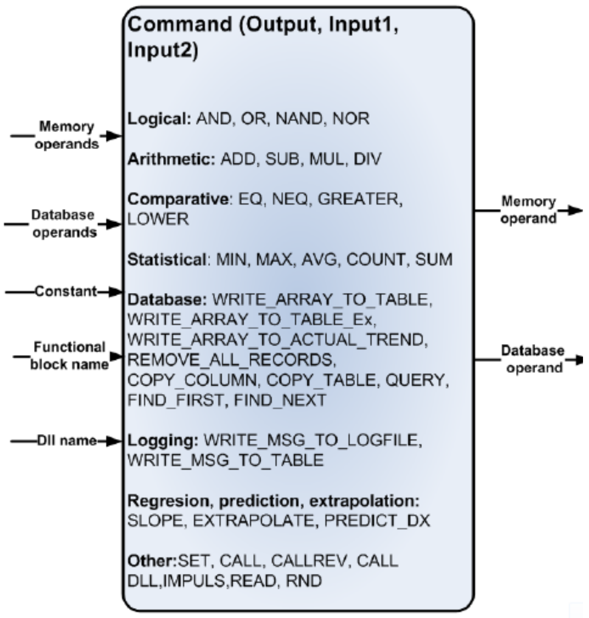

Command (Output, Input1, Input2)
Memory operands
Database operands
Constant
Functional block name
Dll name
Memory operand
Database operand
Logical: AND, OR, NAND, NOR
Arithmetic: ADD, SUB, MUL, DIV
Comparative: EQ, NEQ, GREATER, LOWER
Statistical: MIN, MAX, AVG, COUNT, SUM
Database: WRITE_ARRAY_TO_TABLE, WRITE_ARRAY_TO_TABLE_Ex, WRITE_ARRAY_TO_ACTUAL_TREND, REMOVE_ALL_RECORDS, COPY_COLUMN, COPY_TABLE, QUERY, FIND_FIRST, FIND_NEXT
Logging: WRITE_MSG_TO_LOGFILE, WRITE_MSG_TO_TABLE
Regresion, prediction, extrapolation: SLOPE, EXTRAPOLATE, PREDICT_DX
Other:SET, CALL, CALLREV, CALL DLL,IMPULS,READ, RND

Figure 2 Categorization of commands with used inputs and outputs

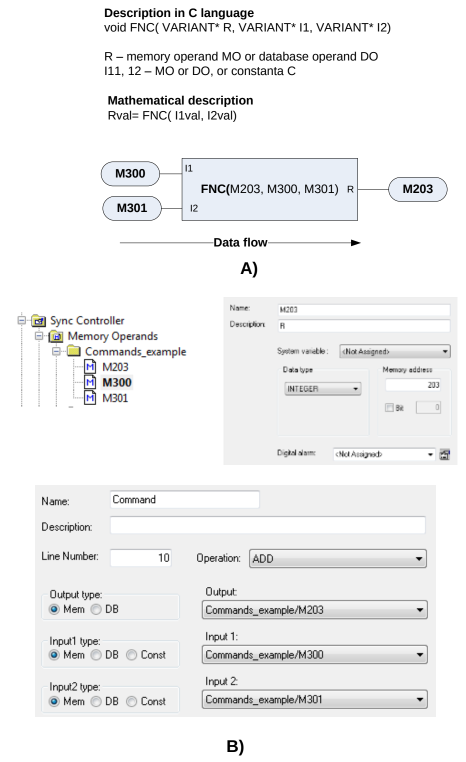
Function blocks can be either configured using built-in **configurable commands** (Figure 2) or programmed using different **programming languages**. The programmed function blocks can be added as part of configurable functionality for later usage. Engineers used to configure PLC's with languages described in IEC61131-3 can use approach alike as in languages FB, ST and SFC, on the other hand, programmers can reduce using of configured functionality on minimum and to program almost all functionality in languages as C++, C# and Visual Basic.NET... Both groups can take advantage of the fact that OpcDbGateway is in the same time a **middleware for easy access to different data sources**.

In this paragraph, both approaches configuring and programming of function blocks is described.

Function blocks can be created as **sequences of configurable commands**. Some commands provide easy functionality which can be described mathematically by arithmetic or logical functions as shown in the Figure 3 A). Some of them provide more complex functionality as for example writing of parameterised message to log file or to database. But, all those types have maximally two inputs and one output. Categorization of commands with possible inputs and outputs is in the Figure 2. Except of already mentioned MO, also database operands enabling direct working with cells, records and columns in database tables can be used as inputs/outputs. Some inputs as name of the enhancing DLL or a name of the function block are read directly from configuration database of the OpcDbGateway. Constants that have been put to the configuration can be used as inputs as well.

Using C-language, commands can be described as functions with the void return value (Figure 3A). Arguments of the function are over given **as references** on memory operands (MO). Result of command is saved also to a MO[5]. MO can be perceived as array of global variables. They are accessible from configurable functionality as well as from programmed functionality of the OpcDbGateway. Because of this, they can be used to provide **interface** between those two parts[6].

As shown in the Figure 3B), MO can be configured, it means, a symbolical name as well as a memory address - an **index in the array of memory operands[7]**. When configuring commands, output, and inputs can be chosen in the OpcDbGateway configurator according to their symbolical names. In the Figure 3B), memory operands with the


Description in C language
void FNC(VARIANT* R, VARIANT* I1, VARIANT* I2)

R – memory operand MO or database operand DO
I11, 12 – MO or DO, or constanta C

Mathematical description
Rval= FNC(I1val, I2val)

Name: M203
Description: R
System variable: <Not Assigned>
Data type: INTEGER
Memory address: 203
Bit: 0
Digital alarm: <Not Assigned>

Name: Command
Description:
Line Number: 10
Operation: ADD
Output type: [✓ Mem] DB
Output: Commands_example/M203
Input1 type: [✓ Mem] DB Const
Input 1: Commands_example/M300
Input2 type: [✓ Mem] DB Const
Input 2: Commands_example/M301

Figure 3 A) Command descriptions and using, B) Command configuring

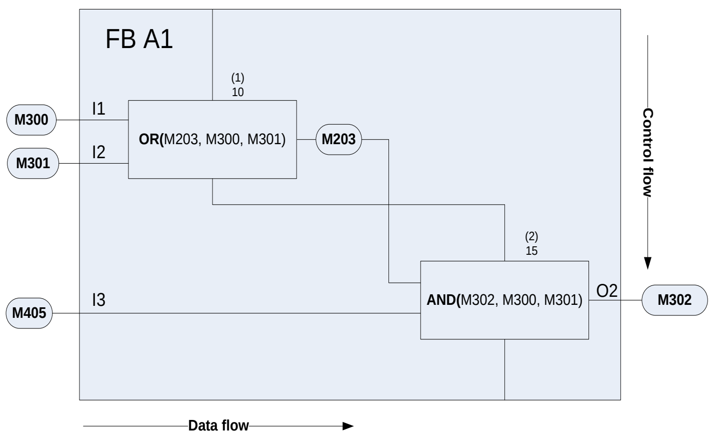


Figure 4 Function block A1 providing computation: O2 = (I1 OR I2)

[5] MO's itself are saved as variables of the type VARIANT which can carry many different data types. Using of just this data type was chosen to facilitate working with different data sources.
[6] MO with indexes 0-100 are reserved as interface to the internal functionality of the OpcDbGateway. They can be used for monitoring of this functionality within user programs or from outside.
[7] This index can be used to access data within a programmed functionality.

symbolical name *M300* and *M301* (configured in the directory *commands_example* are used as inputs for the command *ADD,* and MO with the name ***R*** as command result or output operand. There is configured also line number for a command. Function blocks can be configured as a sequence of commands. **Processing sequence** of commands within a function block is given just by **line numbers** which are ordered from lowest number to the highest one.

In the Figure 4, within *FB A1*, an evaluation of the logical expression *O2 = (I1 OR I2) AND I3* is executed. There is the command OR with the line number 10 processed as the first and then the command AND with the line number 15[8] as the second. As shown in the Figure 3 B), memory operands are **firmly associated** with inputs and outputs of the commands, and so also with inputs, outputs and intermediate data of a FB. The **function block instance** is in this case constituted by **steady set of commands and also memory operands**. To provide reusability of the FB for different data sets, it would be necessary to provide setting up data to memory operands associated with inputs (*M300, M301, M405*) before calling the *FB A2* and to overtake output from output MO *M302*. Except of this, it is necessary to evaluate risks related to parallel running of more instances of the same instance of the FB[9].

Alike way as in the Figure 4, it is possible and useful to describe the whole application when preparing its design**. Data flows are drawn always from left to the right** and **control flows down from top.** Commands are nested in function blocks and function blocks called from another function blocks using commands CALL all CALL REV can be either nested in those commands, as shown in the Figure 1, or drawn directly in the higher level function blocks to make drawing more synoptical. Function blocks processed within asynchronous or synchronous event *CallFunctionBlock* should be drawn as in the Figure 1. Drawing should be created hierarchically. In the drawing of higher hierarchical level, it is not necessary to draw boxes for all commands nested in function blocks. It is enough to draw only input and output data of the function blocks.

Except of function blocks configured using built-in commands, also function blocks programmed within enhancing DLL by user (Figure 5) can be included to the configuration. **They can become a part of configurable functionality**, and can be called from other function blocks using the command:

CALL DLL(Result, DLL Name, index of the 1st input operand).

The first input of that is the name of DLL saved in configuration file and the second the index of the MO in the MO array where sub array of inputs and outputs of the programmed function block begins.

[8] Gaps in line numbering can be used by editing of configuration to be able to put additional commands to the FB configuration easily

[9] As will be shown later, it can be solved using programmed FB.

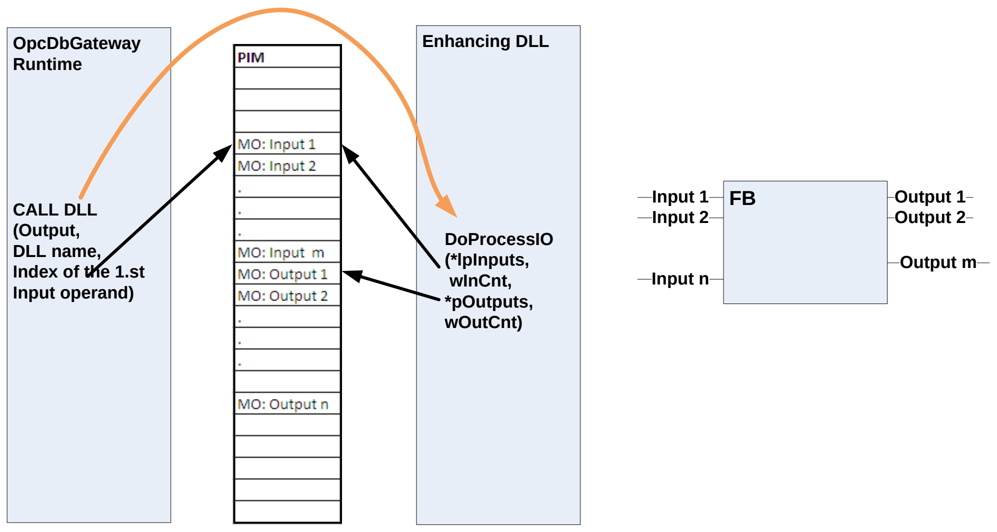


Figure 5 Using of Function block implemented within enhanced DLL

Alike as function blocks created using built-in commands, they can be drawn as rectangles with inputs and outputs. But, there is one important difference in terms of functionality. Contrary to the FB created using configurable commands and using global variables - MO's also for intermediate results, the programmed function blocks use **internal local variables for intermediate results**. Programmed FB's use MO's only for inputs and outputs of the FB.   Programmed function blocks have better reusability comparing to configured ones because **input and output data are not configured firmly**. This set can be changed easily by changing of Input 2 by the calling command *CALL DLL* to point on another set of input and output variables. Because of this, we can speak in this case about ***function block types***, instead of *function blocks.*

Integration of applications by programming

In the previous paragraph, it was explained how to use enhancing DLL's to implement **custom function blocks** and to use them in **configured application**. Within this paragraph, it will be shown how to use enhancing DLL to **integrate an application almost completely by programming**. Why almost completely? OpcDbGateway is used in integrated applications as middleware to interface different data sources. Data from those data sources must be mapped to MO's to use them in integrated application. These MO's and the mapping of external data to them is configured. All other functionality of the integrated application can be programmed within enhancing DLL.

This functionality can be provided within **individual thread** started together with other parts of the OpcDbGateway. Communication between this thread and other parts of the OpcDbGateway can be **synchronized over shared MO's** as shown in the Figure 6. Instead of calling the function *DoProcessIO()* from OpcDbGateway runtime only when a related function block implemented in enhancing DLL have to be used, the functionality implemented in enhancing DLL is still alive.

This functional mode is very useful e.g. to implement a communication driver for a device notably if it can be supposed that device will be able to send notifications captured by the communication driver.

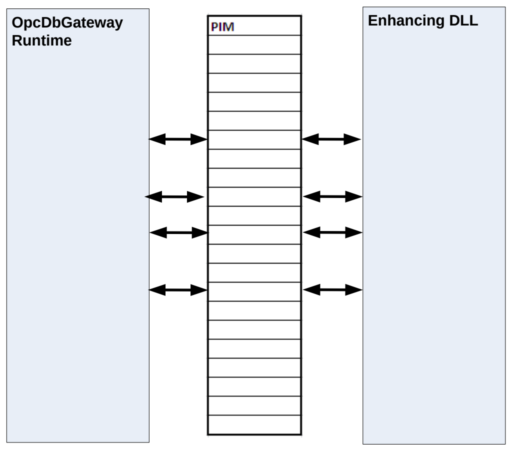


Figure 6 Cooperating of OpcDbGateway with enhancing DLL trough shared memory operands (without configured command CAL DLL)

It is also optimal to implement dedicated OPC server for the device using the SAEAUT UNIVERSAL OPC Server. The device driver running in enhancing DLL put items from the device to the MO's that are mapped to the address space of SAEAUT UNIVERSAL OPC Server.

Another useful application can be **implementing of status machine** (which can be described using SFC). The status machine can use MO's as status flags. These flags can be used as triggering variables for events calling executive or transition function blocks.

Using interfacing MO's, the enhancing DLL can use also other rich functionality of the OpcDbGateway as e.g. alarming and logging system, writing data to databases, sending SMS, e-mails, connection to SOA systems… This way, it is possible to create applications by **optimal combination of configured and programmed functionality**.

Conclusions

OpcDbGateway enables easily implement control and monitoring systems using **approach alike as described in IEC61131-3**, and so it is well usable **by automation engineers used to work with PLC's**.

On the other hand, it can be used by software engineers and **programmers used to work with high level universal programming languages and object oriented design** to create e.g. business process management applications.

Because of used communication technologies and standards as web services, OPC, database drivers, e-mailing, SMS messaging it enables to transfer, process and aggregate so **real time data** from plant floor as **aggregated data** on higher levels of enterprise hierarchy.

The unlimited possibility to enhance it with new communication standards and modules enables to keep it to be still the state of the art product.

© 2012 SAE - Automation, s.r.o.
All rights reserved.
Distribution and/or reproduction of this document or parts thereof in any form is permitted solely with the written permission of the SAE - Automation company. The technical data contained herein have been provided solely for informational purposes and are not legally binding. Subject to change, technical or otherwise. www.saeautom.sk, sae-automation@saeautom.sk,
tel.:+421-(0)42-445 07 01, fax:+421-(0)42-445 07 02,
Adresa: Trenčianska 19, 018 51 Nová Dubnica