OpcDbGateway – Hello World

[1]*OpcDbGateway is a software application platform consisting of a configurator and a runtime application. The configurator is used to create, debug and test created applications.*

As in other development environments, we can try the first steps on the "Hello World" application. Here's an analogy to creating and debugging a C ++ application:

```
#include <iostream>

int main()
{
	auto HW = "Hello World!\n";
	std::cout << HW;
}
```

Let's create a HV string variable containing "Hello World! \ N". Use the std :: cout << HV command to send the contents of this variable to standard output. We translate the program and run the console application.

For the creation and debugging of such a program we can use e.g. Visual Studio 2019 development environment.

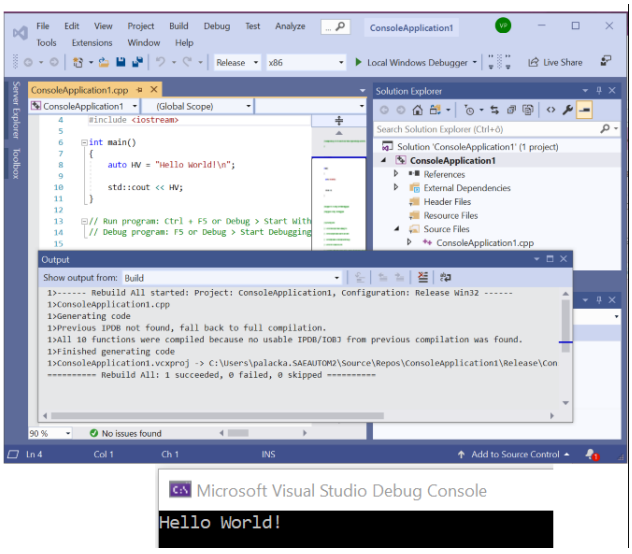


Figure 1 Creating, compiling and running "Hello World" in Visual Studio 2019

[1] The functionality described in this article is the same as in the SAEAUT UNIVERSAL OPC Server, except for the database and internal OPC client functionality.

Now let's show you how to create and test a similar application in the OpcDbGateway configurator.

Running a previous application resulted in a constant string display in the console application. However, **the runtime OpcDbGateway application** is not a console application, **but a server application without a user interface that provides data to client applications or stores them in databases and files.**

In Visual Studio, we have a string in the Debug Console. As a replacement for such a console, we can use **the OpcDbGateway configuration application**, which also functions as **a client application that communicates via the OPC DA interface with the OPC DA server of the runtime application.** OPC DA server runtime application provides access to so-called. process variables (PVs), which can contain variables of different types, including strings.

In typical OpcDbGateway applications, data from external devices and applications communicating with the runtime OpcDbGateway application are stored in the PVs. The PVs are stored in an array of structures in the runtime application's memory (which include, in addition to value, also type, timestamp, and quality) and made available through indexes of this array (Figure 2).

When creating a configuration, we need to provide an overview of the use of individual PVs in individual parts of the application. For this purpose, named memory operands (MOs) are used that can be created within **tree directory structures in the configurator and mapped to individual PVs** in the memory of a runtime application. We use them to create an **application data model**.

For each MO, it is also possible to create an OPC variable in the address structure of the address space of the internal OPC server with a single click. OPC variables created in this way provide a view of individual PVs for OPC client applications, such as the OpcDbGateway configurator itself or e.g. SAEAUT SCADA web client.

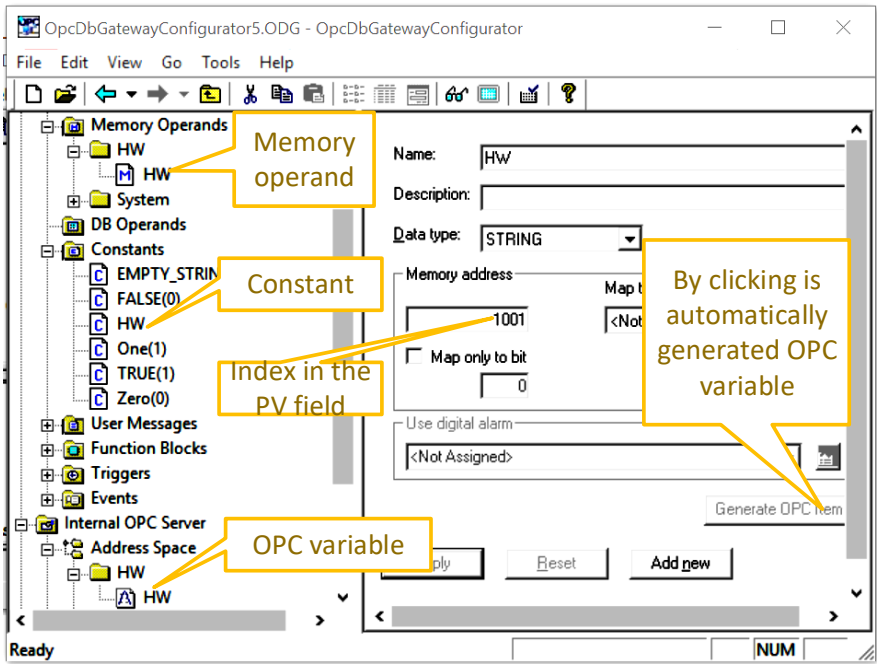


Figure 2 Location of memory operands, constants and OPC variables of the internal OPC server in the configuration

In OpcDbGateway, we implement the application by configuring a string constant that contains the word "Hello World". We copy this constant to a process variable using a configurable SET statement. This PV will be mapped to the OPC variable, the value of which will be displayed in the OPC client of the configurator (Figure 3).

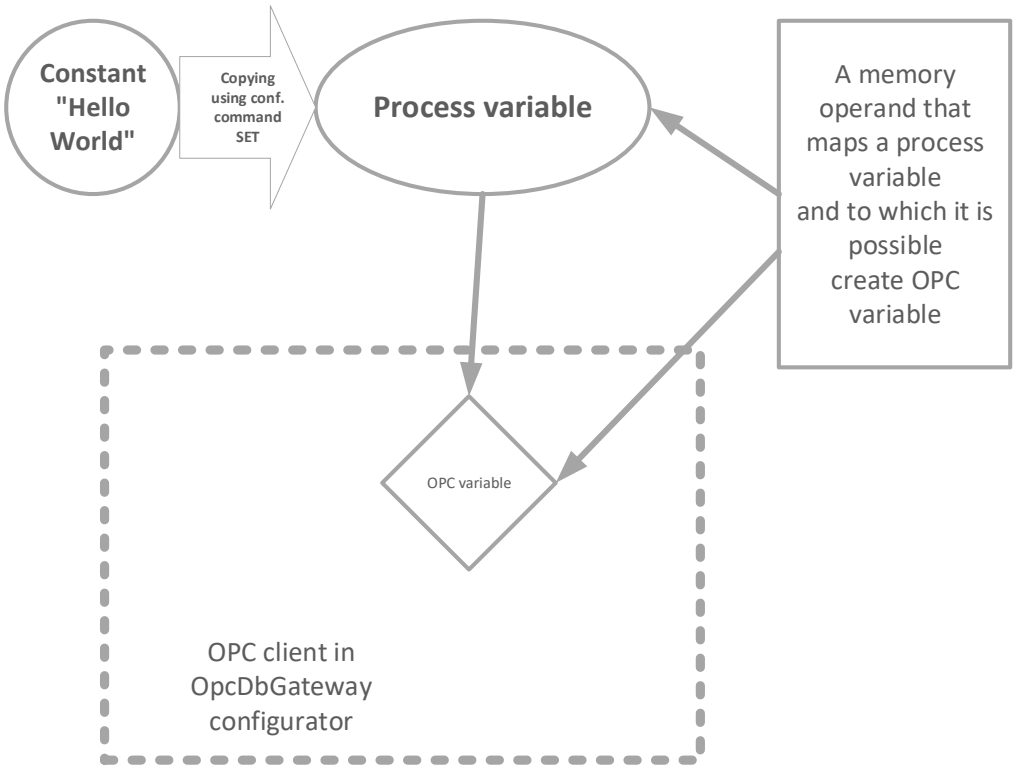
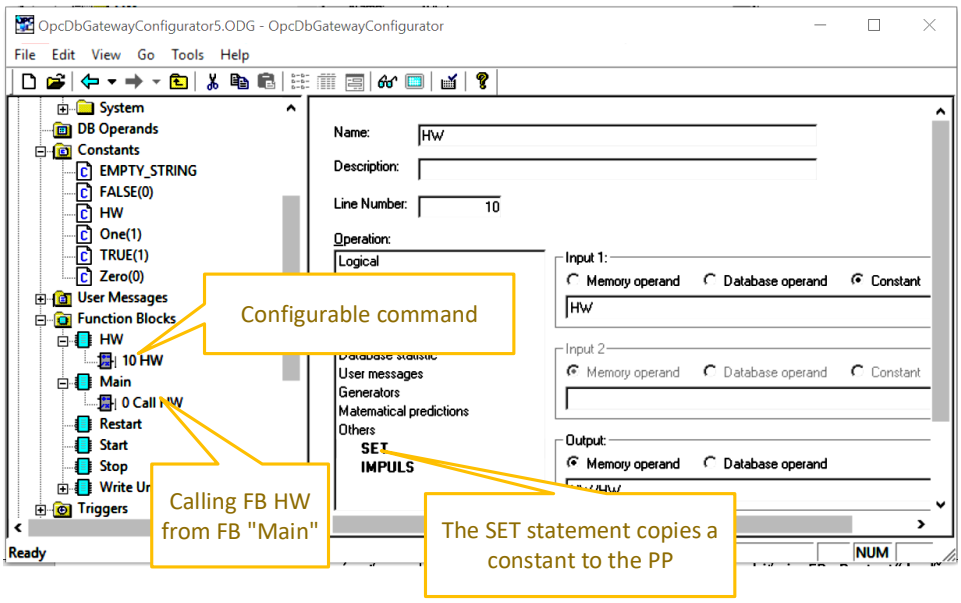


Figure 3 Implementation of "Hello World" in OpcDbGateway

OpcDbGateway allows you **to structure application functionality using function blocks (FBs)** that can be nested up to 15 levels.

FBs (Figure 4) contain configurable commands that perform various operations on memory operands, **database operands**, **constants**, and **user messages**. These operations can be very **simple** e.g. addition of the value of two PVs, or the already mentioned SET statement, **more complicated** e.g. executing a **configured SQL command over a process database**, or **calling a user program module** (eg, a PID controller) implemented in a DLL.

Cyclic (synchronous) data processing (with a configurable period) is performed in a single FB called "**Main**". All FBs embedded therein are also cyclically executed (Figure 4)



4).

Figure 4 The HW function block contains a SET command that copies the HW constant to the PV. FB HW is nested in FB "Main" and is therefore cyclically called by the CALL command

If you place a configurable SET command in FB “Main”, it will also be executed repeatedly. Since in our case we only need to copy the constant string once, such functionality is redundant.

However, OpcDbGateway also has several options for a one-time FB call. When the runtime application is started, the FB “**Start**” or “**Restart**” is executed once. So, if you place the SET command in FB “Start”, it is executed only once.

FB “Start” is executed at the beginning only if a new configuration has been created or an old one changed. FB “Restart” is performed when the application is restarted. Therefore, the question arises - is it necessary to place the SET command in the FB "Restart"?

OpcDbGateway ensures **simple persistence of data** in the sense that upon the termination of the application, the status of all PVs is stored in a database, from which it is automatically restored on restart. Since we used the SET command to copy a constant string to the PP already in the FB “Start”, we do not need to do this in the FB “Restart” because at the restart the state of the PV was automatically restored.

For the sake of completeness, it should be noted that there is another FB that is executed by default called "Stop", which is automatically called when the application is closed. What happens if we put the SET command in FB "Stop"? When you start the application for the first time, you will see an empty string in the OPC client. When restarting such an application, “Hello World” will already appear, because at the previous start the PP was already set in FB “Stop”.

However, we are not finished with one-time FBs. In fact, any FBs, except the “Main” FB, can be executed once. **FBs can be initiated as "events".** An event is a functionality triggered by some “**trigger**”. There are several types of events (except the mentioned FB calling) in OpcDbGateway – e.g. start an external application. Now, however, we will only deal with the FB start event.

The event can be activated as one of three types: **event of the type time** – i.e. at a defined time, of the **type value** - when the selected PV has one of the selected values "true" or "false". There is also the possibility to start the **event when both conditions - defined time and defined PV value - are met**.

Events also have **a defined priority**. If two different events are to be performed at the same time, the order of their execution will be given by their priority.

Although **periodic FB execution** can be ensured in FB “Main”, a **periodically triggered event** can also provide periodic FB calling. So, what is the difference between periodic functionality in "Main" and that implemented through events? The cycle performed through "Main" starts by **implicitly loading inputs from external applications and devices**, this is such that are not mediated by configurable commands, **continues to process them using the configurable commands in FB "Main"** and ends with **implicit copying from PVs to outputs for external applications**. Such a cycle is typically implemented in industrial control systems (PLCs) that simulate quasi-parallel evaluation of logical expressions. In connection with OpcDbGateway we refer to it as **synchronous cycle**.

Functionality in FB "Main" can be combined with so-called **synchronous events**, which are characterized by the fact that the **"triggering" conditions** for events are always evaluated at **the beginning of the synchronous processing period**. On the other hand, the "triggering" conditions for **asynchronous events** are evaluated, and their corresponding events are executed (in order of priority) **almost immediately as they occur**. Asynchronous events also provide either FB execution, external process/application calls or some special operations such as create a new log file. **The synchronous and asynchronous functionality in OpcDbGateway is divided into two program threads.** This ensures that the performance of asynchronous events, even if time-consuming, does not interfere with the synchronization processing period. Regarding the periodic triggers, it is useful to mention that **it is possible to define the number of repetitions**. **Execution time** can be defined as **absolute or relative to the start of the application**.

Let's review how we create and monitor the implementation of the “Hello World” application in the OpcDbGateway configurator:

1. Create a configuration for the new application using the command in the menu File-> New. The first **automatic configuration check** is performed. (Figure 5).
2. Define a string constant named HW containing string "Hello World" (Figure 2).
3. Create a MO in a new directory called HW. MO is mapped to OPC variable with the same name and in the directory of the internal OPC server address space (Figure 2)
4. Define the FB with the name HW, which will contain the only configurable command “SET”, by means of which the HW constant content is copied to the PV, which is referenced by the MO with the name HW located in the directory HW (Figure 4).
5. Run the configuration check (Figure 5).
6. Run the OPC client in the configurator (glasses icon) and the message "Hello World" will appear in the monitoring window (Figure 6).

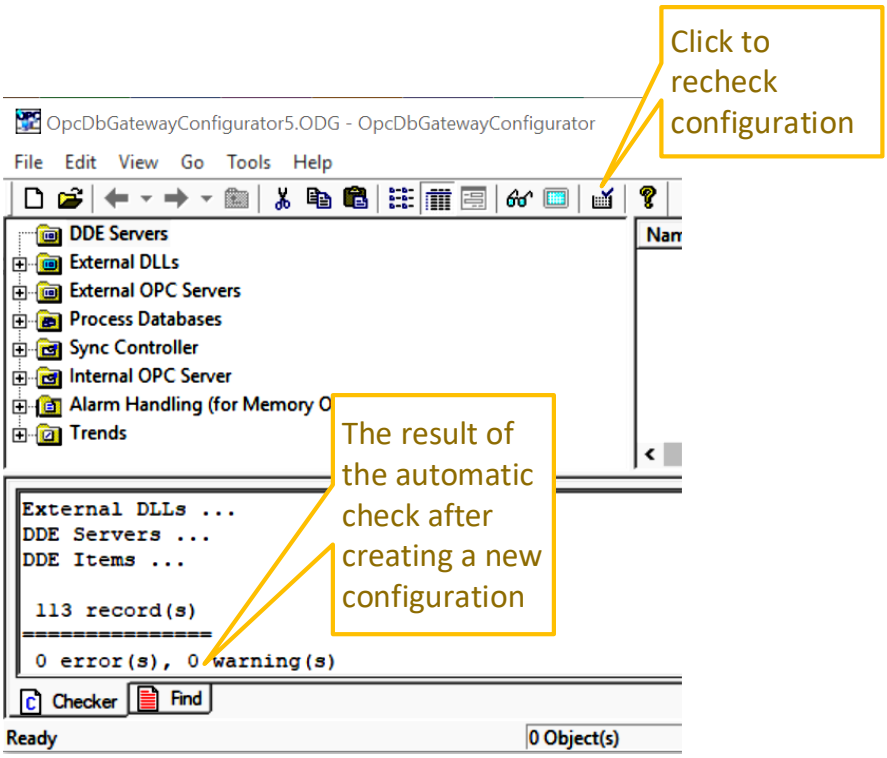


Figure 5 Check configuration after opening and editing

In OpcDbGateway, we can monitor and influence the functionality of the configured application not only through PVs and OPC variables that are application-specific (Figure 6), but also through **system PVs that are part of an automatically created configuration template**. Part of these variables (in the "Status" directory - Figure 7) is read-only and a part enable also allows writing. E.g. the *System.Status.AsyncQueSize* variable tells you how many activated asynchronous events have not yet been executed. By means of variables in the Control / Log directory named *TimeLog* and *TraceLog*, it is possible, for example, to change a how much application runtime information is written to the log file at runtime (Figure 8, Figure 9**). You can change the value of a control variable by right-clicking on the variable in the monitoring window**.

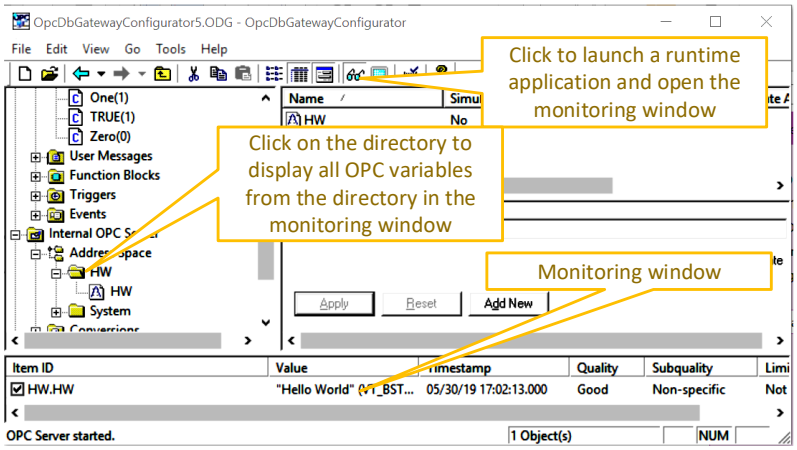


Figure 6 Monitoring of PV content via OPC variable to which it is mapped

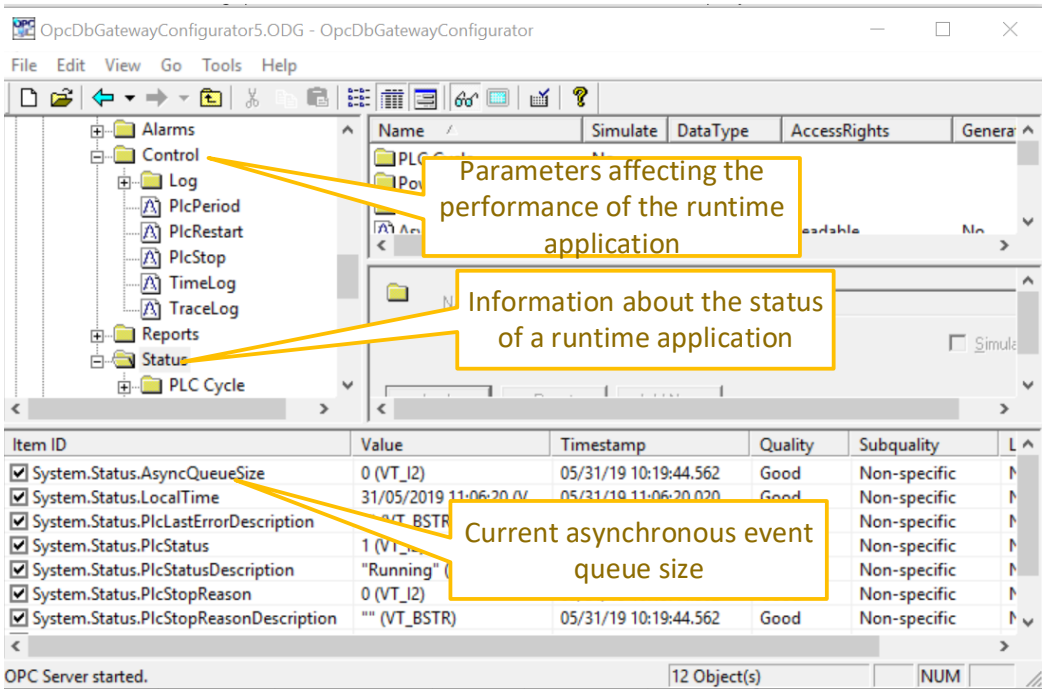


Figure 7 Monitoring and control the runtime 0pcDbGateway application. The variables in the "Control" directory serve for control and in the "Status" directory for monitoring.

The log file display is continuously updated. You can watch it in the configurator by selecting View-> Output view-> Log Viewer in the main menu.

User logs can also be written to the log file via the configurable *WRITE_MSG_TO_LOGFILE* command. In this way, we could expand our application. Simply define the user message and write it to the log file using the configurable *WRITE_MSG_TO_LOGFILE* command (Figure 10). **User messages need not only contain the text, but also the MO parameters, which ensure that the current values corresponding to the PVs are written to the log file.**

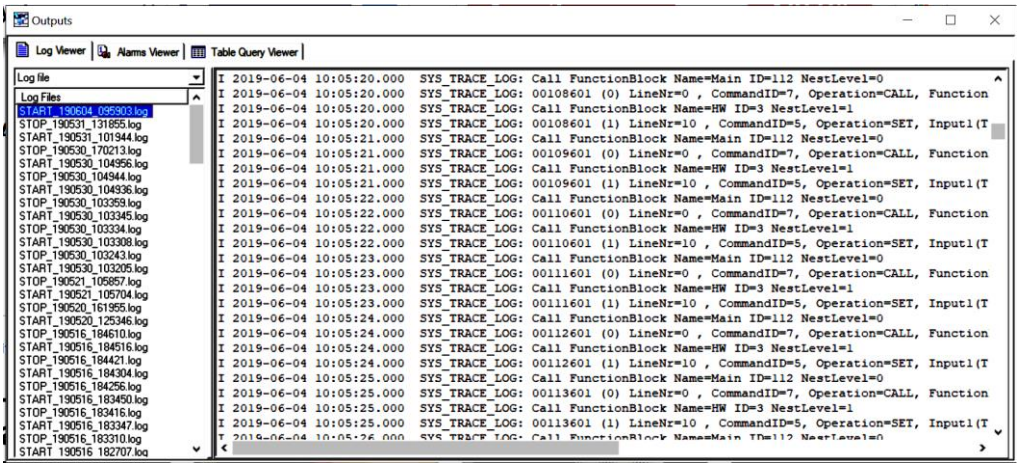


Figure 8 Log file when TraceLog system variable is set to true. Data on the execution of individual function blocks, configurable commands, including their arguments, is recorded

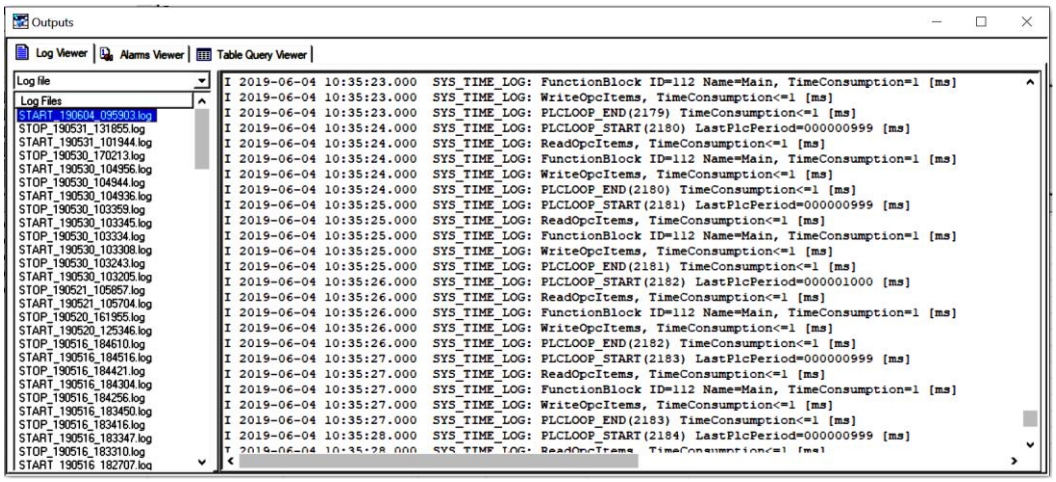


Figure 9 Log file with time data recording for individual activities (system variable TimeLog = true)

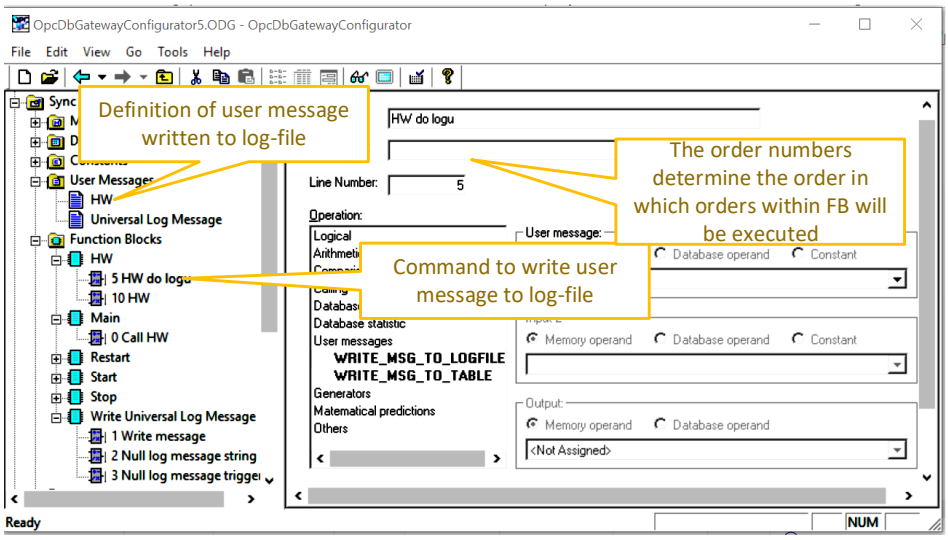


Figure 10 Configuration extension by writing user message to log-file. The order of execution of configurable commands within FB is given by their numbering.

Within the FB HW in Figure 10, we now have two configurable commands - “HW to Log” with number 5 and “HW” with number 10. **These numbers ensure that the execution of configurable commands is executed in ascending order**. Spacing numbering was already used with the good old BASIC language to make it easy to insert new commands without renumbering by program enhancing.

From a comprehensive description of such a simple example, it may seem that configuring applications in OpcDbGateway is difficult. However, the difficulty lies in the fact that we have tried to describe at least part of the basic functionality that OpcDbGateway offers.

Using **configurator auxiliary functions** such as I/O mapping from various external devices and databases, also **applications with huge amounts of I/O can be configured relatively quickly**. In addition, OpcDbGateway includes several functionalities that would otherwise require separate applications.