SIEMENS

SIMATIC

S7 Software Controller IOT2000EDU

Operating Manual

03/2018

A5E42342500-AB

Siemens AG
Division Digital Factory
Postfach 48 48
90026 NÜRNBERG
GERMANY

A5E42342500-AB
Ⓟ 04/2018 Subject to change

Table of contents

1 Guide through this Operating Manual

Purpose of the documentation

The information in this operating manual will enable you to install and use the "SIMATIC S7 Software Controller IOT2000EDU" software.

You will also learn how to work with TIA Portal projects to program, start and stop the IOT2000EDU.

Readership

This documentation is intended for students in educational institutions. It is used for training purposes and provides information on handling TIA products.

Required basic knowledge

The following knowledge is required to understand the documentation:

- General knowledge of automation engineering
- An understanding of the SIMATIC industrial automation system
- Experience in programming with C/C++
- Experience in working with MATLAB/Simulink and Simulink Coder
- Experience in working with Eclipse
- Proficiency with Microsoft and Linux operating systems
- Experience in working with computer technology

Validity of the documentation

This document is valid for the following products:

- IOT2000EDU: **6ES7671-0LE00-0YB0**
- IOT2020: **6ES7647-0AA00-0YA2**
- IOT2040: **6ES7647-0AA00-1YA2**
- SIMATIC IOT2000, Input/Output Module: **6ES7647-0KA01-0AA2**

Notes

Please also observe the notes marked as follows:

Note

A note contains important information on the product described in the documentation, on the handling of the product or on the section of the documentation to which particular attention should be paid.

Definitions and naming conventions

The following generic terms are used in this documentation:

- **Arduino Shield:** ARDUINO UNO (Rev3)
- **SIEMENS Shield:** SIMATIC IOT2000, Input/Output Module
- **IOT2000EDU:** SIMATIC S7 Software Controller IOT2000EDU
- **IOT20x0**: SIMATIC IOT2020 and SIMATIC IOT2040
- **STEP 7**: We refer to the configuration and programming software as "STEP 7" in this documentation synonymously for the version "STEP 7 V15 (TIA Portal)".
- **SO:** Shared Object

2 Safety information

2.1 Security information

Siemens provides products and solutions with industrial security functions that support the secure operation of plants, systems, machines and networks.

In order to protect plants, systems, machines and networks against cyber threats, it is necessary to implement – and continuously maintain – a holistic, state-of-the-art industrial security concept. Siemens' products and solutions constitute one element of such a concept.

Customers are responsible for preventing unauthorized access to their plants, systems, machines and networks. Such systems, machines and components should only be connected to an enterprise network or the internet if and to the extent such a connection is necessary and only when appropriate security measures (e.g. firewalls and/or network segmentation) are in place.

For additional information on industrial security measures that may be implemented, please visit (https://www.siemens.com/industrialsecurity).

Siemens' products and solutions undergo continuous development to make them more secure. Siemens strongly recommends that product updates are applied as soon as they are available and that the latest product versions are used. Use of product versions that are no longer supported, and failure to apply the latest updates may increase customers' exposure to cyber threats.

To stay informed about product updates, subscribe to the Siemens Industrial Security RSS Feed under (https://www.siemens.com/industrialsecurity).

2.2 Application of the product

Note

Do not use this product in productive operation or for controlling potentially hazardous processes. This product is only intended for students and is therefore reserved for educational purposes.

Description 3

3.1 Overview of functions IOT2000EDU

IOT2000EDU implements a SIMATIC S7-compatible software controller on the IOT2020 or IOT2040 and is designed for training purposes. It is specially designed for learning how to program controllers with the SIMATIC TIA Portal.

In addition, you can use the TIA Portal program editor to program the IOT2000EDU. You can do so with the help of organization blocks, function blocks and data blocks as well as functions. You can use these blocks and the TIA Portal libraries to write your programs.

The TIA Portal or the command line program "CPU_Control" enables you to control the IOT2000EDU. You can also configure your IOT20x0 device, program it and execute the functions RUN, STOP and MRES.

Note

Application of the product

Do not use this product in productive operation or for controlling potentially hazardous processes. This product is only intended for students and is therefore reserved for educational purposes.

3.2 Required components

Make sure that the product is complete. The scope of delivery of the IOT2000EDU includes the following parts:

- IOT2000EDU IPK file for installing the Runtime on the SIMATIC IOT2020 or IOT2040
- IOT2000EDU HSP file for providing runtime in SIMATIC STEP7 V15
- IOT2000EDU manual as PDF file (German and English)
- IOT2000EDU license label for attaching to SIMATIC IOT2020 or IOT2040

3.3 Operating conditions

Deployment requirements

The IOT2000EDU is solely intended for educational purposes. This means it must not be used in production plants or safety critical areas.

Hardware requirements

- SIMATIC IOT2020 or IOT2040
- Power supply 24 V DC
- Recommended: Arduino Shield or SIEMENS Shield

Note

Using the IOT2000EDU without Shield

If you do not configure a shield, the IOT2000EDU runs without access to the I/O interfaces over the Arduino pins.
Plus you will see the following message in "var/log/messages": "Can not open IO configuration file, IO Module not initialized."

- microSD card
- LAN cable
- PC with host OS

Software requirements

- SIMATIC IOT2000EDU
- TIA Portal V15 or higher
- Hardware Support Package "HSP_V15_0245_001_S7_IOT2000EDU_1.1.isp15"
- Yocto Linux OS
- OPKG Package Manager Version 0.3.3 and higher
- MRAA 1.6.1
- Libc6 and libstdc++6
- SSH or installed serial terminal
- 200 MB free memory on the root file system
- Root privileges

4 Preparing the product for first use

4.1 Selecting the hardware

There are two SIMATIC IOT20x0 versions:

- SIMATIC IOT2020
- SIMATIC IOT2040

SIMATIC IOT2020

- Intel Quark® x1000
- 512 MB RAM
- 1 Ethernet port
- 1 USB Host Type A
- 1 USB Client microUSB

SIMATIC IOT2040

- Intel Quark® x1020
- 1 GB RAM
- 2 Ethernet ports
- 1 USB Host Type A
- 1 USB Client microUSB
- 2 RS232/485 interfaces
- Battery-buffered RTC

4.2 Installing Yocto Linux

4.2.1 Requirements and settings

The SIMATIC IOT2000EDU uses a Yocto Linux operating system which must be installed on a microSD card. This means you need a microSD card with a memory capacity of 8 GB to 32 GB.

This documentation assumes that you have already commissioned the IOT. You can find additional information on commissioning here (https://support.industry.siemens.com/tf/ww/en/posts/setting-up-the-simatic-iot2000/155642/?page=0&pageSize=10).

4.2.2 Installation steps

Download SD cards example image or create it yourself

To use the full scope of functions offered by the IOT2000EDU,you need an SD card with an installed Yocto Linux operating system. You can find the operating system in the Siemens Industry Online Support (SIOS Portal). You can download it here (https://support.industry.siemens.com/cs/document/109741799/simatic-iot2000-sd-card-example-image?dti=0&lc=en-WW).

But you can also create your own image. You can find the instructions on the Internet (https://github.com/siemens/meta-iot2000).

If you observe these instructions and do not want to use the example image from the SIOS Portal, you must install the following software prior to building your own image:

- OPKG Package Manager Version 0.3.3 and higher
- MRAA 1.6.1
- Libc6 and libstdc++6

Installing the OPKG Package Manager

You install the OPKG Package Manager with the help of the Yocto Build System by adding the image feature "package-management" to the "EXTRA_IMAGE_FEATURES" tag and setting `PACKAGE_CLASSES ?= "package_ipk"`.
Detailed information on Yocto tags is available on the Internet here (http://www.yoctoproject.org/docs/1.8/ref-manual/ref-manual.html#ref-features-image) and here (http://www.yoctoproject.org/docs/1.8/ref-manual/ref-manual.html#var-PACKAGE_CLASSES).

Installing MRAA 1.6.1

The IOT2000EDU requires the Intel MRAA Library (1.6.1) for Arduino based IO tasks. Additional information is available here (https://github.com/intel-iot-devkit/mraa#installing-on-intel-32bit-yocto-based-opkg-image).

Installing Libc6 and libstdc++6

You must also set the following tags before you can start the image building process:

```
IMAGE_INSTALL_append = " libstdc++"
```

To use additional packages, such as ssh-server-openssh, add the entry `meta-oe meta layer` to the "bblayer.conf" file.

The IOT2000EDU also needs libjson-c 0.12 which is included in the ipk package.

4.3 Installing IOT2000EDU

4.3.1 Requirements and settings

To install the IOT2000EDU without problems, you should meet the following requirements and prepare your Linux operating system according to the following recommendations.

- IOT2020 or IOT2040
- Installed Linux OS on a microSD card
  - Linux with TCP/IPv4 support
  - libc6 and libstdc++6
  - mraa 1.6.1
  - OPKG Package Manager Version 0.3.3 and higher
  - IKP Package ecosystem
  - 200 MB free memory on the root file system
  - Root privileges
- A host OS with:
  - SSH or installed serial terminal
  - Keyboard
  - Monitor
  - Serial FTDI cable or LAN cable

4.3.2 Installing IOT2000EDU

Requirement

- Root rights

Procedure

1. Check the opkg installation: `opkg -v`
2. Install IOT2000EDU_<version-info>_i586.ipk using the OPKG Package Manager with the command: `opkg install IOT2000EDU_<version-info>_i586.ipk`

Example for version 1.1.0: `opkg install IOT2000EDU_1.1.0_i586.ipk`

Result

You can find the executable files "IOT2000EDU" and "CPU_Control" under "/usr/local/bin".

The default IO configuration file "io.conf" is located in the directories "/usr/local/etc".

The "Libjson-c" files are located in the directories "/usr/lib" and "/usr/include/json-c".

4.3.3 Checking the installation

1. Check the installed packages with the following OPKG List command:
   - `opkg info IOT2000EDU`
     This command represents the package name that is called by the OPKG database.
2. The executable files and their version can be checked with the parameters `-v` or `--version`:
   - `/usr/local/bin/IOT2000EDU -v`
   - `/usr/local/bin/CPU_Control -v`

The OPKG Manager shows an error message and cancels the installation when Dependencies such as "mraa" are missing.

When you restart the installation of the package, the OPKG shows an error message and cancels the installation.

The Web server is included in the installation of the IOT2000EDU installation package "IOT2000EDU_1.1.0_i586.ipk". Find out more in the "Web server (Page 42)" section.

4.3.4 Operating IOT2000EDU

Starting the IOT2000EDU

You start the "IOT2000EDU" application in the installation path with a user role.

You can start the IOT2000EDU application with the suffix "&" so that it is being executed in the background. This process frees up the shell for the next user inputs.

Example:

```
/usr/local/bin/IOT2000EDU&
```

You can execute the application from the current path with the prefix `./` that identifies the current directory.

To execute the application from any path, you must enter the complete storage path instead of the current path prefix.

You can also use the init.d autostart service, which is pre-installed by the IOT2000 Yocto image. If you set the init.d service for the IOT2000, the IOT2000EDU starts automatically when the Linux OS is booted.

Logs

Logs are created in the file "/var/log/messages".

I/O configuration

During installation of the IOT2000EDU_<version>_i586.ipk package, the preconfigured example file io.config.sample is installed in the directory "/usr/local/etc". This example file contains the pin configuration settings "SIMATIC IOT IOT2000, Input/Output Module".

When the IOT2000EDU is started for the first time and the folder "IOT2000EDU" does not exist in the "$HOME" directory, the folder is created and the example file "$HOME/IOT2000EDU/io.conf.sample" is created as well.

To use the pins of the IOT20x0, either adapt the file "io.conf.sample" and change the name to "io.conf" or create a io.conf file manually.

If the IOT2000EDU cannot locate the file "$HOME/IOT2000EDU/io.conf" during startup, the IOT2000EDU continues to start and the Arduino pins are not initialized.

Alternatively, you can also forward the manually created io.conf file with the parameters `-c` or `--conf` to the IOT2000EDU.

Example:

- `/usr/local/bin/IOT2000EDU -c <file name with path of the manually created io.conf file>`
- `/usr/local/bin/IOT2000EDU --conf <file name with path of the manually created io.conf file>`

When you set the status of the CPU to "STOP" or "SHUTDOWN", for example with "CPU_Control" or the TIA Portal, the PWMs stop and the GPIO outputs are set to "0". At the start of the "run" state, all pins are reset to the default settings that you entered in the io.conf file.

waf files

The IOT2000EDU checks the directory "$HOME/IOT2000EDU" for instance-specific files when started. The IOT2000EDU saves the control program downloaded from the TIA-Portal and the hardware configuration in the waf file. The waf file is downloaded and executed during booting when the PLC is in "RUN". If the IOT2000EDU cannot locate these files, it generates project-specific waf files in the directory "$HOME/IOT2000EDU".

In doing so, the IOT2000EDU uses the default directory "$HOME/IOT2000EDU". However, you can use the parameters `-p <project directory>` or `--projectdir <project directory>` to adjust the path of the project directory.

Example:

- `/usr/local/bin/IOT2000EDU -p <path to other project directory>`
- `/usr/local/bin/IOT2000EDU --projectdir <path to other project directory>`

Displaying version information

To display the version information, use the parameters `-v` or `--version`.

Example:

- `/usr/local/bin/IOT2000EDU -v`
- `/usr/local/bin/IOT2000EDU --version`

Changing the instance name of the Runtime

To change the instance name of the IOT2000EDU, use the parameters `-i` or `--instancename`.

Example:

- `/usr/local/bin/IOT2000EDU -i <new instance name>`
- `/usr/local/bin/IOT2000EDU --instancename <new instance name>`

Displaying help information

To display the help information of the IOT2000EDU, use the parameters `-h` or `--help`.

Example:

- `/usr/local/bin/IOT2000EDU -h`
- `/usr/local/bin/IOT2000EDU --help`

4.3.5 Installing IOT2000EDU support package in the TIA Portal

This section provides information on how to install the Hardware Support Package (HSP) of the IOT2000EDU in the TIA Portal. This is necessary so that you can work with the IOT2000EDU in the TIA Portal.

Requirements

- You have installed TIA Portal V15 (or higher).
- You have HSP "HSP_V15_0245_001_S7_IOT2000EDU_1.1.isp15".

Procedure

1. Start the TIA Portal as administrator.
2. In the TIA Portal, click "Support Packages" in the "Options" menu bar.
   The "Detailed information" dialog opens. A table lists all support packages from the directory that you selected as the storage location for support packages in the settings.
3. Click the "Add from file system" button.
4. Select the file "HSP_V15_0245_001_S7_IOT2000EDU_1.1.isp15".
5. Select the support package and click on "Install".
6. Close and then restart the TIA Portal.

Result

After successful installation, you can now access the IOT2000EDU in the TIA Portal device catalog along with all other functions, such as Configuration, Download or Online.

Uninstalling IOT2000EDU 5

Proceed as follows to uninstall the IOT2000EDU:

1. Start the uninstall with the following OPKG command: `opkg remove IOT2000EDU`

This command deletes the following files:

- "io.conf.sample" unter "/usr/local/etc/"
- "IOT2000EDU" and "CPU_Control" under "/usr/local/bin/"
- libjson-c files under "/usr/lib/" and "/usr/include/json-c/"

The user-specific files in the directory "IOT2000EDU" are not deleted during uninstall.

When you try to uninstall the package multiple times, a message will inform you that a package is not available to uninstall.

6 Operating the TIA Portal

6.1 Introduction to the TIA Portal

The Totally Integrated Automation Portal (TIA portal) integrates various SIMATIC products in a software application with which you can increase your productivity and efficiency. The TIA products work together within the TIA portal and support you in all areas required for the creation of an automation solution.

The most important configuration steps are:

- Creating the project
- Configuring the hardware
- Networking the devices
- Programming the PLC
- Configuring the visualization
- Loading the configuration data
- Using the online and diagnostic functions

In the chapter below you will learn how to create, configure and program the IOT2000EDU in the TIA Portal.

6.2 Adding and configuring IOT2000EDU in the TIA Portal

In this chapter you will learn how to add the IOT2000EDU in the TIA Portal.

There are multiple ways in which you can add a device in the TIA Portal. This section describes the procedure using the "Add new device" dialog.

Requirement

- You have started the TIA Portal.
- You are in the portal view.
- You have created a new project.

Add new device

1. Click on the option "Configure a device".
2. Click on the option "Add new device".
   The "Add new device" dialog opens.
3. Click "PC systems".
4. Go to the entry under "PC systems > SIMATIC IOT2000EDU" > "6ES7 671-0LE00-0YB0" and select it.
   A preview of the IOT as well as its article number, version and description is shown on the right side of the dialog.
5. Click "Add".

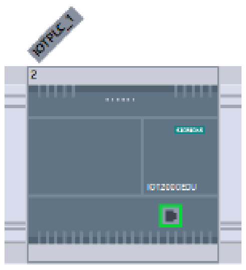
Result

The project view opens and the IOT2000EDU has been added to your project.

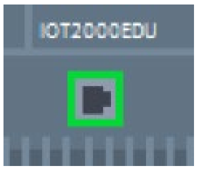
Configuring the IOT2000EDU

Once you have created the IOT2000EDU as a new device, you can view and configure the IOT2000EDU and its interfaces and change the name.

- In the working area, click the device "IOTPLC_1".
  The "Properties" tab of the IOT2000EDU opens in the inspector window.



- In the working area, click on the interface with the green border.
  The "Properties" tab of the IOT interface opens in the inspector window.



You can enter the IP address and the subnet mask address in the "Properties" tab.

Note

The IP address set in the interface properties must be the same as the address of the IOT20x0.

6.3 IOT2000EDU programming with the TIA Portal

In this section you will learn about the means that are available to you when programming the IOT2000EDU in the TIA Portal.
You will learn how you can insert the organization blocks, function and data blocks and the functions.

Organization blocks

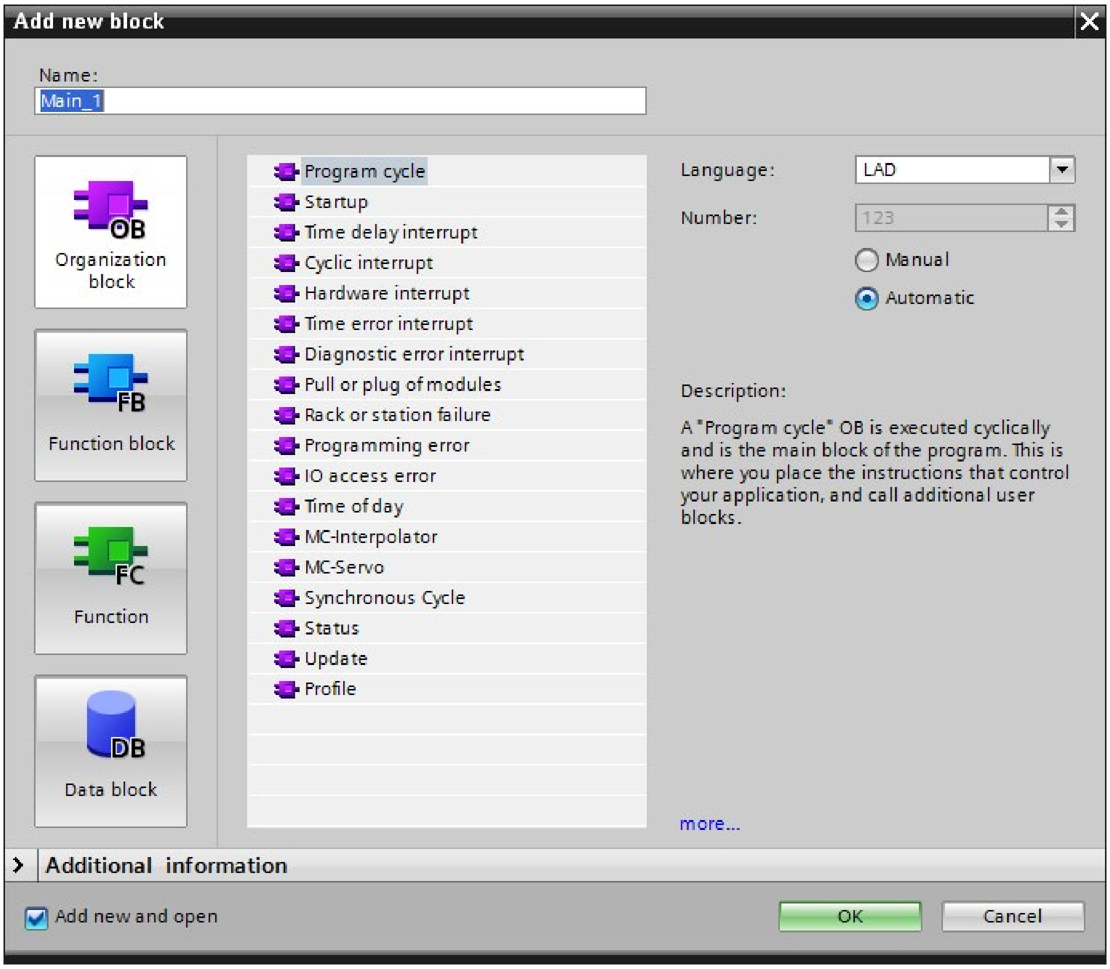
The following organization blocks are supported by the IOT2000EDU and can be configured in the TIA Portal:

OB	Description
OB 1	Free cycle
OB 10	Time-of-day interrupts
OB 20	Time-delay interrupts
OB 30 - OB 38	Cyclic interrupts 1)
OB 100, OB 102	Startup
OB 80, OB 84, OB 85	Asynchronous error interrupts
OB 121	Synchronous error interrupts

1) When a cyclic interrupt OB exceeds the cycle time, a timeout error OB (OB 80) is started.

"Add new block" dialog

1. In the project tree, double-click the entry "Add new block" which is located in the "Program blocks" folder.
   The "Add new block" dialog opens.



Name: Main_1

Language: LAD

Number: 123

( ) Manual
[✓ Automatic]

Description:

A "Program cycle" OB is executed cyclically and is the main block of the program. This is where you place the instructions that control your application, and call additional user blocks.

[✓] Add new and open

Figure 6-1 Blocks

2. Select an organization block, function block, data block or a function in the "Add new block" dialog.
3. You can also select a programming language in the "Language" box.
4. Click "OK" to confirm your selection.

"Instructions" pane

The "Instructions" pane lists all available libraries which will support you during programming. Libraries that are not available are grayed out and cannot be used.

1. Double-click a library element.
   The "Call options" dialog opens.
2. You can change the name and number of the element in this dialog.
3. Click "OK" to confirm.

6.4 Downloading to device

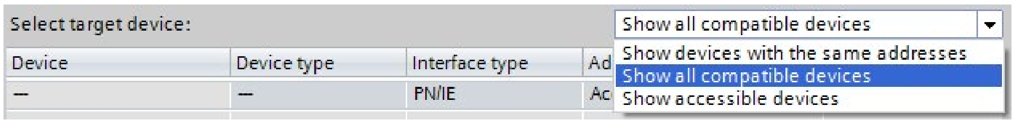
In this chapter you will learn how to download your changes to the device.

Requirements

- You have connected the IOT20x0 to your PC via Ethernet.
- You have started the IOT2000EDU.
- You have created the device in the TIA Portal.
- You have entered an IP address in the properties of the interface.

Procedure

1. Click "Download to device" in the toolbar.
   The "Extended download to device" dialog box opens automatically during the initial download of a project to a device.
2. In this dialog assign the protocol and interface or the destination path for the project in accordance with the device Runtime used.
3. Select the PG/PC interface at which your PC is connected to the IOT2000EDU.
4. Select a search in the drop-down menu.


Select target device:
Show all compatible devices
Device
Device type
Interface type
Ad
—
—
PN/IE
Ac
Show devices with the same addresses
Show all compatible devices
Show accessible devices

5. Click "Start Search".
6. As soon as the search has found the connected IOT2000EDU, click "Download".
   The project is compiled. Warnings and errors during compilation are displayed in the "Load preview" dialog.
7. Click on download after successful compilation.

Result

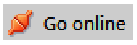
The project was successfully downloaded to the device.

6.5 Go online

This section explains how you connect the device online and further illustrates the functions RUN, STOP and MRES of the SIMATIC IOT2000EDU via TIA Portal.

Go online

After the "Download to device" function has been successfully executed, click "Go online" in the toolbar to establish a connection to the device.



"RUN", "STOP" and "MRES"

As soon as you are connected to the device online, you will have access to the functions RUN, STOP and MRES. You can use these functions to start and stop the CPU and perform a memory reset. For detailed information on how you use these functions, see the section "RUN/STOP/MRES functions via TIA Portal (Page 38)".

Note

Only execute the RUN function when the IOT2000EDU is in "STOP" mode. And you only execute the STOP function when the IOT2000EDU is in "RUN" mode. Otherwise you will receive an error message.

6.6 Diagnostics events

Once you have connected with the device online, you can use the "Online & Diagnostics" function.

Procedure

1. Double-click "Online & Diagnostics" in the project tree.
   The "Online & Diagnostics" view opens in the working area.
2. Open the "Diagnostic buffer" menu under "Diagnostics" > "Diagnostic buffer".

The events are displayed in the diagnostic buffer in the sequence in which they occur.

Configuring and operating shields 7

7.1 SIMATIC IOT2000, Input/Output Module

In this section you will learn which settings you have to make for the pin configurations of a Siemens Arduino shield using the example below.

The Siemens shield has 5 digital inputs, 2 digital outputs and 2 analog inputs. The analog inputs can be selected as current and voltage.
For detailed information on the Siemens shield "Input/Output Module" go here (https://support.industry.siemens.com/cs/document/109745681/iot2000-io-input-output-module?dti=0&lc=en-US).

Assigning an address

Start by assigning the I/O addresses. The I/O addresses must not overlap in the same process image (input/output) memory area. The input and output addresses must be arranged without overlap.

Example:

Digital_in_start can be a value between 0 and 509. If you select the value "100", the values "101" and "102" are assigned. Another input address area, for example, the value for "analog_in_start", must be between 0 and 500 and not 100, 101 and 102.

Program code

```
"address_space":[{
    "digital_in_start":100,
    "digital_out_start":100,
    "pwm_out_start":103,
    "analog_in_start":103,
}],
```

Assigning a GPIO

The five digital inputs of the Siemens shield have the pin numbers 4, 9, 10, 11 and 12. When you enter the value "100" for "digital_in_start", this means:

I-address	102					101								100							
Bit	Reserved	3	2	1	0	7	6	5	**4**	**3**	**2**	**1**	0	7	6	5	**4**	3	2	1	0
Arduino pin	Reserved	19	18	17	16	15	14	13	**12**	**11**	**10**	**9**	8	7	6	5	**4**	3	2	1	0

Arduino pin	Process image (I-area)	I/O description on the shield
4	I100.4	DI4
9	I101.1	DI3
10	I101.2	DI2
11	I101.3	DI1
12	I101.4	DI0

The two digital outputs of the Siemens shield have the pin numbers 7 and 8. When you enter the value "100" for "digital_out_start", this means:

Q-address	102					101								100							
Bit	Reserved	3	2	1	0	7	6	5	4	3	2	1	**0**	**7**	6	5	4	3	2	1	0
Arduino pin	Reserved	19	18	17	16	15	14	13	12	11	10	9	**8**	**7**	6	5	4	3	2	1	0

Arduino pin	Process image (I-area)	I/O description on the shield
7	Q100.7	DQ1
8	Q101.0	DQ0

Assigning PWM outputs

The Siemens Arduino shield does not have any PWM outputs.

Assigning analog

You can enter the two analog inputs either as "0 to 10 V" or "0 to 20 mA". The pin resolution of these inputs is 9 bits for this shield. When you enter the value "103" for "analog_in_start", this means:

I-address	113	111	109	107	105	103
Arduino pin	A5	A4	A3	A2	A1	A0

Arduino pin	Process image (I-area)	I/O description on the shield
A0	IW103	U0
A1	IW105	I0
A2	IW107	U1
A3	IW109	I1

Content of the example file io.conf.sample of the Siemens IOT2000 Arduino shield

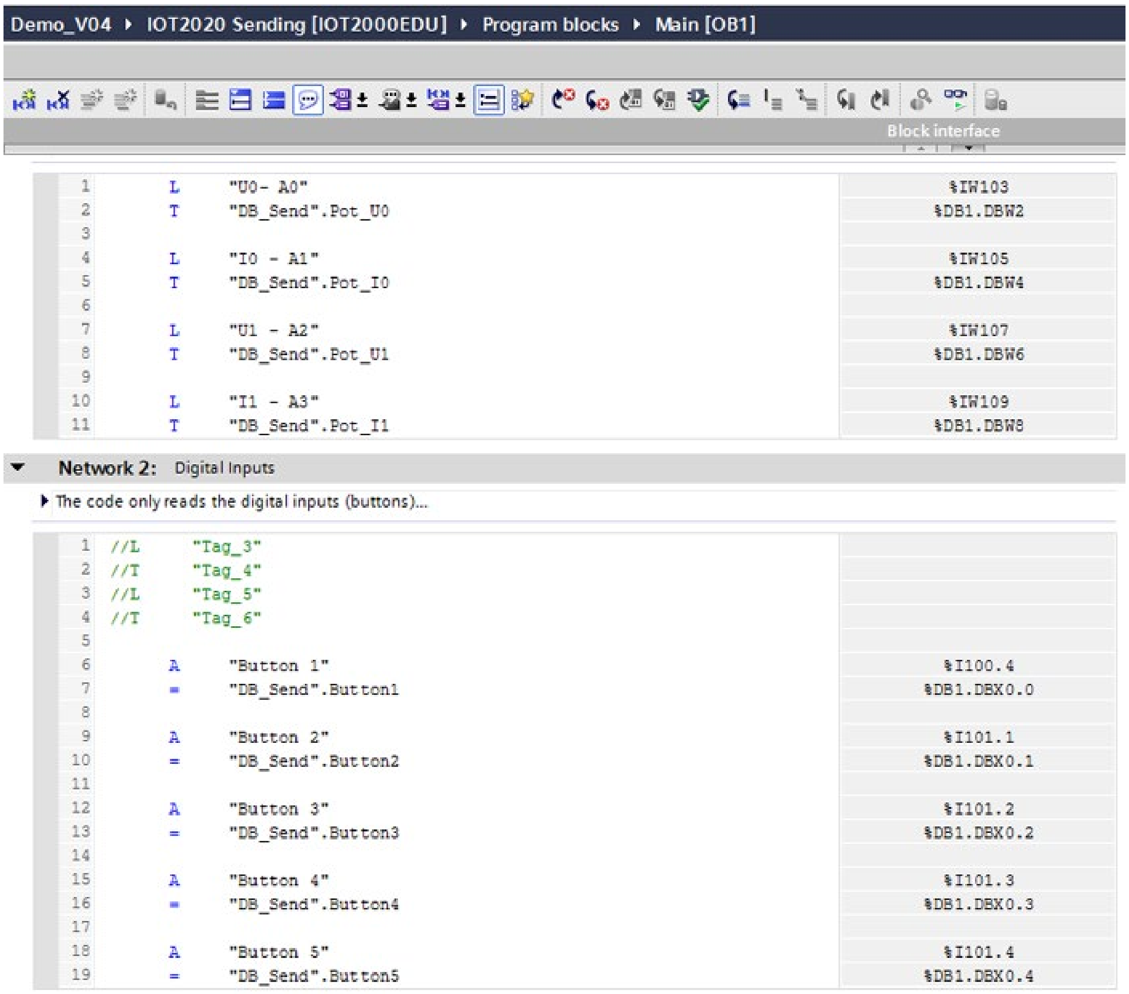
```
Program code                       Comment
{
 "__comments": [{
 "name":                           "SIMATIC IOT2000, Input/Output Module",
 "compatibility":                  "Arduino Uno R3",
 "supports":                       "IOT2020 & IOT2040 Devices",
 "gpio":                           "5 Digital Input Pin: DI4,DI3,DI2,DI1,DI0,
                                   "2 Digital Output Pin: DQ1, DQ0",
 "analog":                         "2 Analog Input Pin: U0,I0,U1,I1 (0 .. 10 V or 0 … 20 mA can be
                                   selected), Resolution:9-bit",
 "address":                        "Look at the surface of IOT2000 I/O module and see port names.
                                    I/O Description Arduino Pin PLC Adress
                                    DI4 4 I100.4
                                    DI3 9 I101.1
                                    DI2 10 I101.2
                                    DI1 11 I101.3
                                    DI0 12 I101.4

                                    DQ1 7 Q100.7
                                    DQ0 8 Q101.0

                                    U0 A0 IW 103
                                    I0 A1 IW 105
                                    U1 A2 IW 107
                                    I1 A3 IW 109",
 }],
 "address_space": [{
 "digital_in_start":100,
```

Program code	Comment

"digital_out_start":100,
"pwm_out_start":103,
"analog_in_start":103,
}],
"gpio": [{
"pin": 4,
"is_output": 0,
"initial_value": 0
}, {
"pin": 9,
"is_output": 0,
"initial_value": 0
}, {
"pin": 10,
"is_output": 0,
"initial_value": 0
}, {
"pin": 11,
"is_output": 0,
"initial_value": 0
}, {
"pin": 12,
"is_output": 0,
"initial_value": 0
}, {
"pin": 7,
"is_output": 1,
"initial_value": 0
}, {
"pin": 8,
"is_output": 1,
"initial_value": 0
}],
"analog": [{
"pin": 0,
"pin_resolution":9
}, {
"pin": 1,
"pin_resolution":9
}, {
"pin": 2,
"pin_resolution":9
}, {
"pin": 3,	

```
Program code                                   Comment
 "pin_resolution":9
 }]
}
```

Example:



Demo_V04 ▸ IOT2020 Sending [IOT2000EDU] ▸ Program blocks ▸ Main [OB1]

```
      L     "U0- A0"                    %IW103
      T     "DB_Send".Pot_U0            %DB1.DBW2

      L     "I0 - A1"                   %IW105
      T     "DB_Send".Pot_I0            %DB1.DBW4

      L     "U1 - A2"                   %IW107
      T     "DB_Send".Pot_U1            %DB1.DBW6

      L     "I1 - A3"                   %IW109
      T     "DB_Send".Pot_I1            %DB1.DBW8
```

Network 2: Digital Inputs

The code only reads the digital inputs (buttons)...

```
//L    "Tag_3"
//T    "Tag_4"
//L    "Tag_5"
//T    "Tag_6"

      A     "Button 1"                  %I100.4
      =     "DB_Send".Button1           %DB1.DBX0.0

      A     "Button 2"                  %I101.1
      =     "DB_Send".Button2           %DB1.DBX0.1

      A     "Button 3"                  %I101.2
      =     "DB_Send".Button3           %DB1.DBX0.2

      A     "Button 4"                  %I101.3
      =     "DB_Send".Button4           %DB1.DBX0.3

      A     "Button 5"                  %I101.4
      =     "DB_Send".Button5           %DB1.DBX0.4
```

7.2 Configuring the Arduino shield

It is possible to use an Arduino shield instead of the Siemens shield.

The section below explains the necessary steps, how you must configure an ARDUINO UNO (Rev3), and it explains how you must adapt the configuration file io.conf.

Interfaces of the IOT20x0 Arduino shield

The figure below shows the motherboard of the SIMATIC IOT20x0, the interfaces X10, X11, X12 and X13 as well as the arrangement of the pins. The orange rectangles indicate the first pin on the respective interface.

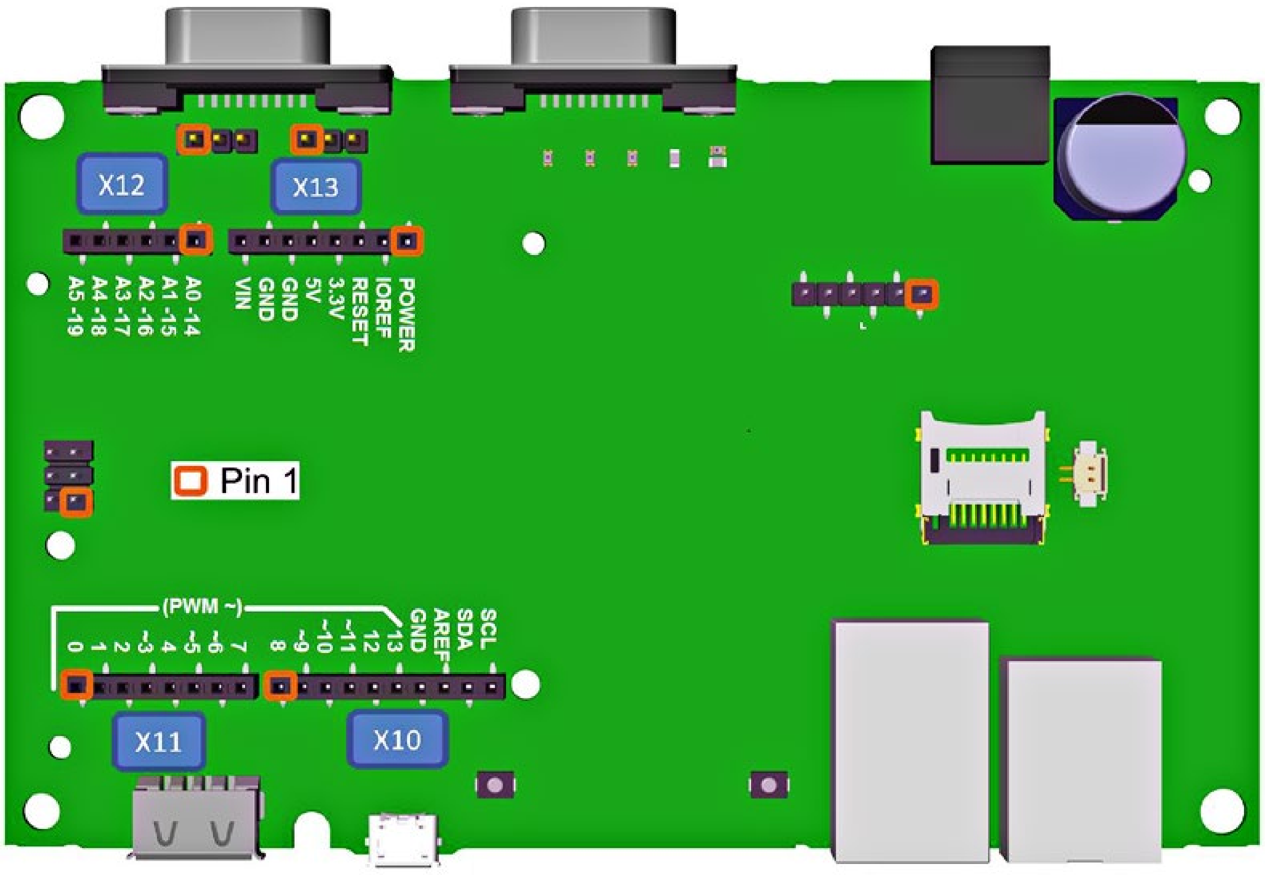


Figure 7-1 IOT20x0 motherboard with interfaces

The table below shows the pin numbers of the interfaces in the io.conf file depending on the operating mode.

Interface	**Pin**	**io.conf pin numbers**		
		Digital	**Analog**	**PWM**
X11	1	0		
	2	1		
	3	2		
	4	3		3
	5	4		
	6	5		5
	7	6		6
	8	7		
X10	1	8		
	2	9		9
	3	10		10
	4	11		11
	5	12		
	6	13		
	7			
	8			
	9	18	4	
	10	19	5	
X12	1	14	0	
	2	15	1	
	3	16	2	
	4	17	3	
	5	18	4	
	6	19	5	

The interface X13 is not configured in the software.

Configuration file io.conf

Fill in the following sections in the configuration file:

1. "address_space": Specification of the start address of PWM, GPIO and AIO.
2. "gpio": Assignment of an Arduino pins as digital input/output.
3. "pwm": Assignment of an Arduino pins as PWM with its period.
4. "analog": Assignment of an Arduino pins as AIO with its resolution.

The following sections explain how you must fill the individual section in the configuration file.

7.2.1 Assigning Address_space

Define the PIN addresses in the following format:

Program code

```
"address_space":[{
    "digital_in_start" :gpioIN_address,
    "digital_out_start" :gpiOut_address,
    "pwm_out_start" :pwm_address,
    "analog_in_start" :aio_address,
}],
```

These addresses correspond to the default process image (OB1 PI) addresses in the PLC program. Only the address range 0 to 511 can be used in form of a byte.

This address section contains:

- "digital_in_start" (uses 3 bytes) and "analog_in_start" (uses 12 bytes) addresses in the process image (inputs) memory area
- "digital_out_start" (uses 3 bytes) and "pwm_out_start" (uses 6 bytes) addresses in the process image (outputs) memory area

The input and output addresses must be arranged in their address range without overlap.

The minimum and maximum values can be used as start addresses in the IO configuration:

Table 7- 1 Process image addresses

Process image	adress_space	min	max
Input	digital_in_start	0	509
Output	digital_out_start	0	509
Output	pwm_out_start	0	506
Input	analog_in_start	0	500

Use the following memory areas in the TIA Portal together with the addresses:

- I, IB, IW, ID: Process image memory areas for reading the input
- Q, QB, QW, QD: Process image memory areas for writing outputs
- PIB, PIW, PID: Direct access for reading the I/O input
- PQB, PQW, PQD: Direct access for writing the I/O output

7.2.2 Assigning a GPIO

Only 1 bit is assigned for each input/output pin.

The IOT20x0 has 20 pins (0 to 19). A total of 3 bytes each are reserved for the Process Input Image (I, PI) and the Process Output Image (Q, PQ). Pin addresses are assigned to the "digital_in_start" or the "digital_out_start" sequentially by the "address_space" area in the configuration file io.conf.

Select "digital_in_start" as gpioIN_address, which means:

I-address	gpioIN_address + 2					gpioIN_address +1								gpioIN_address							
Bit	Reserved	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Arduino pin		19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Select "digital_out_start" as gpiOut_address, which means:

Q-address	gpioOut_address + 2					gpioOut_address +1								gpioOut_address							
Bit	Reserved	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Arduino pin		19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

The following section shows an example of how you enter two pins as GPIO in the io.conf file:

```
Program code                                Comment
"gpio": [{
 "pin": 4,                                  // Arduino pin (0-19)
 "is_output": 1,                            // Output
 "initial_value": 0                         // Logic '0' or '1'
}, {
 "pin": 9,                                  // Arduino pin
 "is_output": 0,                            // Input
 "initial_value": 0                         // Logic '0' or '1'
} ],
```

You can find additional information on the assignment of the GPIOs here (https://support.industry.siemens.com/tf/WW/en/posts/how-is-the-assignment-of-the-gpios/155609?page=0&pageSize=10).

7.2.3 Assigning PWM

1 BYTE is assigned to each PWM output.

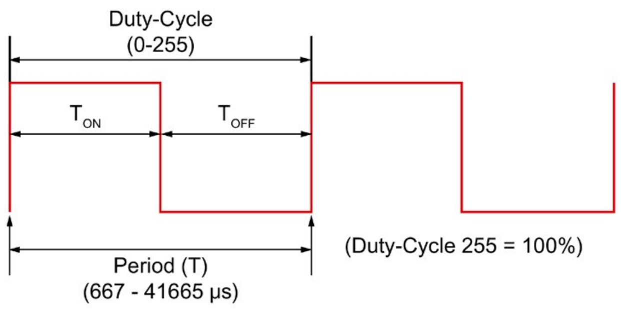
The IOT20x0 has 6 PWM pins: 3, 5, 6, 9, 10 and 11. A total of 6 bytes are reserved for Process Output Image memory area (Q, PQ) to control the PWMs. The address bytes are assigned sequentially to the "pwm_address".

Select "pwm_out_start" as pwm_address, which means:

Q-address	pwm_address +5	pwm_address +4	pwm_address +3	pwm_address +2	pwm_address +1	pwm_address
Adruino pin	11	10	9	6	5	3

The PWM contains two parameters that must be controlled, "Period" and "Duty-Cycle".

"Period" is the time interval that represents the difference in time between two subsequent waves or signals. You enter the "Period" in the io.conf file. Its value is between 667 µs and 41665 µs. These are the period limits in microseconds (µs) supported by the platform. When you enter a value outside these limits, the value is changed to the next maximum or minimum value.



"Duty-cycle" is a parameter that defines the time for which the signal remains "on". This value is given as a percentage. The value can be entered and changed in STEP7 Professional.

If you want to generate a rectangle signal, the value must be changed to 50%, which means you must change the value to 127. In this case the resolution of the duty cycle is 8 bits (max: 255).

Duty cycle = T_{on} / (T_{on} + T_{off}) = x / 255

X is the value you enter in STEP7 Professional.

```
L 127
T QB pwm_address
```
// Pin3 is activated to generate a signal at 50%

The following section shows an example of how you enter two pins as PWM in the io.conf file:

```
Program code                              Comment
"pwm": [{
 "pin": 3,                                // Arduino pin
 "period_us": 10000                       // Period (microsecond)
}, {
 "pin": 5,
 "period_us": 10000
}],
```

Note

Note that the values for "period_us" must be the same for all pins.

7.2.4 Assigning analog

2 BYTES are assigned to each analog input due to the 12-bit resolution.

The IOT20x0 has 6 AIO pins: A0, A1, A2, A3, A4, A5. A total of 12 bytes are reserved for Process Input Image Memory Area (I, PI) to control the AIOs.

Select "analog_in_start" as aio_adress, which means:

I-address	aio_address +10	aio_address +8	aio_address +6	aio_address +4	aio_address +2	aio_address
Adruino pin	A5	A4	A3	A2	A1	A0

In addition to the PIN number, there is another parameter that you enter in the io.conf file: the resolution.

Enter a value between 1 and 12 bits for the resolution. When you enter a value outside this range, the value of the resolution is by default changed to 10 bits.

The following section shows an example of how you enter two pins as AIO in the io.conf file:

```
Program code                              Comment
"analog": [{
 "pin": 0,                                // Arduino pin (A0)
 "pin_resolution":12                      // Resolution
}, {
 "pin": 1,                                // A1
 "pin_resolution":10
}],
```

7.3 Error messages

The io.conf file is parsed in the IOT2000EDU during initialization of the IOT2000EDU. If one of the parameters was not configured correctly and/or contains a spelling error, the IOT2000EDU cannot start and aborts with an error message.

The following items are checked in the software. If the io.conf file is invalid, the corresponding message is displayed:

1. Address areas must be compatible with the table "Process image addresses (Page 32)".
   Message:
   "Digital Input Start Address should be greater than 0 and less than 509"
   "Digital Output Start Address should be greater than 0 and less than 509"
   "PWM Start Address should be greater than 0 and less than 506"
   "Analog Input Start Address should be greater than 0 and less than 500"
2. Addresses must not overlap. For example: If "digital_in_start=100", analog_in_start may not be "101"; it must be at least "103".
   Message:
   "Invalid address space configuration."
   "Please be sure that Start Address values don’t overlap"
   "GPIO allocates 3 Bytes, PWM allocates 6 Bytes and Analog Input allocates 12 Bytes"
3. GPIO pin number must be between 0 and 19.
   Message:
   "GPIO Pin number should be 0 – 19"
4. GPIO Is_output must be "0" or "1".
   Message:
   "is_output should be 1 or 0"
5. GPIO initial_value must be "0" or "1".
   Message:
   "initial_value should be 1 or 0"
6. PWM pin numbers must be 3, 5, 6, 9, 10, 11.
   Message:
   "Only 3,5,6,9,10,11 pins can be used as PWM"
7. PWM period must be between 667 and 41665 µs. Otherwise the value is changed to the next minimum or maximum value. You will see some informational messages.
   Message:
   "PWM Period %d is more than maximum supported value. It will be set as %d"
   "PWM Period %d is less than minimum supported value. It will be set as %d"
8. AIO pins must be between 0, 1, 2, 3, 4 and 5.
   Message:
   "Analog Pin numbers should be between 0-5"
9. AIO resolution must be between 1 and 12. Otherwise the AIO resolution is changed to the value "10" by default.
   Message:
   "Analog Resolution should be between 1-12; it will be set as 10"

10. Some pins are multifunctional. You should therefore be careful when you configure the pins. For example: Pin 3 can be set as PWM as well as GPIO.
    Message:
    "PWM Pin Number: %d has already been configured. Please check multifunctional GPIO/PWM Pins. Invalid PWM Pin Configuration"

All error messages are saved in the file "/var/log/messages". You can display the messages in different ways:

```
cat /var/log/messages
```

`tail-f /var/log/messages` (indicates the end of the file.)

See also

Assigning Address_space (Page 32)

Operating IOT2000EDU 8

8.1 RUN/STOP/MRES functions via TIA Portal

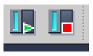
This section further illustrates the functions RUN, STOP and MRES of the SIMATIC IOT2000EDU via TIA Portal.

As soon as you are connected to the device online, you will have access to the following functions:

- RUN, STOP and MRES: Over the CPU operating panel in the "Online & Diagnostics" working area.
- RUN and STOP: Using the buttons "Start CPU" and "Stop CPU" in the toolbar.

"RUN" and "STOP"

1. Click the "Start CPU" button in the toolbar when you want to set the IOT2000EDU to RUN mode or "Stope CPU" when you want to set the IOT2000EDU to STOP mode.

2. Click "OK" in response to the confirmation prompt.

Or:

1. Activate the "Online Tools" task card of the IOT2000EDU.
2. Click the "RUN" button in the "CPU control panel" pane if you want to change the IOT2000EDU to RUN mode or the "STOP" button if you want to change the IOT2000EDU to STOP mode.
3. Click "OK" in response to the confirmation prompt.

Note

Only execute the RUN function when the IOT2000EDU is in "STOP" mode. And you only execute the STOP function when the device is in "RUN" mode. Otherwise you will receive an error message.

MRES

The MRES function allows you to perform a memory reset of the CPU. The memory of the device is deleted in the process and the device is set to the so-called "initial state".

This means:

- All user data is deleted.
- Depending on the target system, user programs and hardware configurations can be deleted.
- All connections to the module are disconnected.

Proceed as follows to perform a memory reset:

1. Activate the "Online Tools" task card of the IOT2000EDU.
2. Click the "MRES" button in the "CPU operator panel" palette.
3. Click "OK" in response to the confirmation prompt.

8.2 RUN/STOP/MRES functions via command line

In the chapter below you will learn how to change the operating state of the IOT2000EDU using the command line program "CPU_Control". The program was installed together with the "IOT2000EDU" application in the directory "/usr/local/bin".

"CPU_Control" can currently execute five commands: RUN, STOP, MRES, Shutdown and Status

RUN mode

Use the following command to switch the IOT2000EDU to RUN mode:

```
CPU_Control run
```

STOP mode

Use the following command to switch the IOT2000EDU to STOP mode:

```
CPU_Control stop
```

MRES

Keep in mind that you must set the IOT2000EDU to STOP mode before you execute this command.

You trigger the memory reset of the IOT2000EDU controller with the following command:

```
CPU_Control mres
```

Shutdown

Use the following command to shut down the IOT2000EDU controller:

```
CPU_Control shutdown
```

Status

Use the following command to check the current operating state of the IOT2000EDU controller:

```
CPU_Control status
```

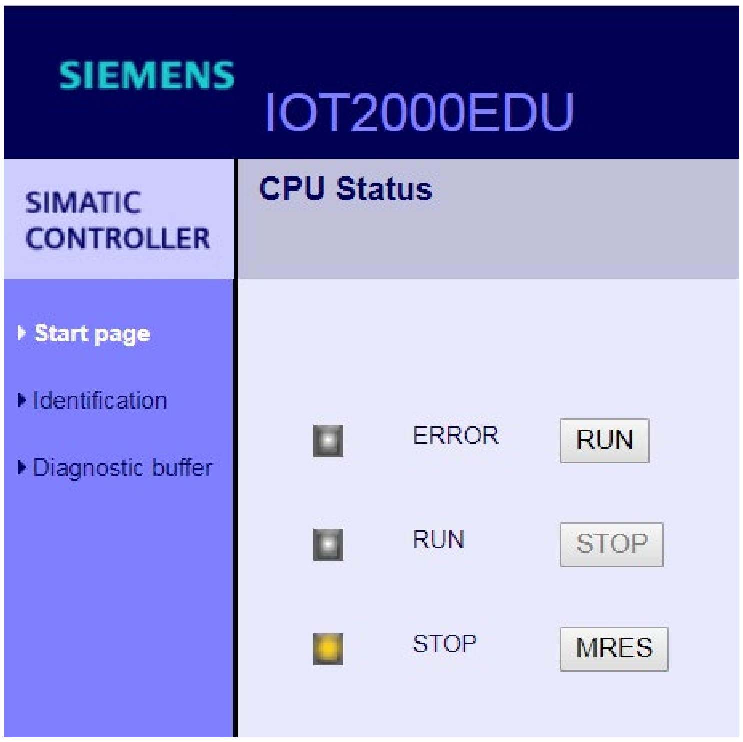
This way you can also check whether a preceding command was successful or has failed.

Help

When you execute the command CPU_Control without any of the five parameters, the following help screen is displayed:

```
Usage :

Switch to RUN state : CPU_Control run

Switch to STOP state : CPU_Control stop

Trigger MASTER RESET : CPU_Control mres

Shutdown the Controller : CPU_Control shutdown

Get Status : CPU_Control status

Show version info : CPU_Control -v
```

8.3 RUN/STOP/MRES via Web server



On this page of the Web server, you can change the operating mode of the IOT2000EDU using the buttons and perform a memory reset of the IOT2000EDU. You can find additional information on this in the "Home page (Page 44)" section.

9 Web server

9.1 Introduction

The following section describes the IOT2000EDU Web server activation, Web page content and interaction with the IOT2000EDU.

The IOT2000EDU Web server is supplied automatically with the installation of the IOT2000EDU installation package "IOT2000EDU_1.1.0_i586.ipk".

9.2 Requirements

Successful installation of the IOT2000EDU V1.1 requires the following:

- "init system" must be present in the operating system.
  You can see if "init system" is present by checking whether "/etc/init.d" is present in the operating system.
- "node v6.9.2" package must be available in the operating system

9.3 Operating the Web server with Linux commands

The IOT2000EDU installation automatically adds the Web server application to the autostart service and activates the process. If the system overloads or possibly crashes, you can start the IOT2000EDU Web server manually with the following command:

```
/etc/init.d/IOT2000Web start
```

The IOT2000EDU Web server is automatically activated during the boot process of the IOT20x0. You can also still use the following commands:

- Stop Web server: `/etc/init.d/IOT2000Web stop`
- Display Web server status: `/etc/init.d/IOT2000Web status`

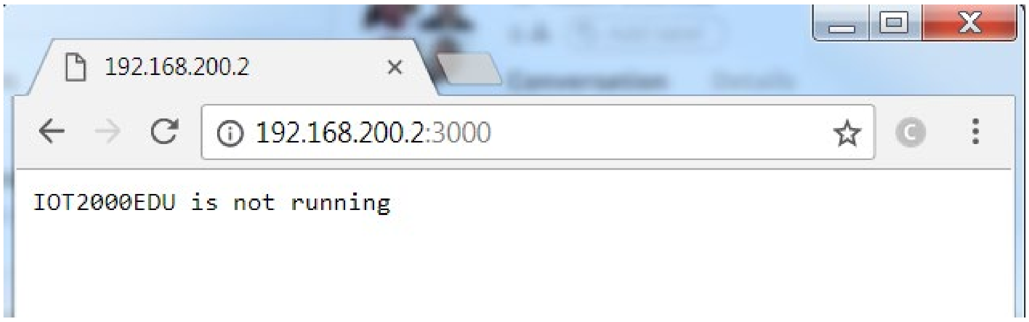
9.4 Accessing Web server

To access the Web server, use the IOT20x0 IP address via port "3000" as `ipaddress:3000`

Example:

192.168.200.1:3000

If the IOT2000EDU is not running on an IOT20x0 device, the server refers you to the following error page:



```
IOT2000EDU is not running
```

9.5 Web server pages

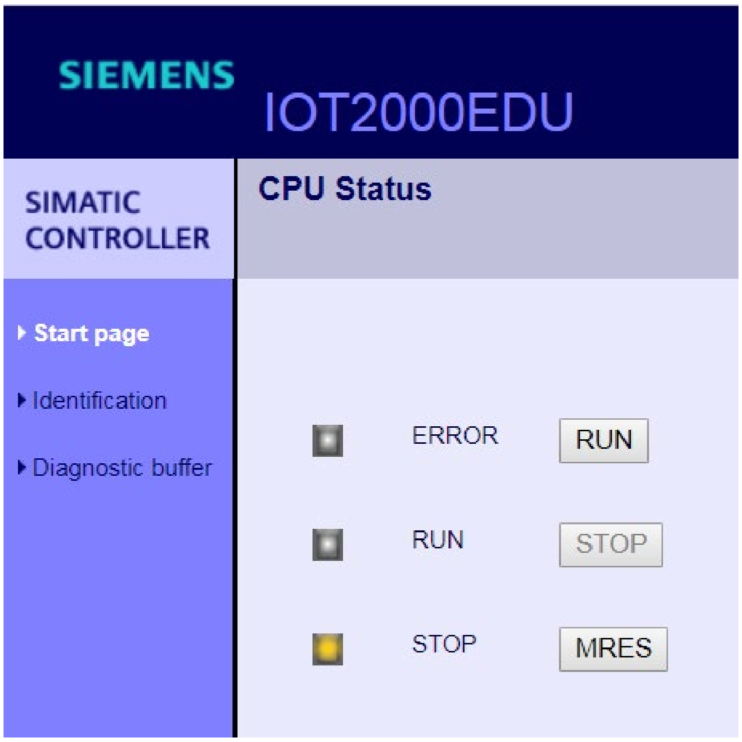
9.5.1 Introduction

The following three Web pages are the main component of the IOT2000EDU Web server:

- Home page
- Identification
- Diagnostic buffer

Web pages are not automatically updated. You therefore have to load the pages manually if you want to update the website.

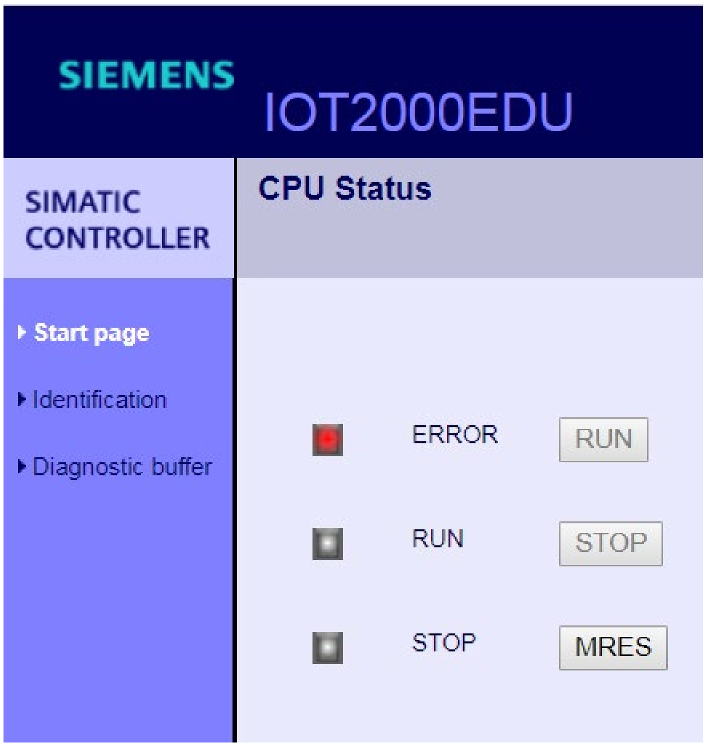
9.5.2 Home page



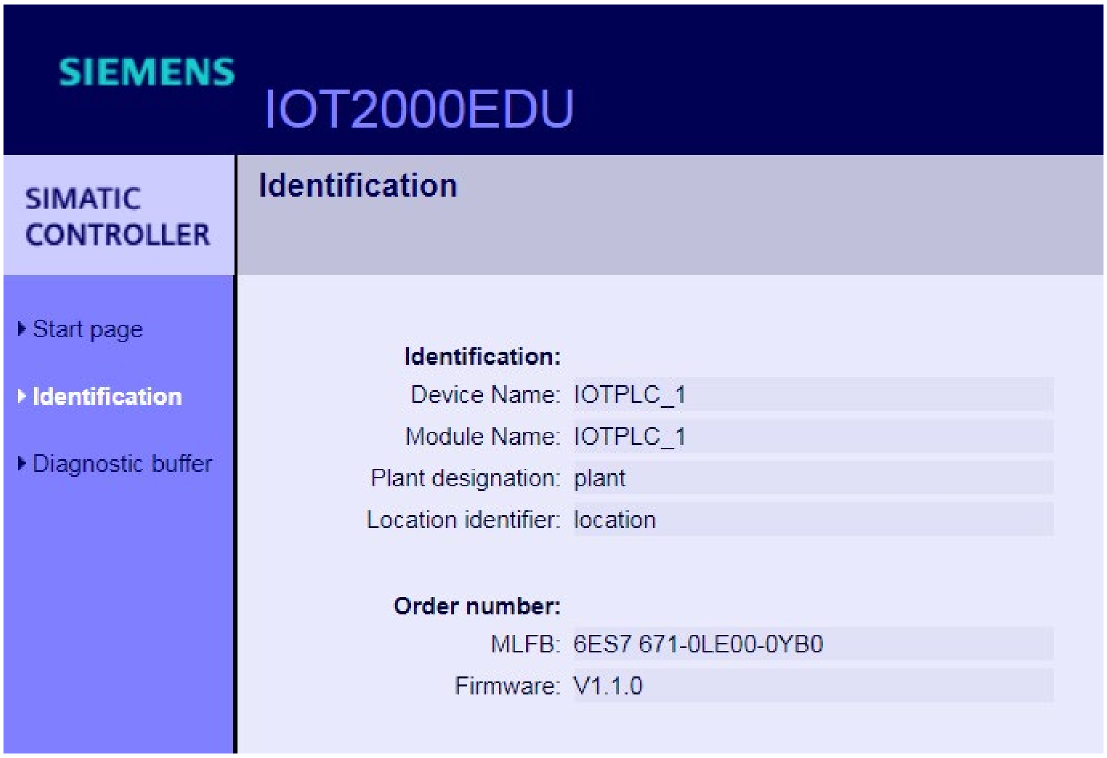
The home page is the default Web page of the IOT2000EDU Web server. On this page, you can see the status of the IOT2000EDU based on the colored LEDs. Each LED has its own color during activity. However, only one LED can light up at a time. You can change the operating mode of the IOT2000EDU using the "RUN" and "STOP" buttons, and perform a memory reset of the IOT2000EDU using the "MRES" button.

The LEDs can indicate the following states of the IOT2000EDU:

"RUN" 🟩	The "RUN" button is grayed-out and the "RUN" LED lights green.
"STOP" 🟨	The "STOP" button is grayed-out and the "STOP" LED lights yellow.
"ERROR" 🟥	The "RUN" and "STOP" buttons are grayed-out and the "ERROR" LED lights red.



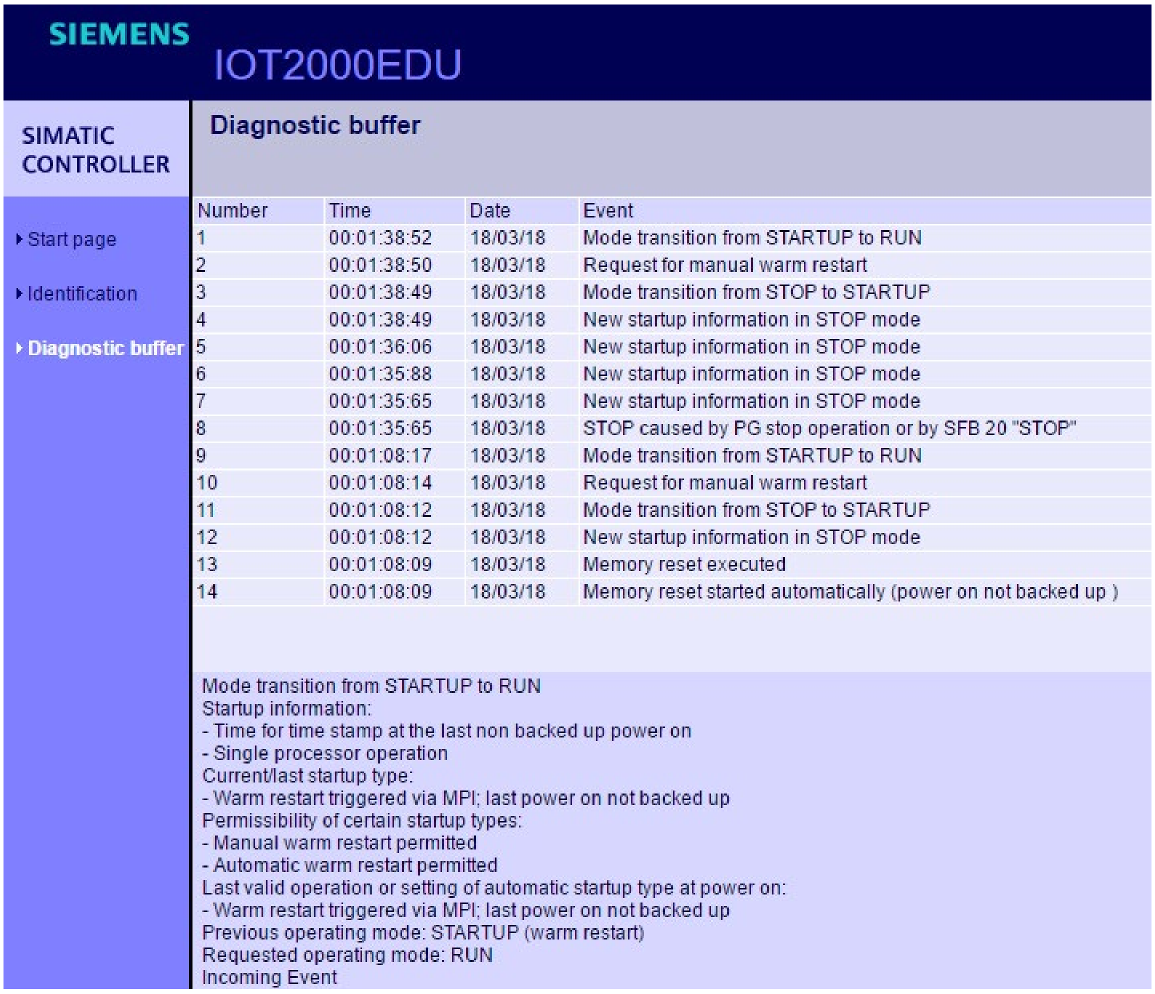
9.5.3 Identification



Identification:	
Device Name:	IOTPLC_1
Module Name:	IOTPLC_1
Plant designation:	plant
Location identifier:	location
Order number:	
MLFB:	6ES7 671-0LE00-0YB0
Firmware:	V1.1.0

This page contains identification, maintenance and basic project information that has been loaded from the TIA Portal to the IOT2000EDU.

9.5.4 Diagnostic buffer



Diagnostic buffer

Number	Time	Date	Event
1	00:01:38:52	18/03/18	Mode transition from STARTUP to RUN
2	00:01:38:50	18/03/18	Request for manual warm restart
3	00:01:38:49	18/03/18	Mode transition from STOP to STARTUP
4	00:01:38:49	18/03/18	New startup information in STOP mode
5	00:01:36:06	18/03/18	New startup information in STOP mode
6	00:01:35:88	18/03/18	New startup information in STOP mode
7	00:01:35:65	18/03/18	New startup information in STOP mode
8	00:01:35:65	18/03/18	STOP caused by PG stop operation or by SFB 20 "STOP"
9	00:01:08:17	18/03/18	Mode transition from STARTUP to RUN
10	00:01:08:14	18/03/18	Request for manual warm restart
11	00:01:08:12	18/03/18	Mode transition from STOP to STARTUP
12	00:01:08:12	18/03/18	New startup information in STOP mode
13	00:01:08:09	18/03/18	Memory reset executed
14	00:01:08:09	18/03/18	Memory reset started automatically (power on not backed up)

Mode transition from STARTUP to RUN
Startup information:
- Time for time stamp at the last non backed up power on
- Single processor operation
Current/last startup type:
- Warm restart triggered via MPI; last power on not backed up
Permissibility of certain startup types:
- Manual warm restart permitted
- Automatic warm restart permitted
Last valid operation or setting of automatic startup type at power on:
- Warm restart triggered via MPI; last power on not backed up
Previous operating mode: STARTUP (warm restart)
Requested operating mode: RUN
Incoming Event

This page contains all diagnostic logs created by an event in the IOT2000EDU.

At the top of the page, you can find the events and related general information, such as event ID, time and date.

When you select an entry from the list, detailed information is displayed at the bottom of the page.

Function library 10

The CPU function library is a binary file that contains executable functions. The CPU function library can be called by the IOT2000EDU.

This allows you to implement your own software in a higher programming language (C/C++) and have it work with the IOT2000EDU.

10.1 Overview of the CPU function library

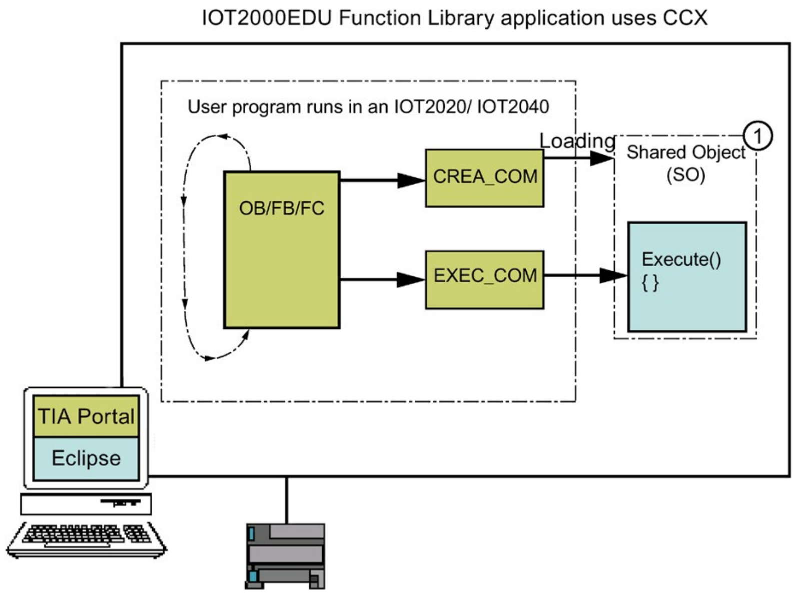
With a development environment like Eclipse, you can create a CPU function library as a "Shared Object (SO)". The SO contains your software written in the C/C++ programming language. You program your user program in the TIA Portal so that the TIA Portal calls the "CREA_COM" (SFB65001) and "EXEC_COM" (SFB65002) instructions. This causes the SO to be loaded dynamically and your code being executed directly by the IOT2000EDU user program.

To connect to your software, call the instructions from the IOT2000EDU program. Using the "CREA_COM" instruction, the IOT2000EDU loads an SO. The "EXEC_COM" instruction executes the loaded SO. The TIA Portal contains these two instructions.

By using the CPU function libraries to integrate your software into a user program you can enhance the performance of an IOT2000EDU user program.

The use of a CPU function library offers you the following advantages:

- Recording of special control logic written in C or C++
- Using complex or proprietary algorithms or control functions, e.g. PID or gas flow. These algorithms can be written and maintained in C/C++ using integrated development environments (IDE) such as Eclipse, or integrated from model-based development environments such as Matlab/Simulink.
- Connecting to other applications or devices, such as Motion Control or Arduino Shields, from third-party manufacturers. Many manufacturers distribute their Arduino drivers in C/C++.
- Accessing functions of the IOT20x0 board which cannot be accessed with standard programming languages.


IOT2000EDU Function Library application uses CCX
User program runs in an IOT2020/ IOT2040
OB/FB/FC
CREA_COM
EXEC_COM
Loading
Shared Object (SO)
1
Execute() { }
TIA Portal
Eclipse

① Use CCX to embed your C/C++ code into an SO.

The CPU function library allows you to create your own thread functions. These allow you to execute your own code outside the main thread of the execution.

Programming multiple function blocks

IOT2000EDU can load more than one SO. Each SO can perform several tasks. When the "CREA_COM" instruction loads an SO, it outputs a unique object handle (OBJHandle). Calling the "EXEC_COM" instruction uses the OBJ handle of a specific object to execute the software for the same SO.

10.2 Creating a user-defined SO file

The functions of a function library are implemented by means of the "EDU_Function.cpp" file. The "EDU_Function.cpp" file is available in the supplied example project. The most important function within the file is the "Execute" function, in which you embed the actual functions of your function library.

10.2.1 Programming the "Execute" function

The "Execute" function is the function that is started from the user program with the "EXEC_COM" instruction. The following parameters are transferred when it is called:

- Pointer to the buffer of the input data
- Pointer to the buffer of the output data
- Length of buffers
- Command variable

```
external "C" SEA_EXT_CALL EDU_RESULT Execute (unsigned long command, long nInBytes,byte bInData [],
long nOutBytes, long * pnUsedOutBytes, byte bOutData [] )
{
         CWinLCReadData Input(nInBytes, bInData);
         CWinLCReadWriteData Output(nOutBytes, bOutData);
         EDU_RESULT retVal = EDU_SUCCESS;

         switch(command)
         {
          case 0:
          // add your functions here
          retVal = Standard_Deviation(Input, Output);
          break;
          case 1:
          // add your functions here
          retVal = Function_1(Input, Output);
          break;
          default:
          retVal = EDU_COMMAND_NOT_IMPLEMENTED;
         }
         *pnUsedOutBytes = Output.EDU_LastByteChanged() + 1;

         return retVal;
} //
Execute
```

The interface of the "Execute" function must not be changed. The parameters are defined as follows:

- Unsigned long command:
  The "command" parameter makes it possible to implement various functions within a function library. These functions are identified and called via a "Switch case" instruction This allows almost any number of functions to be implemented within a function library.
- Long nInBytes:
  This parameter contains the length of the input data buffer in bytes.
- byte bInData []:
  The parameter is a pointer to the input data buffer. The individual variables are arranged one after the other as they are stored by the IOT2000EDU user program. The individual input values can be read using the "Data Access Helper" classes.
  For details on this, see section "Data Access Helper classes (Page 53)".
- long nOutBytes:
  This parameter contains the length of the output data buffer in bytes.
- long * pnUsedOutBytes:
  This parameter contains the length of the output data buffer length actually used.
- byte bOutData []:
  This parameter is a pointer to the output parameter buffer. The individual variables are located one behind the other as they are to be used by the IOT2000EDU user program. Individual input values can be read or written using the "Data Access Helper" classes.
  For details on this, see section "Data Access Helper classes (Page 53)".
- Return value:
  EDU_Result returns the status value or error messages of the called function. The standard return values (e.g. EDU_SUCCESS) are defined in the "EDUso.h" file.

10.2.2 Programming additional functions

In addition to the "Execute" function, there are other functions that are called automatically in certain situations. These functions must be included in the code, but can otherwise be left blank.

The interface functions of the CPU function library are initially empty. This allows you to program according to the requirements of your application.

IOT2000EDU Runtime automatically calls these interface functions when:

- The IOT2000EDU user program loads the SO.
- The IOT2000EDU user program executes the SO simultaneously.
- The IOT2000EDU changes from STOP to STARTUP, from HOLD to RUN, or when the SO is loaded for the first time.
- The IOT2000EDU switches to STOP or HOLD mode.

The interface functions:

ODKCreate:

The "ODKCreate" function initializes data or creates objects after the "CREA_COM" instruction has loaded the SO. If the Shared Object is initially loaded and you want to perform evaluations, load it into "ODKCreate". Otherwise leave "ODKCreate" empty.

Activate:

The "Activate" function is called before the IOT2000EDU changes from STOP or HOLD mode to STARTUP or RUN mode.
"Activate" is always called after the SO is loaded and before the first call of the "Execute" function starts. If you want to perform evaluations before the initial call of the "Execute" function, you can load it into the "Activate" function. Otherwise, leave "Activate" empty.

DeActivate:

The "DeActivate" function is called after IOT2000EDU changes from STARTUP or RUN mode to STOP or HOLD mode. If you want to perform the evaluations after the change from STOP or HOLD, you can load them into the "DeActivate" function. Otherwise, leave the function empty.

ODKRelease:

The "ODKRelease" function is responsible for publishing/unloading the shared object. If you want to perform evaluations when the object is published, load it in ODKRelease. Otherwise, leave the function empty.

The IOT2000EDU publishes the SO during a memory reset (MRES) or a controller shutdown. The IOT2000EDU always calls "DeActivate" before calling "ODKRelease", because both the MRES and the controller shutdown put the controller in STOP mode.

Scan cycle impact:

Your software in the "Execute" function and the sub-functions are part of the Main Program Scan cycle when called by the "EXEC_COM" instruction. The IOT2000EDU program executes the instruction which your software calls as a simple instruction. This instruction call cannot be interrupted. The watchdog timer, OBs and process alarms are not available during execution of your software. They are operated after the instruction of your software has been called.

Note

User-specific software that significantly extends cycle times delays the evaluation of critical activities in the controller.

To avoid delays in the OB scan cycle, do not write evaluations into the "Execute" function or sub-functions that extend the scan cycle beyond an acceptable cycle time. This also includes calls to a non-deterministic function, such as "Printf".

Note

The CPU function library guarantees that IOT2000EDU calls these functions from a single execution string.

10.2.3 Data Access Helper classes

The Data Access Helper classes allow access to the variables transferred by IOT2000EDU in the input and output parameters.

The two Data Access Helper classes are:

- CWinLCReadData: Reading the variables from the input parameters
- CWinLCReadWriteData: Reading and writing the variables in the output parameters

The IOT2000EDU controller transfers the input parameters in S7 format and also expects the output parameters to be returned in S7 format. All input and output variables starting from the passed pointers are stored one after the other. Note the following:

To use the data in C/C++ code, it must be converted in the Data Access Helper classes using the access functions. Likewise, data must be converted into the S7 format before being returned to the IOT2000EDU.

The methods EDU_ReadS7<data type> and EDU_WriteS7<data type> perform these type conversions automatically and thus allow easy and secure access to the input and output variables.

The example project shows you how to use the helper classes with a basic example:

```
EDU_RESULT Function_1 ( CWinLCReadData& rInput ,CWinLCReadWriteData& rOutput )
{
      short  MyInteger;
      float  MyReal;
      bool   MyBool;
      // extract input variables from input data buffer
      rInput.EDU_ReadS7INT(0,MyInteger);
      rInput.EDU_ReadS7REAL(2,MyReal);
      rInput.EDU_ReadS7BOOL(6,0,MyBool);
      // Add your Code here
      // printf("\n---------- InputData ----------\nMyInteger: %d \n MyReal: %.5f
      \n MyBool: %s\n ",MyInteger,MyReal,MyBool ? "TRUE" : "FALSE");
      MyInteger +=5;
      MyReal += (float)3.0000;
      MyBool = false;
      // printf("\n---------- OutputData ----------\nMyInteger: %d \n MyReal: %.5f
      \n MyBool: %s\n ",MyInteger,MyReal,MyBool ? "TRUE" : "FALSE");

      // put output variables into output data buffer
      rOutput.EDU_WriteS7INT(0,MyInteger);
      rOutput.EDU_WriteS7REAL(2,MyReal);
      rOutput.EDU_WriteS7BOOL(6,0,MyBool);

      return EDU_SUCCESS;
}
```

In the example project, a UDT with the following elements is used as an input parameter:

```
UDT:
 Integer     MyInteger
 Real        MyReal
 BOOL        MyBool
END UDT
```

The individual variables or members of the UDT are stored in the input parameter as follows:

Offset	0		2				6
	MyInteger		MyReal				MyBool

Supported S7 data types

The helper classes support the following S7 data types:

Table 10- 1 CWinLCReadData

EDU_GetBuffer	Returns a pointer to the data area.
EDU_SetBuffer	Initializes the "Input Data" area and file size.
EDU_GetBufferSize	Returns the size of the data area (in bytes).
EDU_ReadS7BYTE	Reads a byte (1 byte) from the data area.
EDU_ReadS7WORD	Reads a word (2 bytes) from the data area.
EDU_ReadS7DWORD	Reads a double word (4 bytes) from the data area.
EDU_ReadS7S5TIME	Reads a 16-bit time value (2 bytes).
EDU_ReadS7DATE	Reads a date value (2 bytes) from the data area.
EDU_ReadS7TIME_OF_DAY	Reads the time (4 bytes) from the data area.
EDU_ReadS7INT	Reads an integer (2 bytes) from the data area.
EDU_ReadS7DINT	Reads a double integer (4 bytes) from the data area.
EDU_ReadS7REAL	Reads a floating point number (4 bytes) from the data area.
EDU_ReadS7TIME	Reads a time value (4 bytes) from the data area.
EDU_ReadS7CHAR	Reads a character (1 byte) from the data area.
EDU_ReadS7BOOL	Reads a Boolean value (1 bit) from the data area.
EDU_ReadS7STRING_LEN	Reads the string length information of an S7 string in the data area.
EDU_ReadS7STRING	Reads an S7 string from the data area and returns it as C++ character string.
EDU_ReadS7DATE_AND_TIME	Reads a generic data and time area.

Table 10- 2 CWinLCReadWriteData

EDU_GetBuffer	Returns a pointer to the data area.
EDU_SetBuffer	Initializes the "Input Data" area and file size.
EDU_GetBufferSize	Returns the size of the data area (in bytes).
EDU_WriteS7BYTE	Writes a bytes (1 byte) to the data area.
EDU_WriteS7WORD	Writes a WORD (2 bytes) to the data area.
EDU_WriteS7DWORD	Writes a double WORD (4 bytes) to the data area.
EDU_WriteS7INT	Writes a 16-bit time value (2 bytes).
EDU_WriteS7DINT	Writes a date value (2 bytes) to the data area.
EDU_WriteS7S5TIME	Writes the time of day (4 bytes) to the data area.
EDU_WriteS7TIME	Writes an integer (2 bytes) to the data area.
EDU_WriteS7DATE	Writes a double integer (4 bytes) to the data area.
EDU_WriteS7TIME_OF_DAY	Writes a floating point number (4 bytes) to the data area.
EDU_WriteS7CHAR	Writes a time value (4 bytes) to the data area.
EDU_WriteS7REAL	Writes a character (1 byte) to the data area.
EDU_WriteS7BOOL	Writes a boolean value (1 bit) to the data area.
EDU_WriteS7STRING	Writes an S7 string to the data area and returns it as C++ characters (string).
EDU_WriteS7DATE_AND_TIME	Writes a generic data and time range.

You can obtain information about the relevant data types in the "Data types (Page 56)" section.

You can find additional information about the supported data types of the IOT2000EDU in the online help of the TIA Portal. The same range of data types can be used in the IOT2000EDU as in an S7-300 controller.

Note

The methods of the Data Access Helper classes may only be used within the "Execute" function. The helper classes therefore help you avoid programming errors, e.g. out-of-range errors or writing to invalid pointers.

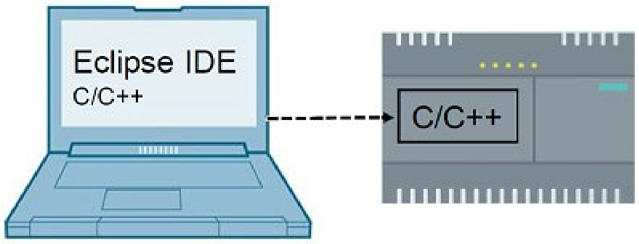
10.2.4 Data types

The following table lists examples of all relevant data types:

C/C++ (Simulink Coder)	C/C++ (IOT2000EDU function library)	TIA Portal	Bytes	Helper functions EDU_ReadS7<data type> EDU_WriteS7<data type>
boolean_T	bool	BOOL	1 bit	EDU_ReadS7BOOL EDU_WriteS7BOOL
int8_T, uint8_T, char_T, uchar_T, byte_T	char	CHAR	1	EDU_ReadS7CHAR EDU_WriteS7CHAR
int16_T, uint16_T	short	INT	2	EDU_ReadS7INT EDU_WriteS7INT
int32_T, uint32_T, int_T, uint_T	long	DINT	4	EDU_ReadS7DINT EDU_WriteS7DINT
ulong_T	long	DINT	4	EDU_ReadS7DINT EDU_WriteS7DINT
real_T	float	REAL (4 bytes) Note: Data loss	4	EDU_ReadS7REAL EDU_WriteS7REAL
real32_T	float	REAL	4	EDU_ReadS7REAL EDU_WriteS7REAL
"Typ"_T["Length"]	"Typ" ["Length"]	Array [lo .. hi] of type	Depending on size and type	
Struct (Simulink: BUS element)	struct	Struct	Depending on size and type	

10.3 Creating function library programs with Eclipse

This section describes the functionality and implementation of a CPU function library example program that was created in Eclipse IDE. Furthermore, you can learn how to compile the program and write it to the IOT20x0 devices. In addition, you can learn how to create a simple TIA Portal project and how to use a shared object (SO) in the PLC user program. The following figure provides an overview of the CPU function library applications:



Example CPU function library project in Eclipse

You can use the Eclipse example project to start creating a function library (SO). This example project contains all interface functions with a simple customer-specific application. This application calculates the "Standard Deviation" of a number group.

In addition to the IOT2000EDU installation package, the example project is delivered as a separate file.

You can also create your own project in another IDE in addition to Eclipse. But you must take notice of the system requirements.

You can work with Windows or a Linux operating system. If you work with Windows, you need a cross compiler.

If the IOT2000EDU was installed on the IOT20x0 devices, the following packages are already pre-installed:

- libc6
- libstdc++6

IOT2000EDU requires the libc6 and libstdc++6 libraries. Therefore, both the native compiler and the cross compiler must support these libraries to generate the SO file. A few alternative development environments are recommended for generating the SO.

10.3.1 Alternative development environments

Windows: Install Eclipse via ODK 1500S

If ODK 1500S is already installed in the system, you can access the Eclipse version (Kepler) under the following path:

"C:\Program Files (x86)\Siemens\Automation\ODK1500S\V2.5\eclipse\Eclipse.exe"

Import the example program with the "Release" build or create your own project using Kepler Eclipse. To generate an SO for IOT2000EDU, install IOT2000 SDK for cross compiling with Windows operating systems. Follow the IOT2000 SDK instructions below for the cross compiler settings.

Windows: Download and install Eclipse

You need "Eclipse IDE for C/C++ Developers" to create a new C/C++ project or to open the example project, "EDU_StandardDeviation". You can download it from the following link: https://www.eclipse.org/downloads/eclipse-packages/ (https://www.eclipse.org/downloads/eclipse-packages/)

To generate an SO for IOT2000EDU, install IOT2000 SDK for cross compiling with Windows operating systems. Follow the IOT2000 SDK instructions (Page 59) for cross compiler settings.

Linux: Install Eclipse

If you are using a Linux operating system, download an Eclipse version. Under Linux, a native compiler is available that is compatible with the libc6 and libstdc++6 libraries.

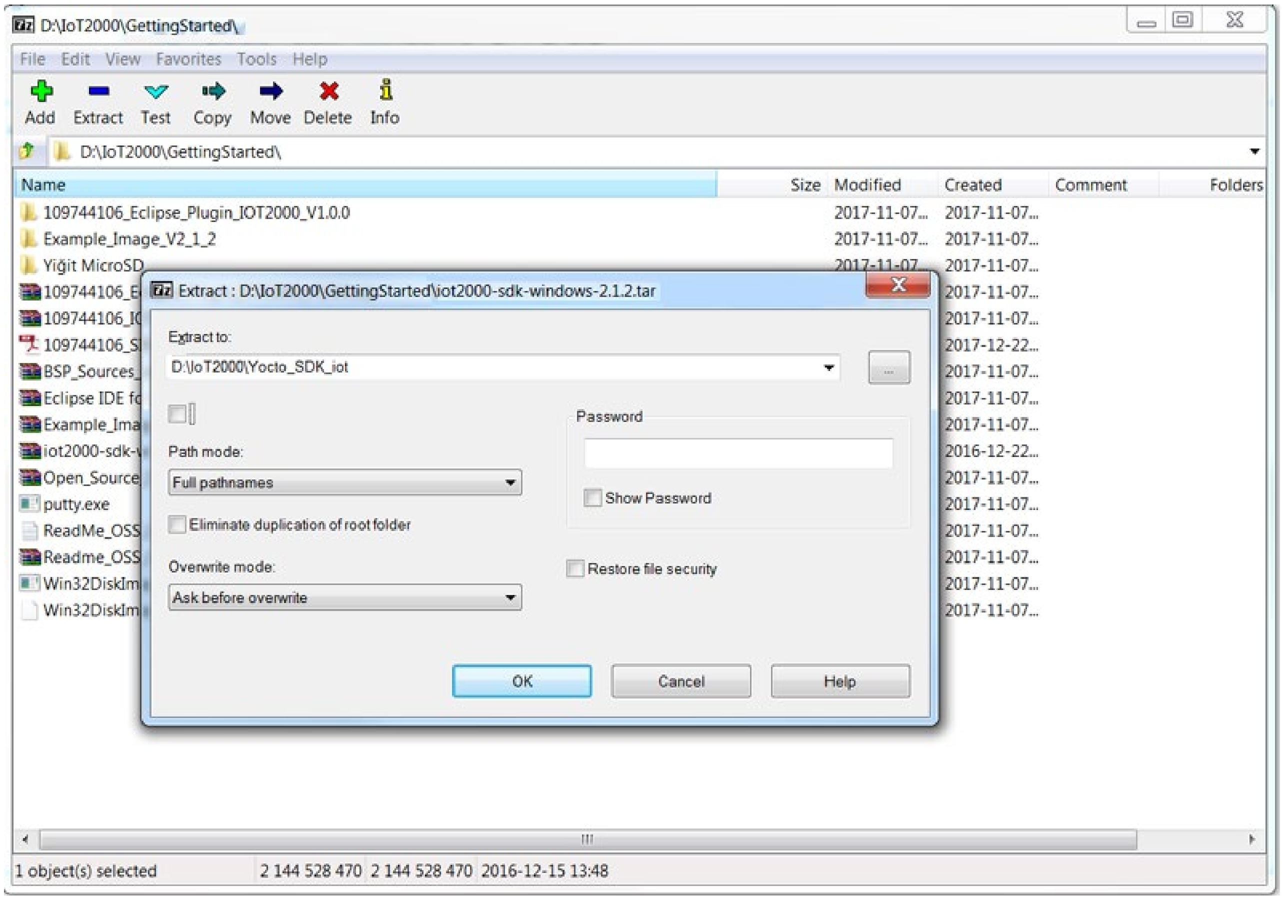
It is also possible to create a cross compiler for Yocto.

10.3.2 SDK installation in Windows

The IOT2000 SDK version must match the version of the example image and only works with a Windows operating system. You can download the plug-in under the following link: SIMATIC IOT2000 Eclipse Plug-in (https://support.industry.siemens.com/cs/document/109744106/simatic-iot2000-eclipse-plugin?dti=0&lc=en-WW)

To install the SDK, proceed as follows:

1. Start 7-Zip as an administrator.
2. Unzip the downloaded file, "IOT2000_sdk_windows_2.1.2.zip".
   Note that a newer version of the SDK may already be available since creation of this manual.
   The "iot2000_sdk_windows.tar" file is unzipped.
3. Unzip the file "IOT2000_sdk_windows.tar".


D:\IoT2000\GettingStarted\
File Edit View Favorites Tools Help
Add Extract Test Copy Move Delete Info

D:\IoT2000\GettingStarted\

Name	Size	Modified	Created	Comment	Folders
109744106_Eclipse_Plugin_IOT2000_V1.0.0		2017-11-07...	2017-11-07...		
Example_Image_V2_1_2		2017-11-07...	2017-11-07...		
Yiğit MicroSD		2017-11-07...	2017-11-07...		
109744106_E			2017-11-07...		
109744106_I			2017-11-07...		
109744106_S			2017-12-22...		
BSP_Sources_			2017-11-07...		
Eclipse IDE fo			2017-11-07...		
Example_Ima			2017-11-07...		
iot2000-sdk-w			2016-12-22...		
Open_Source			2017-11-07...		
putty.exe			2017-11-07...		
ReadMe_OSS			2017-11-07...		
Readme_OSS			2017-11-07...		
Win32DiskIm			2017-11-07...		
Win32DiskIm			2017-11-07...		

Extract : D:\IoT2000\GettingStarted\iot2000-sdk-windows-2.1.2.tar

Extract to: D:\IoT2000\Yocto_SDK_iot

Path mode: Full pathnames

Eliminate duplication of root folder

Overwrite mode: Ask before overwrite

Password

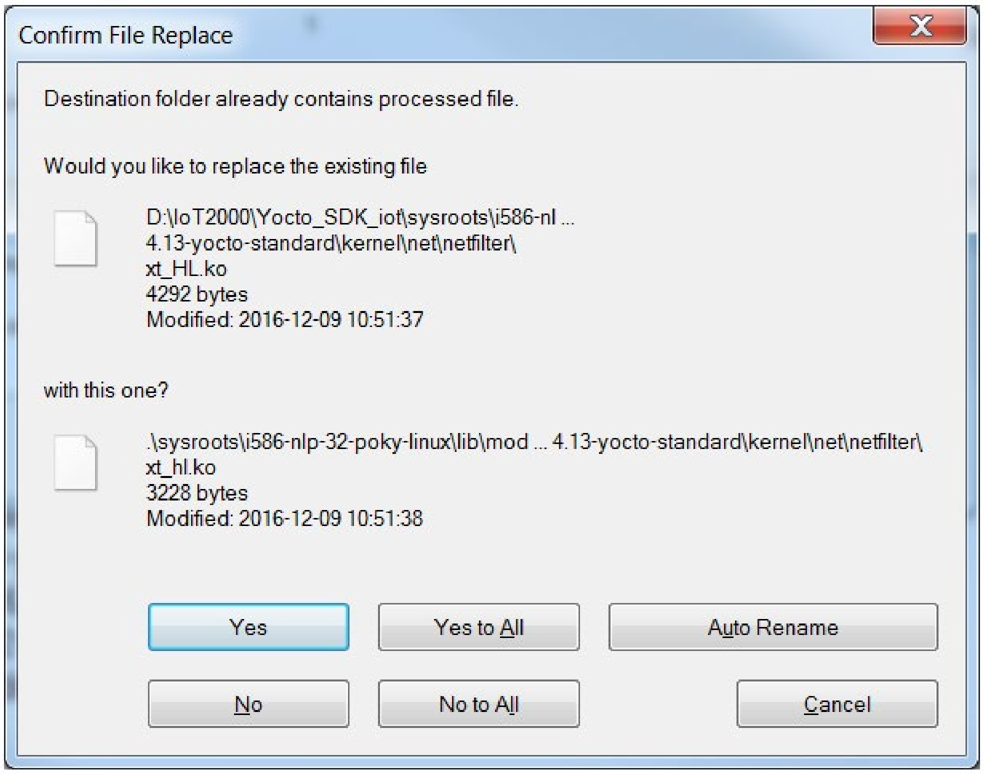
Show Password

Restore file security

OK Cancel Help

1 object(s) selected 2 144 528 470 2 144 528 470 2016-12-15 13:48

A dialog opens asking you whether you want to overwrite the files specific to IOT2000 SDK.


Confirm File Replace

Destination folder already contains processed file.

Would you like to replace the existing file

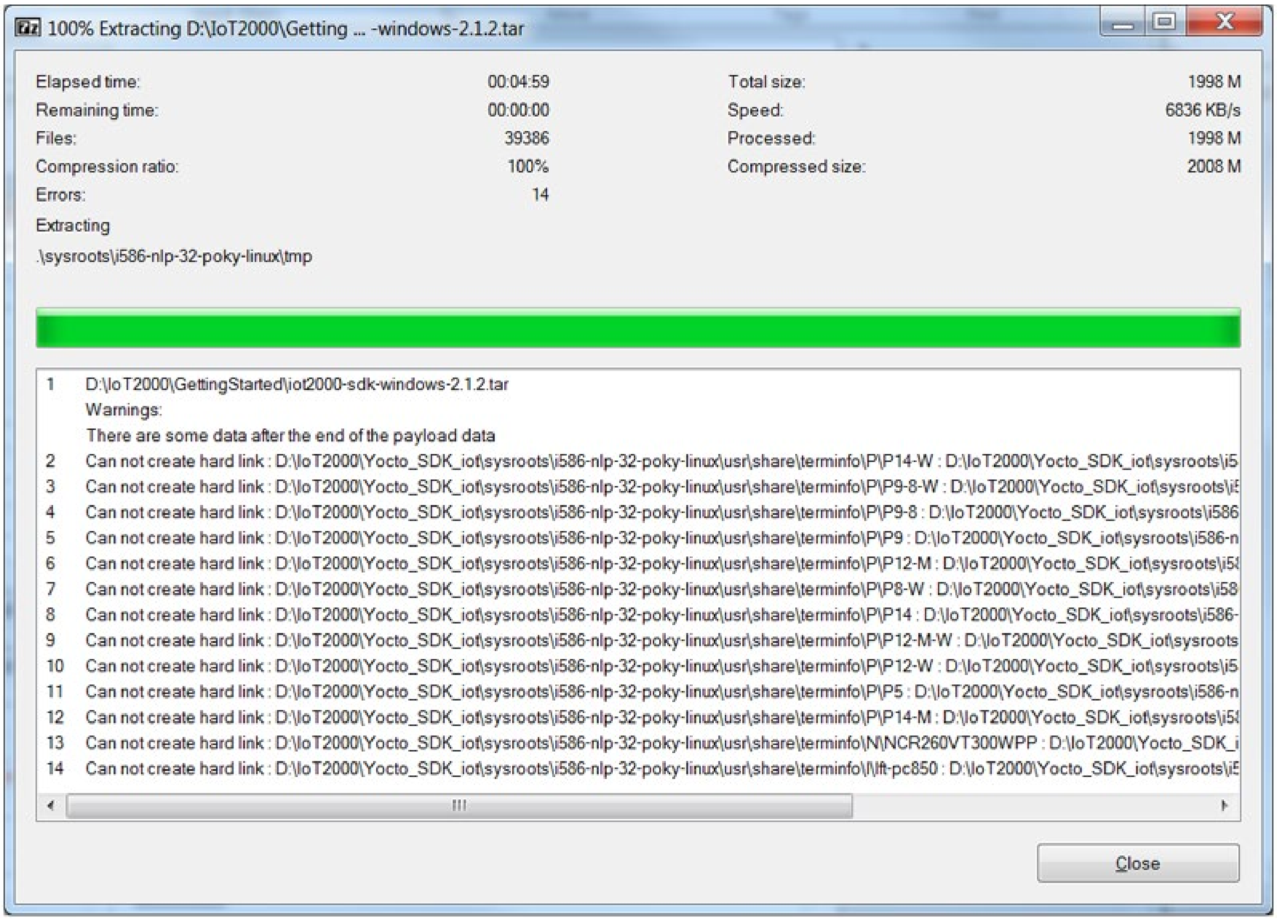
D:\IoT2000\Yocto_SDK_iot\sysroots\i586-nl ...
4.13-yocto-standard\kernel\net\netfilter\
xt_HL.ko
4292 bytes
Modified: 2016-12-09 10:51:37

with this one?

.\sysroots\i586-nlp-32-poky-linux\lib\mod ... 4.13-yocto-standard\kernel\net\netfilter\
xt_hl.ko
3228 bytes
Modified: 2016-12-09 10:51:38

4. Click on the "Yes, all" button to confirm the dialog.

5. If the "Cannot create hard link" dialog appears during unpacking, click the "Ignore" button.



```
Elapsed time:          00:04:59        Total size:          1998 M
Remaining time:        00:00:00        Speed:            6836 KB/s
Files:                    39386        Processed:           1998 M
Compression ratio:         100%        Compressed size:     2008 M
Errors:                      14
Extracting
.\sysroots\i586-nlp-32-poky-linux\tmp
```

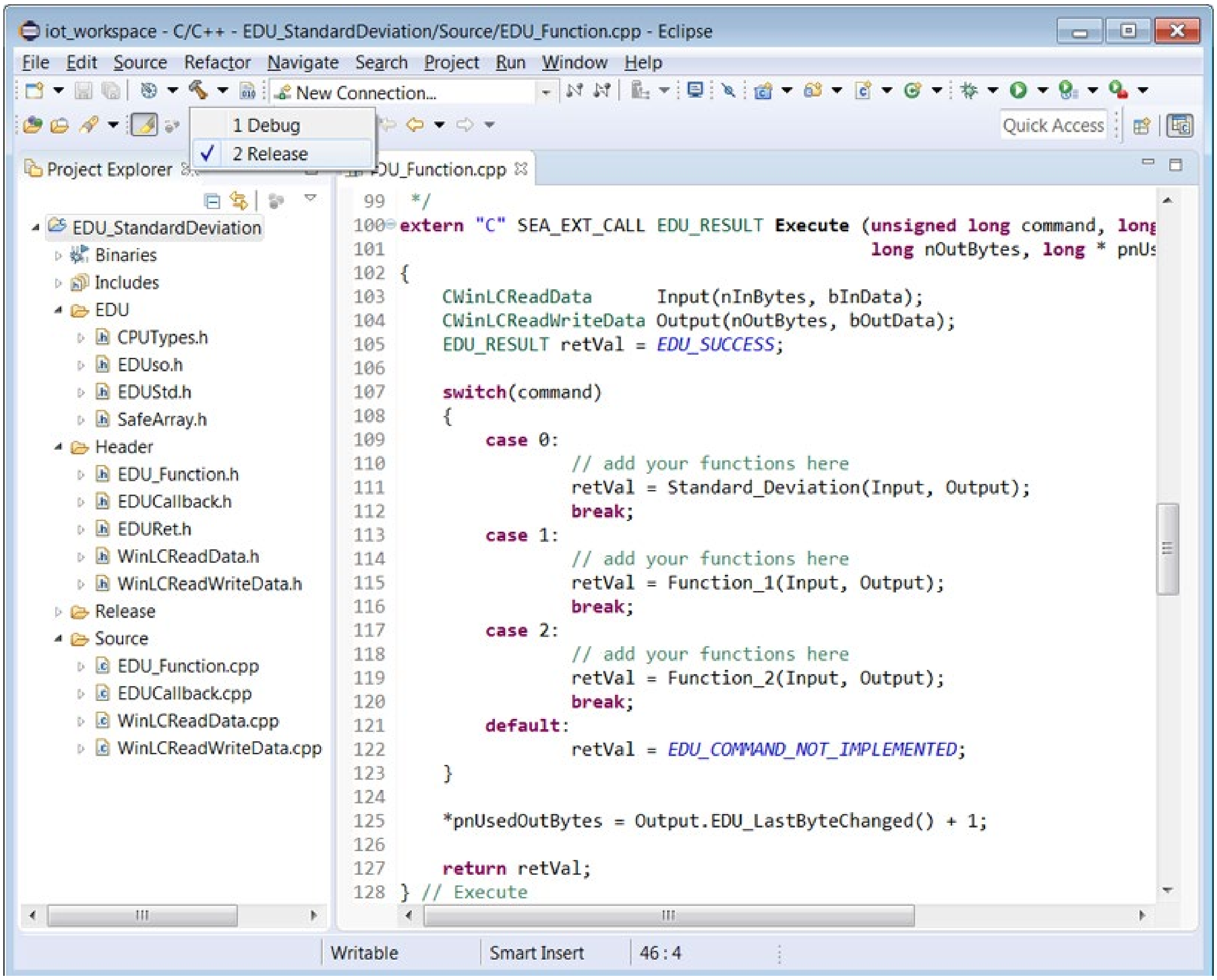
```
1    D:\IoT2000\GettingStarted\iot2000-sdk-windows-2.1.2.tar
     Warnings:
     There are some data after the end of the payload data
2    Can not create hard link : D:\IoT2000\Yocto_SDK_iot\sysroots\i586-nlp-32-poky-linux\usr\share\terminfo\P\P14-W : D:\IoT2000\Yocto_SDK_iot\sysroots\i5
3    Can not create hard link : D:\IoT2000\Yocto_SDK_iot\sysroots\i586-nlp-32-poky-linux\usr\share\terminfo\P\P9-8-W : D:\IoT2000\Yocto_SDK_iot\sysroots\i
4    Can not create hard link : D:\IoT2000\Yocto_SDK_iot\sysroots\i586-nlp-32-poky-linux\usr\share\terminfo\P\P9-8 : D:\IoT2000\Yocto_SDK_iot\sysroots\i586
5    Can not create hard link : D:\IoT2000\Yocto_SDK_iot\sysroots\i586-nlp-32-poky-linux\usr\share\terminfo\P\P9 : D:\IoT2000\Yocto_SDK_iot\sysroots\i586-n
6    Can not create hard link : D:\IoT2000\Yocto_SDK_iot\sysroots\i586-nlp-32-poky-linux\usr\share\terminfo\P\P12-M : D:\IoT2000\Yocto_SDK_iot\sysroots\i5
7    Can not create hard link : D:\IoT2000\Yocto_SDK_iot\sysroots\i586-nlp-32-poky-linux\usr\share\terminfo\P\P8-W : D:\IoT2000\Yocto_SDK_iot\sysroots\i58
8    Can not create hard link : D:\IoT2000\Yocto_SDK_iot\sysroots\i586-nlp-32-poky-linux\usr\share\terminfo\P\P14 : D:\IoT2000\Yocto_SDK_iot\sysroots\i586-
9    Can not create hard link : D:\IoT2000\Yocto_SDK_iot\sysroots\i586-nlp-32-poky-linux\usr\share\terminfo\P\P12-M-W : D:\IoT2000\Yocto_SDK_iot\sysroots
10   Can not create hard link : D:\IoT2000\Yocto_SDK_iot\sysroots\i586-nlp-32-poky-linux\usr\share\terminfo\P\P12-W : D:\IoT2000\Yocto_SDK_iot\sysroots\i5
11   Can not create hard link : D:\IoT2000\Yocto_SDK_iot\sysroots\i586-nlp-32-poky-linux\usr\share\terminfo\P\P5 : D:\IoT2000\Yocto_SDK_iot\sysroots\i586-n
12   Can not create hard link : D:\IoT2000\Yocto_SDK_iot\sysroots\i586-nlp-32-poky-linux\usr\share\terminfo\P\P14-M : D:\IoT2000\Yocto_SDK_iot\sysroots\i5
13   Can not create hard link : D:\IoT2000\Yocto_SDK_iot\sysroots\i586-nlp-32-poky-linux\usr\share\terminfo\N\NCR260VT300WPP : D:\IoT2000\Yocto_SDK_i
14   Can not create hard link : D:\IoT2000\Yocto_SDK_iot\sysroots\i586-nlp-32-poky-linux\usr\share\terminfo\l\lft-pc850 : D:\IoT2000\Yocto_SDK_iot\sysroots\i5
```

6. Install IOT2000 SDK under the following path:
   "D:\IoT2000\Yocto_SDK_iot"

Once you have installed the SDK, you can compile or further modify the EDU_StandardDeviation example project. Alternatively, you can also create your own Eclipse project.

10.3.3 Compile the SO file with the example project

The project files and folder structures are displayed in the following figure:



```
 */
extern "C" SEA_EXT_CALL EDU_RESULT Execute (unsigned long command, long
                                            long nOutBytes, long * pnUs
{
    CWinLCReadData      Input(nInBytes, bInData);
    CWinLCReadWriteData Output(nOutBytes, bOutData);
    EDU_RESULT retVal = EDU_SUCCESS;

    switch(command)
    {
        case 0:
                // add your functions here
                retVal = Standard_Deviation(Input, Output);
                break;
        case 1:
                // add your functions here
                retVal = Function_1(Input, Output);
                break;
        case 2:
                // add your functions here
                retVal = Function_2(Input, Output);
                break;
        default:
                retVal = EDU_COMMAND_NOT_IMPLEMENTED;
    }

    *pnUsedOutBytes = Output.EDU_LastByteChanged() + 1;

    return retVal;
} // Execute
```

You can write your software in the files "EDU_Function.cpp" and "EDU_Function.h". These files contain all interface functions of the CPU function library.

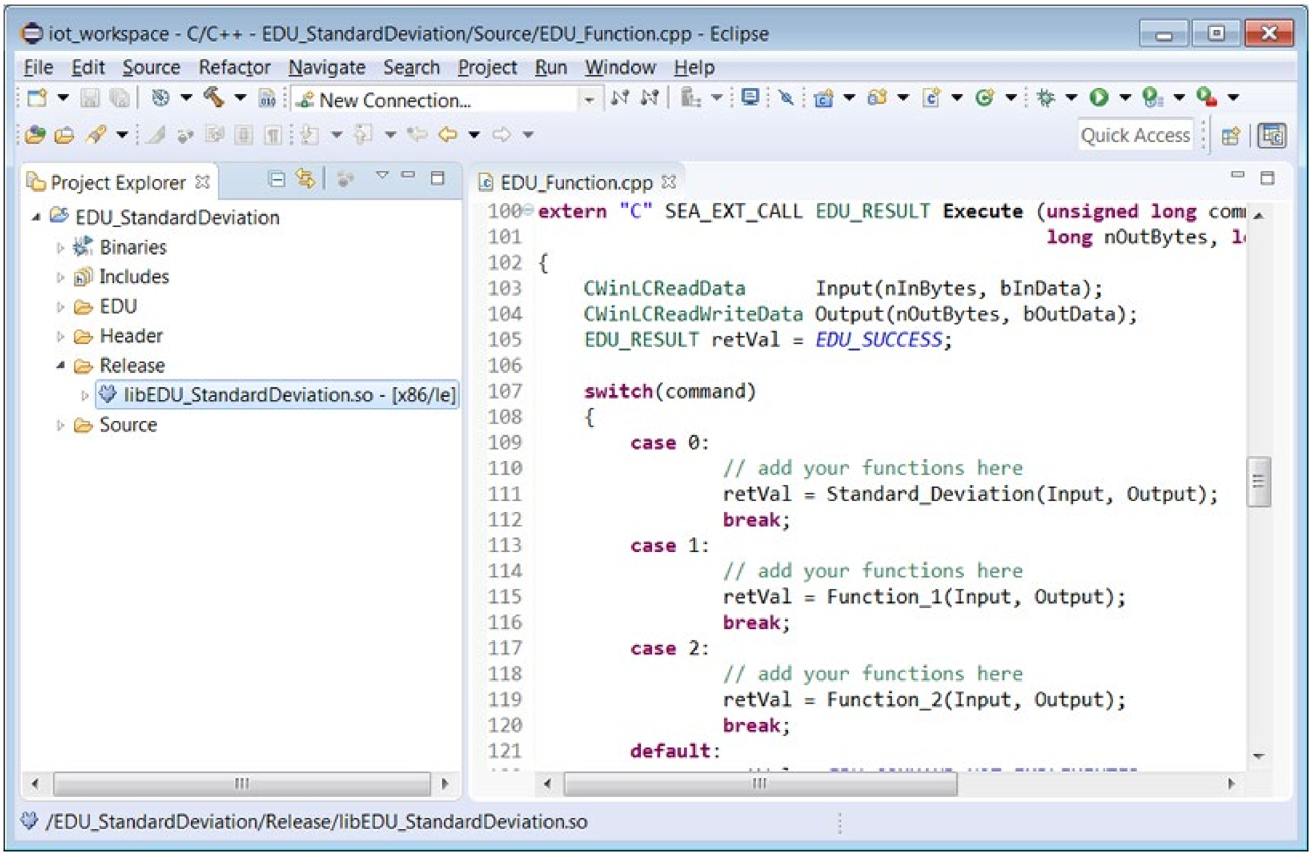
The "Execute" function is activated every time SFB65002 (EXEC_COM) is called. The EXEC_COM instruction in the TIA Portal contains the "Command" parameter. This corresponds to the "switch case" condition in the SO file.

Add your own "Case x" status using the custom functions. Function prototypes must be created in the EDU_Function.h file. The example function "Standard_Deviation (Input, Output)" is placed under the "Case 0" status. This means that if EXEC_COM is called with the "0" command, this involves "Case 0" and the "Standard_Deviation" function is executed. If you are creating your own project on any operating system or IDE, you need to add the following parameters:

`_M_IX86`	Defined symbol or pre-processors
`RELEASE`	
`Position Independent Code (-fPIC)`	
`-c -m32 -march=i586`	Compiler flags
`"${workspace:/${ProjName}/EDU}"`	Including directories
`"${workspace:/${ProjName}/Header}"`	

Compile the project after setting the required compile and link parameters.

The SO file "libEDU_StandardDeviation.so" is generated.


iot_workspace - C/C++ - EDU_StandardDeviation/Source/EDU_Function.cpp - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
New Connection...
Quick Access
Project Explorer
EDU_StandardDeviation
Binaries
Includes
EDU
Header
Release
libEDU_StandardDeviation.so - [x86/le]
Source
EDU_Function.cpp
/EDU_StandardDeviation/Release/libEDU_StandardDeviation.so

```
extern "C" SEA_EXT_CALL EDU_RESULT Execute (unsigned long com
                                           long nOutBytes, l
{
    CWinLCReadData      Input(nInBytes, bInData);
    CWinLCReadWriteData Output(nOutBytes, bOutData);
    EDU_RESULT retVal = EDU_SUCCESS;

    switch(command)
    {
        case 0:
                // add your functions here
                retVal = Standard_Deviation(Input, Output);
                break;
        case 1:
                // add your functions here
                retVal = Function_1(Input, Output);
                break;
        case 2:
                // add your functions here
                retVal = Function_2(Input, Output);
                break;
        default:
```

10.3.4 Cross compiling with the IOT2000 SDK installation

To create a new Eclipse project, you need to specify a cross compiler directory and set compiler flags. You must also copy the following folders of the example project to the directory of the new project:

- Header
- EDU
- Source

Enter the following compiler options in the properties of the new project:

Table 10- 3 C/C++ Build

Property			**Compiler option**	**Comment**
Builder type:			Internal builder	
Environ-ment:	POKY_HOME:		D:\IoT2000\Yocto_SDK_iot\sysroots\i586-nlp-32-poky-linux	
	PATH:		D:\IoT2000\Yocto_SDK_iot\sysroots\i686-pokysdk-mingw32\usr\bin\i586-poky-linux	Insert this path at the end of the PATH variable.
Current toolchain:			Cross GCC	
Settings:	Cross GCC Compiler:	Com-mand:	i586-poky-linux-gcc	Enter the same settings as for a Cross G++ Compiler.
	Cross G++ Compiler:	Com-mand:	i586-poky-linux-g++	
		Defined symbols (-D):	RELEASE, _M_IX86	
		In-cludes:	"${ workspace:/${ProjName}/Header}" "${ workspace:/${ProjName}/EDU}" "${POKY_HOME}\usr\include" "${POKY_HOME}\usr\include\c++\5.3.0" "${POKY_HOME}\usr\include\c++\5.3.0\i586-poky-linux" "${POKY_HOME}\usr\include\mraa" "${POKY_HOME}\usr\include\upm"	
		Miscel-laneous:	-O0 -g3 -Wall -c -fmessage-length=0 -m32 -march=i586 -c -ffunction-sections -fdata-sections	
			Position Independent Code (-fPIC)	Enable this option.
	Cross G++ linker	Com-mand:	i586-poky-linux-g++	
		Library search path (-L):	"${POKY_HOME}"	
		Miscel-laneous:	-m32 -fno-use-linker-plugin --sys-root=D:\\IoT2000\\Yocto_SDK_iot\\sysroots\\i586-nlp-32-poky-linux	
	Cross GCC Assembler	Com-mand:	i586-poky-linux-as	

These options have already been created for the example project.

Note

Note that you should use your specific project name and SDK path, e.g. with "${ workspace:/${ProjName}, POKY_HOME, PATH and --systemroot.

10.3.5 Loading an SO file to IOT20X0

Depending on the development environment you used to compile the SO file, you can load the SO file onto the IOT20X0 device in one of three ways.

Note

The following description is based on the example SO file "libEDU_StandardDeviation.so".

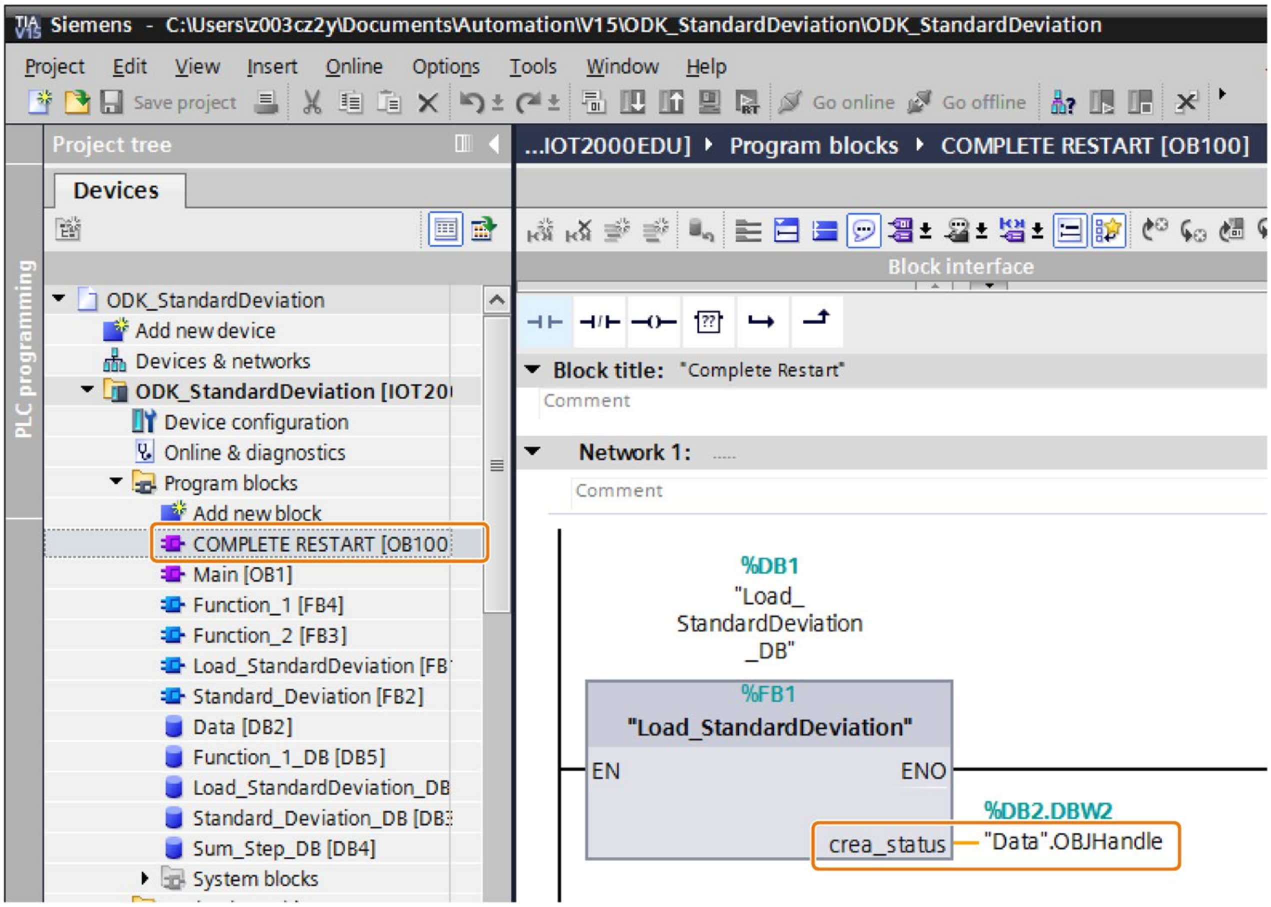
- You are using a Windows operating system and have used the cross compiler:
  - Transfer the "libEDU_StandardDeviation.so" file using the "pscp.exe" program. You can download it free of charge from the Internet.
  - Open a Windows command line and start pscp.exe with the command "-scp":
    For example: `pscp.exe -scp ..\EDU_StandardDeviation (Eclipse)\Release\libEDU_StandardDeviation.so root@192.169.200.1:/home/root/`
  - In this example, `root@192.168.200.1:/home/root/` is used for the IOT2000 device IP and for the directory. You can move the SO to any directory.
- You have compiled your software on a Linux operating system:
  - Use the command "scp" in a terminal.
  - For example: `sudo scp /home/Alex/EDU_StandardDeviation (Eclipse)\Release\libEDU_StandardDeviation.so root@192.168.200.1:/home/root/`
- You can also copy the SO to a USB flash drive:
  - When you connect the USB flash drive to the IOT20x0 device, a device name is displayed under `/dev folder`. The USB flash drive can then be mounted in any temporary directory. You can now copy the SO.
    For example:

```
Alex@ubuntu:~$ mount /dev/sdx /media (sdx: may be sdb, sdc …)

Alex@ubuntu:~$ cp /media/ libEDU_StandardDeviation.so /home/Alex/
```

10.4 Preparing the TIA Portal project for the application

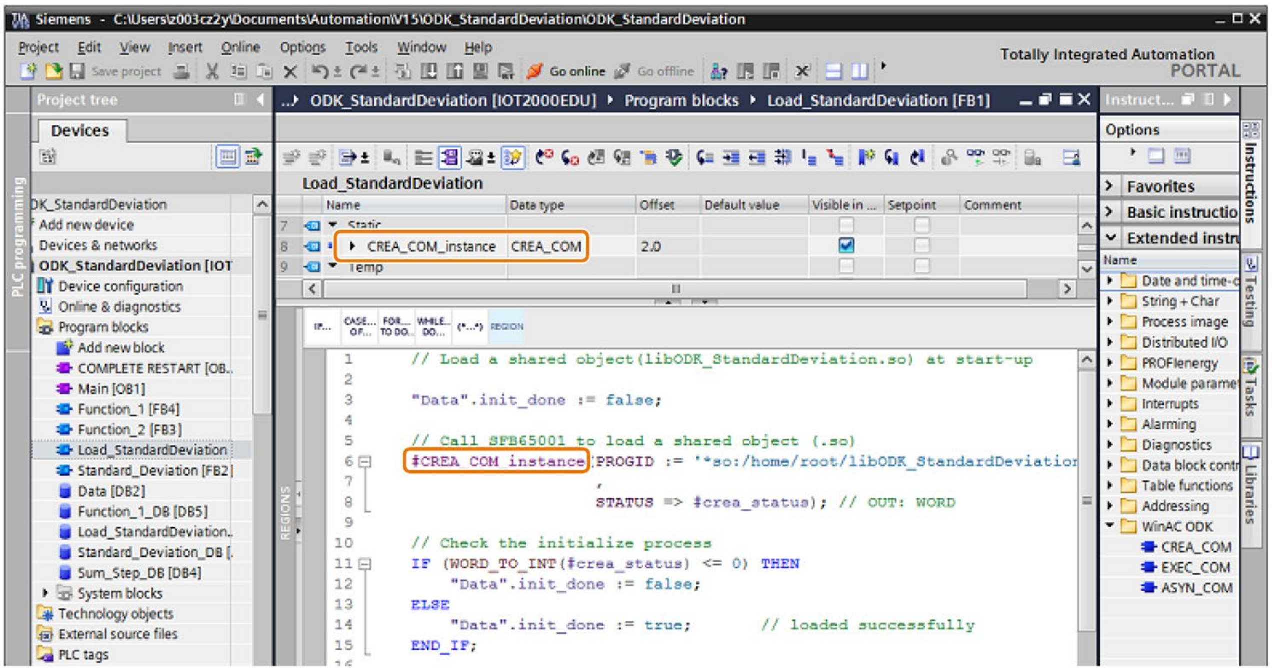
To load and execute an SO, you must call the instructions with the correct parameters in the TIA Portal project:

1. Ensure that your SO file has been loaded on the IOT2000 device.
   For the IOT2000EDU to be able to an SO, the CREA_COM instruction is called with two parameters:
   - PROGID: Points to the object name with the full directory path. This parameter has the "string" type.
     '*so:/home/root/libEDU_StandardDeviation.so'
     The *.so prefix must be written before the object name pointing to the *.so file.
   - STATUS: Returns a handle (OBJHandle) or an error code. If the return value is between 0x0001 and 0x7FFF, it is an object handle. If it is between 0x8001 and 0x810C, it is an error code.
2. In general, instructions are created in a function block, in this case "Load_StandardDeviation[FB1]". To load the SO, an OB then calls the instructions, e.g. OB100.



3. Drag the instructions directly from the "Advanced instructions" palette under the "Instructions" task card into the programming editor.

4. If required, you can also change the static name of the instructions in the module interface, e.g. "CREA_COM_instance" and "EXEC_COM_instance".

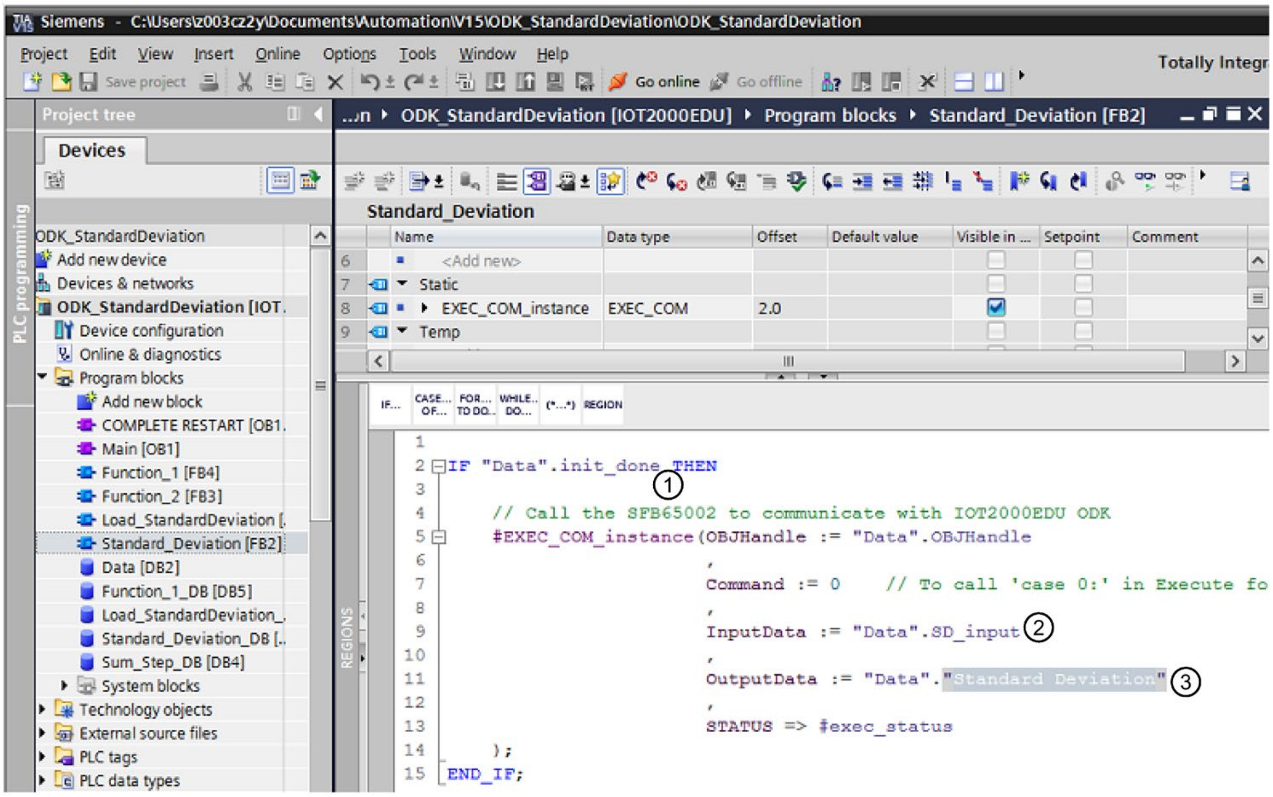


Load_StandardDeviation

	Name	Data type	Offset	Default value	Visible in ...	Setpoint	Comment
7	▼ Static				☐	☐	
8	▸ CREA_COM_instance	CREA_COM	2.0		☑	☐	
9	▼ Temp				☐	☐	

```
// Load a shared object(libODK_StandardDeviation.so) at start-up

"Data".init_done := false;

// Call SFB65001 to load a shared object (.so)
#CREA_COM_instance(PROGID := '*so:/home/root/libODK_StandardDeviatio
                   ,
                   STATUS => #crea_status); // OUT: WORD

// Check the initialize process
IF (WORD_TO_INT(#crea_status) <= 0) THEN
    "Data".init_done := false;
ELSE
    "Data".init_done := true;       // loaded successfully
END_IF;
```

5. The "EXEC_COM" instruction calls the "Execute" function of the SO. This function is specified by the OBJHandle parameter.
   The OBJHandle is the return value of the "CREA_COM" instruction.
   If you want to load two or more shared libraries, the OBJHandle is used to distinguish between the shared libraries.
   The "EXEC_COM" instruction is inserted into a new function block, in this case Standard_Deviation[FB2], and is called Main[OB1].
6. The "Command" parameter is also important. The "Command" parameter specifies a special switch case status in the "Execute" function of the SO.
   In this example, the "Standard_Deviation(Input,Output)" function is placed below "case 0". Therefore, "Command" must have the "0" value for the function to be executed.



	Name	Data type	Offset	Default value	Visible in ...	Setpoint	Comment
6	■ <Add new>				☐	☐	
7	▼ Static				☐	☐	
8	■ ▸ EXEC_COM_instance	EXEC_COM	2.0		✓	☐	
9	▼ Temp				☐	☐	

```
IF "Data".init_done THEN

    // Call the SFB65002 to communicate with IOT2000EDU ODK
    #EXEC_COM_instance(OBJHandle := "Data".OBJHandle
                          ,
                          Command := 0    // To call 'case 0:' in Execute fo
                          ,
                          InputData := "Data".SD_input
                          ,
                          OutputData := "Data"."Standard Deviation"
                          ,
                          STATUS => #exec_status
    );
END_IF;
```

①	"Data"."init_done"	If the SO was successfully loaded, this value is "TRUE".
②	"Data".SD_input	A set of numbers placed in "Data[DB2]". You can increase the size or change the numbers if this requires a different standard deviation calculation.
③	"Data"."Standard Deviation"	Output of the calculation.

Editing input and output data in the SO file

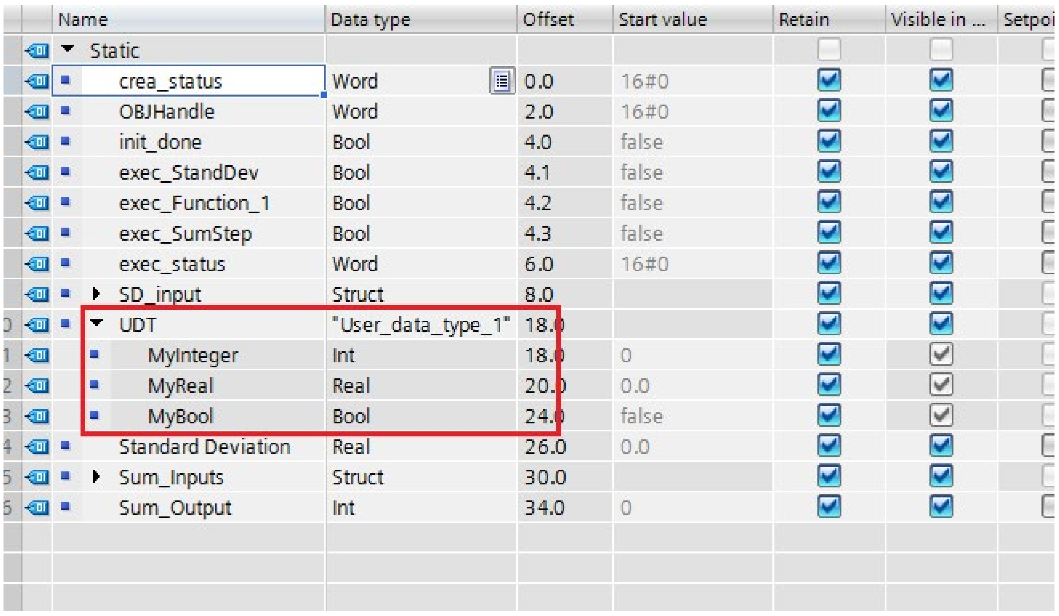
Each function in the switch case has two input values, are "objects":

- Input data (`rInput`)
- Output data (`rOutput`)

To get variables out of these objects, use the data types supported by the helper classes, e.g. "EDU_ReadS7INT".

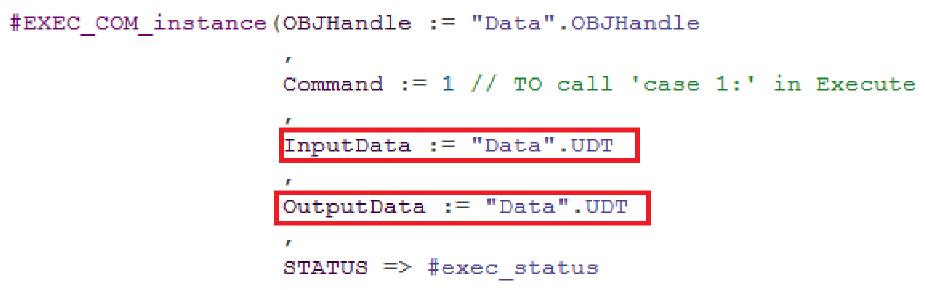
Example:

You first need to define all structures, e.g. "UDT", in the "Data" data block in the TIA Portal.



Name	Data type	Offset	Start value	Retain	Visible in ...	Setpoi
Static				☐	☐	☐
crea_status	Word	0.0	16#0	✓	✓	☐
OBJHandle	Word	2.0	16#0	✓	✓	☐
init_done	Bool	4.0	false	✓	✓	☐
exec_StandDev	Bool	4.1	false	✓	✓	☐
exec_Function_1	Bool	4.2	false	✓	✓	☐
exec_SumStep	Bool	4.3	false	✓	✓	☐
exec_status	Word	6.0	16#0	✓	✓	☐
SD_input	Struct	8.0		✓	✓	☐
UDT	"User_data_type_1"	18.0		✓	✓	☐
MyInteger	Int	18.0	0	✓	✓	☐
MyReal	Real	20.0	0.0	✓	✓	☐
MyBool	Bool	24.0	false	✓	✓	☐
Standard Deviation	Real	26.0	0.0	✓	✓	☐
Sum_Inputs	Struct	30.0		✓	✓	☐
Sum_Output	Int	34.0	0	✓	✓	☐

This structure can be both an input as well as an output value for the "EXEC_COM" instruction in the TIA Portal.



```
#EXEC_COM_instance(OBJHandle := "Data".OBJHandle
                   ,
                   Command := 1 // TO call 'case 1:' in Execute
                   ,
                   InputData := "Data".UDT
                   ,
                   OutputData := "Data".UDT
                   ,
                   STATUS => #exec_status
```

"InputData := "Data".UDT" in the TIA Portal corresponds to `rInput` in the SO file. All struct variables must continue to be taken individually from the `rInput`.

For example: the struct variable "MyBool" in UDT. This variable was read from the SO file using `rInput.EDU_ReadS7Bool(6,0,Mybool);`.

10.4.1 Reference of the IOT2000EDU function library Instructions

The IOT2000EDU function library provides two instructions in the TIA Portal:

- CREA_COM (SFB65001)
- EXEC_COM (SFB65002)

10.4.1.1 CREA_COM (SFB65001)

The "CREA_COM" instruction loads an instance of the SO specified by the PROGID parameter. The IOT2000EDU program assigns the InstanceID parameter.

The following table shows the interfaces of the "CREA_COM" instruction:

Address	Declaration	Name	Data type	Comment
0.0	In	PROGID	STRING[254]	ID of the SO to be loaded
256	Out	Status	WORD	SFB return code: Error code or OBJHandle code

The PROGID is a string containing the file name and storage path of the SO.

For example: `*so:/home/User_name/Desktop/Final_Project/CalculateStandartDeviation.so`

The SO name is "CalculateStandartDeviation.so".

The "CREA_COM" instruction evaluates the input conditions and performs the following actions:

1. If the SO has not yet been loaded, "CREA_COM" calls the "ODKCreate" function to create the SO. To create a ClassID, the InstanceID parameter is used. "CREA_COM" creates only one instance of this object. "CREA_COM" adds the object instance to the internal list of created IOT2000EDU objects.
2. If the SO has already been created, "CREA_COM" retains the IOT2000EDU function block handle for previously created objects (an index to locate the Object Pointer).
3. If this is the first call to "CREA_COM" after exiting STOP mode, or if the SO has just been created, "CREA_COM" calls the Active function.
4. "CREA_COM" sets the status parameter for the IOT2000EDU function block handle (or error code) and sets the BR bit.

Return codes

Return code	Alarm	Meaning
0x0001 - 0x7FFF	OBJ_HANDLE	Returned "object handle".
0x807F	ERROR_INTERNAL	Internal error.
0x8001	E_EXCEPTION	An exception has occurred.
0x8102	E_CLSID_FAILED	Call of CLSIDFromProgID failed.
0x8103	E_COINITIALIZE_FAILED	Call of CoInitializeEd failed.
0x8104	E_CREATE_INSTANCE_FAILED	Call of CoCreateInstance failed.
0x8105	E_LOAD_LIBRARY_FAILED	The library was not loaded.
0x8106	E_NT_RESPONSE_TIMEOUT	A response timeout has occurred in Windows.
0x8107	E_INVALID_OB_STATE	The controller is in an invalid operating state for planning an OB.
0x8108	E_INVALID_OB_SCHEDULE	Schedule information for the OB is invalid.
0x8109	E_INVALID_INSTANCEID	Instance ID for calling the "CREA_COM" instruction is invalid.
0x810A	E_START_ODKPROXY_FAILED	The controller could not load the proxy DLL.
0x810B	E_CREATE_SHAREMEM_FAILED	The controller could not create or initialize a shared memory area.
0x810C	E_OPTION_NOT_AVAILABLE	An attempt has been made to access an unavailable option.

10.4.1.2 EXEC_COM (SFB65002)

The "EXEC_COM" instruction calls the "Execute" function of the SO specified by the OBJHandle parameter.

The following table shows the interfaces of the "EXEC_COM" instruction:

Address	Declaration	Name	Data type	Comment
0.0	In	OBJHandle	WORD	Handle returned by "CREA_COM".
2.0	In	Command	DWORD	Index of the function or command to be executed
6.0	In	InputData	ANY	Pointer to function input area
16.0	In	OutputData	ANY	Pointer to function output area
26.0	Out	Status	WORD	Instruction error code or return code from the "Execute" function

The "EXEC_COM" instruction:

1. Verifies that "CREA_COM" has been called and that the object handle is valid.
2. Processes the ANY pointers and returns the error codes for invalid ANY pointer parameters.
3. Calls the "Execute" function of the SO.

4. Assigns the input and output pointer areas to the IOT2000EDU Data Access Helper classes.
5. Sets the status parameter for the "Execute" return code, unless an error has occurred beforehand, and returns to the IOT2000EDU program.

Note

Note that the runtime of the called function adds to the cycle time.

Return codes

Return code	Alarm	Meaning
0x0000	NO_ERRORS	No errors
0x807F	ERRORS_INTERNAL	Internal error.
0x8001	E_EXCEPTION	An exception has occurred.
0x8002	E_NO_VALID_INPUT	Input: The ANY pointer is invalid.
0x8003	E_INPUT_RANGE_INVALID	Input: The ANY pointer range is invalid.
0x8005	E_NO_VALID_INPUT	Output: The ANY pointer is invalid.
0x8005	E_OUTPUT_RANGE_INVALID	Output: The ANY pointer range is invalid.
0x8006	E_OUTPUT_OVERFLOW	More bytes were written to the output buffer by the shared object than were assigned.
0x8007	E_NOT_INITIALIZED	Function library system was not initialized: No call of the "CREA_COM" instruction yet.
0x8008	E_HANDLE_OUT_OF_RANGE	The assigned handle value does not match any valid shared object.
0x8009	E_INPUT_OVERFLOW	More bytes were written to the input buffer by the shared object than were assigned.

10.5 Creating function library programs with Matlab Simulink

Mathworks MATLAB is software for the primary solution of mathematical problems and their visualization.

Simulink is an add-on for MATLAB for graphical modeling of systems and their simulation.

Simulink Coder Addon enables you to compile C/C++ code directly from a Simulink model. The IOT2000EDU function library allows you to run C/C++ code in IOT20x0 devices.

Note

The following screenshots of the code examples may differ from the delivered product.

10.5.1 Requirements

You need the following software:

- TIA Portal V15
- Matlab 2017b in the following configuration:
  - Matlab 9.3
  - Matlab Coder 3.4
  - Embedded Coder 6.13
  - Simulink 9.0
  - Simulink Coder 8.13
- Eclipse Kepler
- IOT2000 SDK

10.5.2 Creating a Simulink model

Requirement

MATLAB R2017b is open.

Procedure

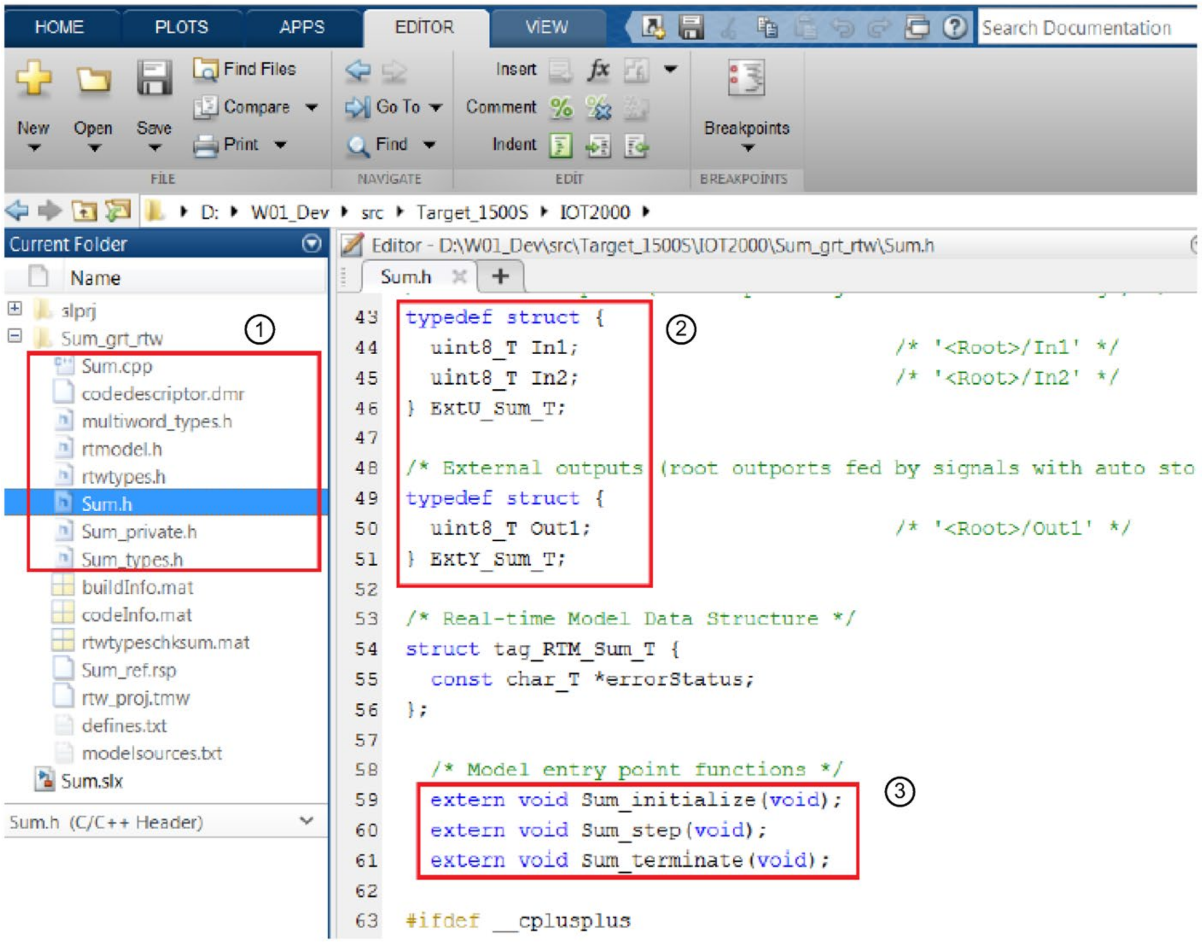
The procedure is described below using the example of a simple mathematical operation.

1. To create a new Simulink model, click the Simulink icon on the menu bar.
2. Drag-and-drop the following blocks from the Simulink Library Browser :
   - Input: under "Simulink > Source > In1"
   - Input: under "Simulink > Source > In2"
   - Output: under "Simulink > Source > Out1"
   - Adding block: under "Simulink > Math Operation > Add"
3. To open the respective block parameters for an input, double-click the corresponding input block in the Simulink model.
4. Change the "Data type" parameter to "uint8" for both input blocks in the "Block Parameter" window under "Signal Attributes".
5. Adapt the Simulink parameters (Page 76).
6. Save the model.

   Use the name "Sum" for this example.
7. To start the build process in Simulink , select the "Code > C/C++ Code > Build Model" command in the menu bar.

Result

The system creates the "Sum_grt_rtw" folder and stores the generated *.cpp and *.h files in it. The file names are prefixed with the model name, in this case "Sum".



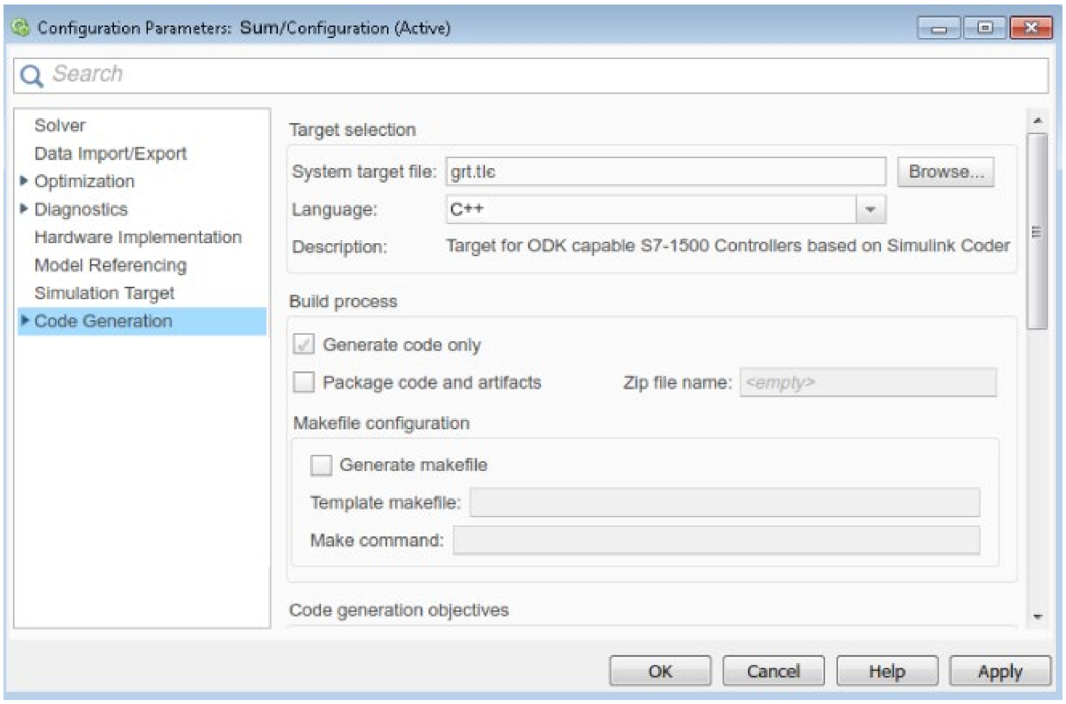
```
slprj
Sum_grt_rtw
    Sum.cpp
    codedescriptor.dmr
    multiword_types.h
    rtmodel.h
    rtwtypes.h
    Sum.h
    Sum_private.h
    Sum_types.h
    buildInfo.mat
    codeInfo.mat
    rtwtypeschksum.mat
    Sum_ref.rsp
    rtw_proj.tmw
    defines.txt
    modelsources.txt
Sum.slx
```

```
typedef struct {
  uint8_T In1;                         /* '<Root>/In1' */
  uint8_T In2;                         /* '<Root>/In2' */
} ExtU_Sum_T;

/* External outputs (root outports fed by signals with auto sto
typedef struct {
  uint8_T Out1;                        /* '<Root>/Out1' */
} ExtY_Sum_T;

/* Real-time Model Data Structure */
struct tag_RTM_Sum_T {
  const char_T *errorStatus;
};

  /* Model entry point functions */
  extern void Sum_initialize(void);
  extern void Sum_step(void);
  extern void Sum_terminate(void);

#ifdef __cplusplus
```

①	Generated files
②	Input and output parameters
③	Interface functions of the Sum model

10.5.3 Description of Simulink parameters

To adapt the Simulink parameters for the build process, open the "Configuration Parameters" with the icon in the menu bar.



System target file: grt.tlc
Language: C++
Description: Target for ODK capable S7-1500 Controllers based on Simulink Coder
[✓] Generate code only
[ ] Package code and artifacts Zip file name: <empty>
[ ] Generate makefile
Template makefile:
Make command:

Figure 10-1 Configuration parameters

Relevant parameters

Solver

A dynamic system is modeled as a mathematical calculation in Simulink. This calculation is performed at certain time intervals to simulate the execution of the system. The size of this time interval is referred to as the "Step-Size". The method for calculating the states of a model is referred to here as solving the model.

- Solver options

  Determine the solver selection

  – Type: Fixed Step

  – Solver: auto
- Fixed-step size: auto

Code Generation

- Target Selection:
  - System target file: grt.tlc (Create Visual C/C++ Solution File for Simulink Coder)
  - Language: C++
- Build process:
  - Generate code only: "Generate code only" option enabled
  - Generate makefile: "Generate makefile" option disabled
- Interface:
  - Code interface packaging: Nonreusable function
  - MAT-file logging: "MAT-file logging" option disabled
  - ASAP2 interface: "ASAP2 interface" option disabled
  - External mode: "External mode" option disabled

10.5.4 Integrating the Simulink model into Eclipse

To run the C/C++ code generated by the Simulink Coder on IOT20x0 devices, the code must be converted by the IOT2000EDU function library. Eclipse is available for this purpose.

Procedure

1. Start Eclipse.
2. Create the "Matlab" folder in "Project Explorer" in your Eclipse project, "EDU_StandardDeviation".
3. Copy the generated *.cpp and *.h files into the "Matlab" folder.
4. Open the EDU_Function.cpp in "Project Explorer" under "Source".
5. Insert the header file "Sum.h" into the EDU_Function.cpp to access the interface functions.

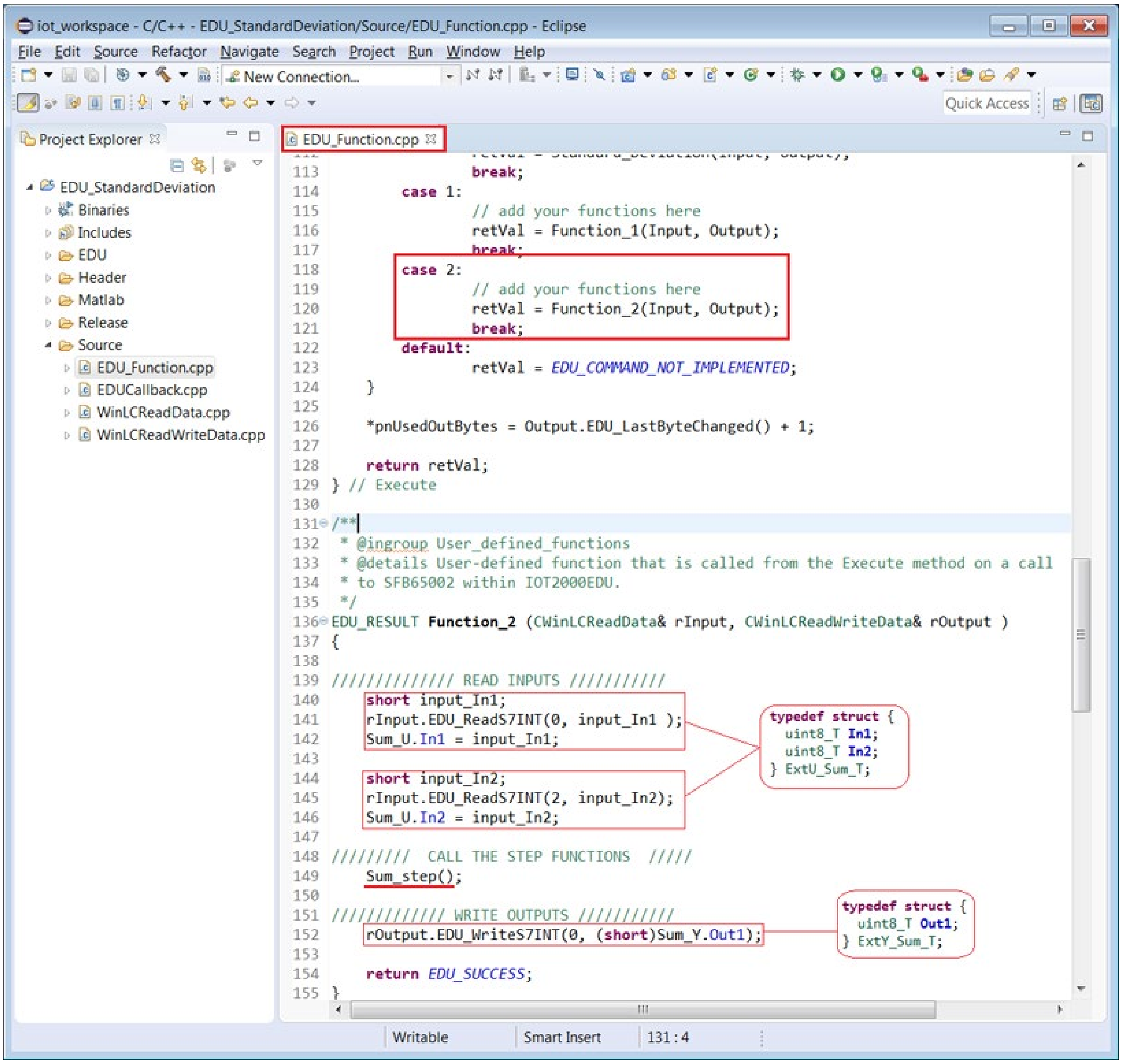
   Localize the three functions:

   - `Sum_initialize()` under `ODKCreate`

     ODKCreate is called when a shared object (*.so) is loaded.
   - `Sum_terminate()` under `ODKRelease`

     ODKRelease is called when a shutdown operation is performed by IOT2000EDU.
   - `Sum_step()` in each case instruction called by the IOT2000EDU function library using the instruction "EXEC_COM" under the Execute callback.

6. In the example project, "Function_2 (Input, Output)" is located under "case 2:" Statement. This function has input and output values, but `Sum_step` is lacking these values. Therefore, you must assign the input values before `Sum_step` and the output values afterwards, and then adapt them to each other:



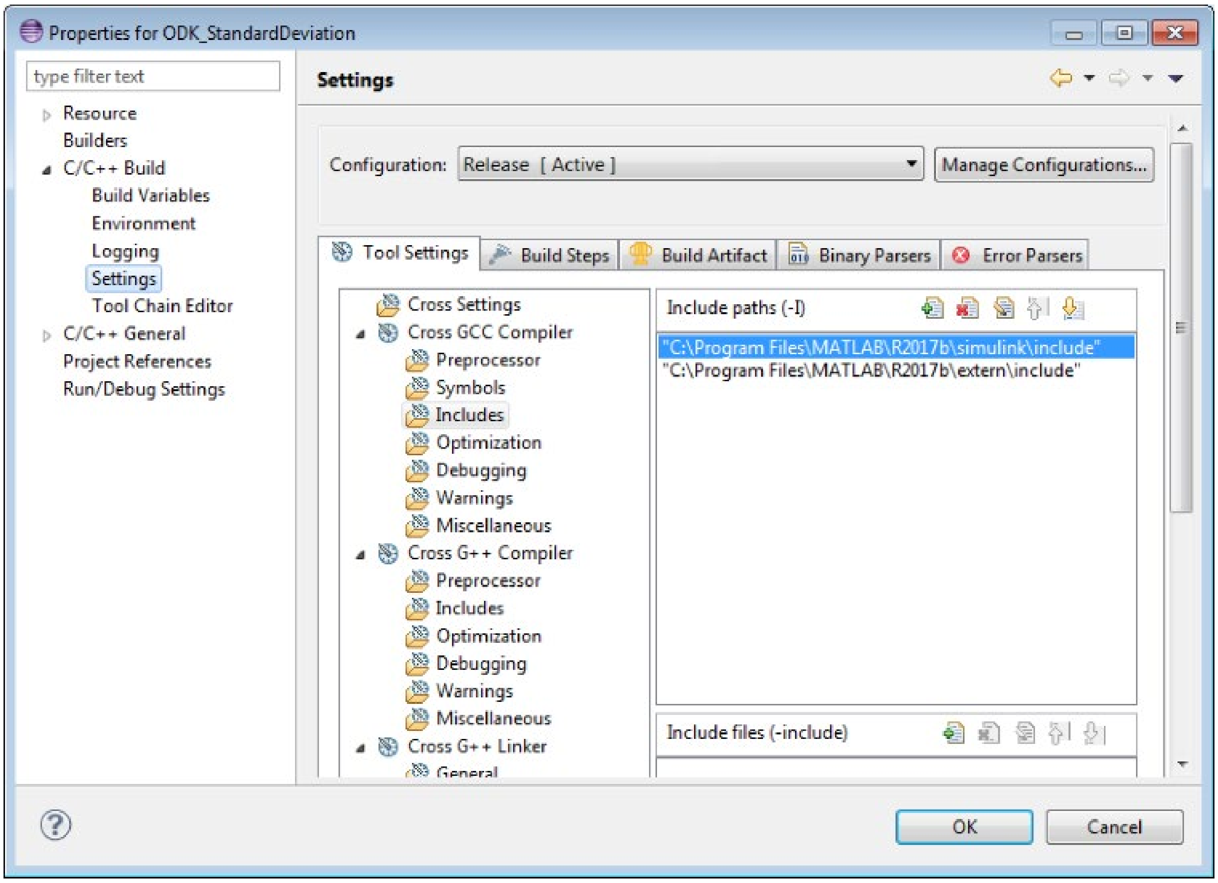
```
            break;
        case 1:
            // add your functions here
            retVal = Function_1(Input, Output);
            break;
        case 2:
            // add your functions here
            retVal = Function_2(Input, Output);
            break;
        default:
            retVal = EDU_COMMAND_NOT_IMPLEMENTED;
    }

    *pnUsedOutBytes = Output.EDU_LastByteChanged() + 1;

    return retVal;
} // Execute

/**
 * @ingroup User_defined_functions
 * @details User-defined function that is called from the Execute method on a call
 * to SFB65002 within IOT2000EDU.
 */
EDU_RESULT Function_2 (CWinLCReadData& rInput, CWinLCReadWriteData& rOutput )
{

////////////// READ INPUTS //////////
    short input_In1;
    rInput.EDU_ReadS7INT(0, input_In1 );
    Sum_U.In1 = input_In1;

    short input_In2;
    rInput.EDU_ReadS7INT(2, input_In2);
    Sum_U.In2 = input_In2;

////////  CALL THE STEP FUNCTIONS  /////
    Sum_step();

///////////// WRITE OUTPUTS //////////
    rOutput.EDU_WriteS7INT(0, (short)Sum_Y.Out1);

    return EDU_SUCCESS;
}
```

```
typedef struct {
  uint8_T In1;
  uint8_T In2;
} ExtU_Sum_T;
```

```
typedef struct {
  uint8_T Out1;
} ExtY_Sum_T;
```

7. Right-click on the Eclipse project "EDU_StandardDeviation" and select "Properties" from the shortcut menu.

   The "Properties for EDU_StandardDeviation" dialog opens.



Configuration: Release [Active]   Manage Configurations...

"C:\Program Files\MATLAB\R2017b\simulink\include"
"C:\Program Files\MATLAB\R2017b\extern\include"

Figure 10-2 "Properties for EDU_StandardDeviation" dialog

8. Under "C/C++ Build > Settings", insert all paths in the directories in which the files specific to Matlab/Simulink are located and confirm with "OK".
   For example:
   – tmwtypes.h: C:\Program Files\MATLAB\R2017b\extern\include
   – simstruc_types.h: C:\Program Files\MATLAB\R2017b\simulink\include
9. To compile the project, click on the icon "🔨".

Result

The SO file "libEDU_StandardDeviation.so" is created in "Project Explorer" under "Release".

10.5.5 Preparing TIA Portal for the Simulink model

To call the "Sum_Step" function from the IOT2000EDU, you must adapt your TIA program.

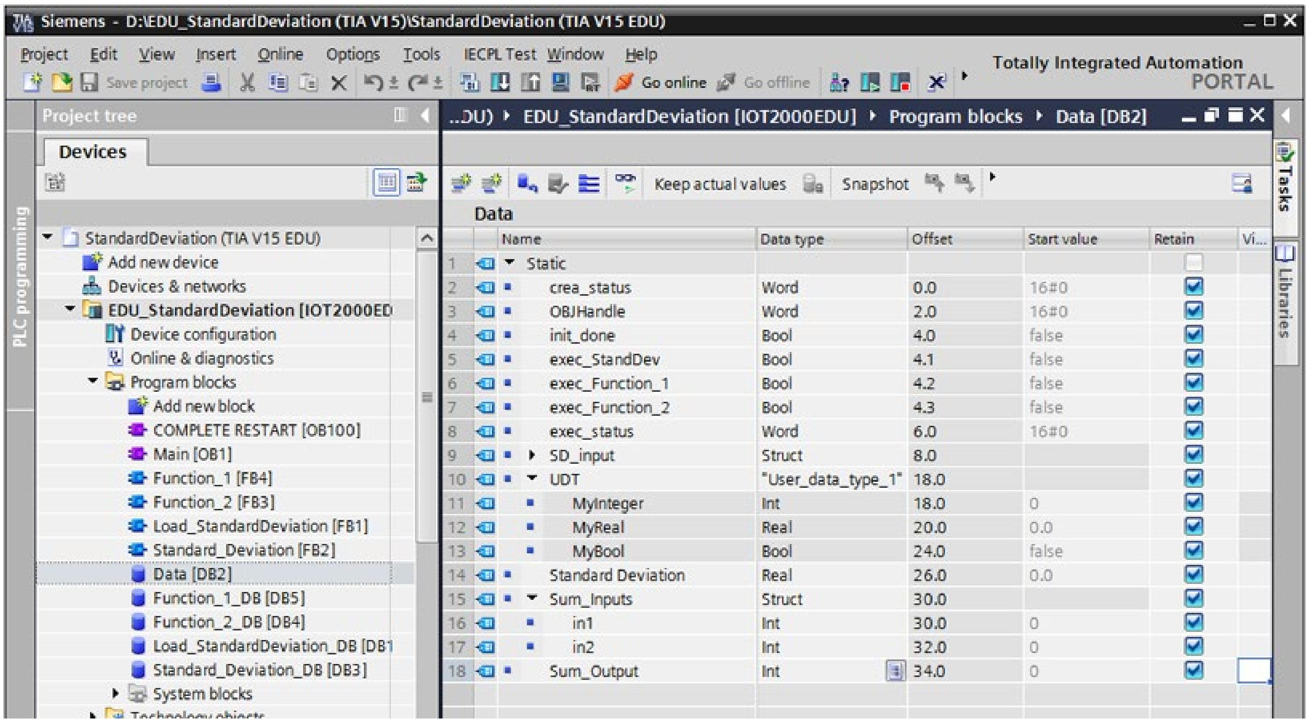
Requirement

You have opened the "EDU_StandardDeviation" program in the TIA Portal.

Procedure

Adaptation using the "EDU_StandardDeviation" program as an example.

1. Open the "Data" data block.


Siemens - D:\EDU_StandardDeviation (TIA V15)\StandardDeviation (TIA V15 EDU)
Project Edit View Insert Online Options Tools IECPL Test Window Help
Save project
Go online
Go offline
Totally Integrated Automation
PORTAL
Project tree
Devices
PLC programming
StandardDeviation (TIA V15 EDU)
Add new device
Devices & networks
EDU_StandardDeviation [IOT2000ED
Device configuration
Online & diagnostics
Program blocks
Add new block
COMPLETE RESTART [OB100]
Main [OB1]
Function_1 [FB4]
Function_2 [FB3]
Load_StandardDeviation [FB1]
Standard_Deviation [FB2]
Data [DB2]
Function_1_DB [DB5]
Function_2_DB [DB4]
Load_StandardDeviation_DB [DB1
Standard_Deviation_DB [DB3]
System blocks
Technology objects
..DU) ▸ EDU_StandardDeviation [IOT2000EDU] ▸ Program blocks ▸ Data [DB2]
Keep actual values
Snapshot
Tasks
Libraries

Data

	Name	Data type	Offset	Start value	Retain	Vi...
1	Static				☐	
2	crea_status	Word	0.0	16#0	✓	
3	OBJHandle	Word	2.0	16#0	✓	
4	init_done	Bool	4.0	false	✓	
5	exec_StandDev	Bool	4.1	false	✓	
6	exec_Function_1	Bool	4.2	false	✓	
7	exec_Function_2	Bool	4.3	false	✓	
8	exec_status	Word	6.0	16#0	✓	
9	SD_input	Struct	8.0		✓	
10	UDT	"User_data_type_1"	18.0		✓	
11	MyInteger	Int	18.0	0	✓	
12	MyReal	Real	20.0	0.0	✓	
13	MyBool	Bool	24.0	false	✓	
14	Standard Deviation	Real	26.0	0.0	✓	
15	Sum_Inputs	Struct	30.0		✓	
16	in1	Int	30.0	0	✓	
17	in2	Int	32.0	0	✓	
18	Sum_Output	Int	34.0	0	✓	

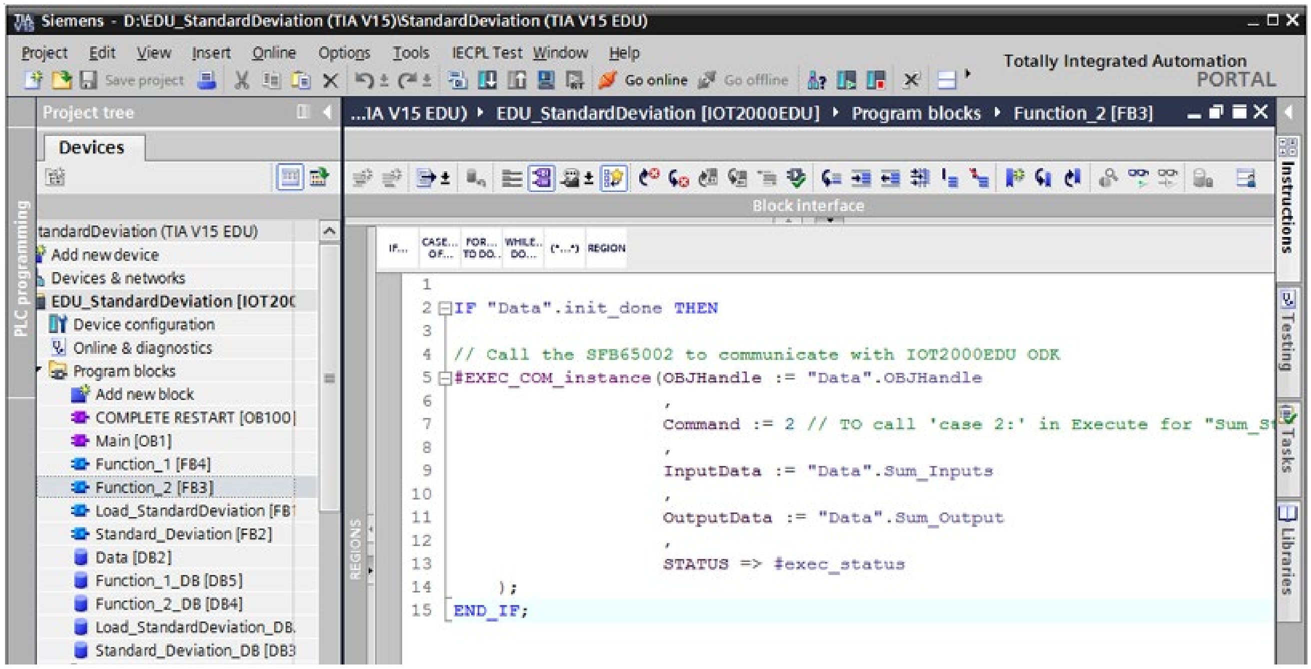
Figure 10-3 Assign input and output parameters

2. Add the input parameter (Sum_Inputs) and the output parameter (Sum_Output) that are transferred to the "Sum_Step" function.

   For "Sum_Inputs", assign the "Struct" data type with two variables (In1, In2) with the "Int" data type.

   Assign the "Int" data type for "Sum_Ouput".

3. Assign the input and output parameters to the "EXEC_COM" instruction via the "Function_2" function block.


Siemens - D:\EDU_StandardDeviation (TIA V15)\StandardDeviation (TIA V15 EDU)
Project Edit View Insert Online Options Tools IECPL Test Window Help
Save project
Go online
Go offline
Totally Integrated Automation PORTAL
Project tree
Devices
PLC programming
tandardDeviation (TIA V15 EDU)
Add new device
Devices & networks
EDU_StandardDeviation [IOT20
Device configuration
Online & diagnostics
Program blocks
Add new block
COMPLETE RESTART [OB100]
Main [OB1]
Function_1 [FB4]
Function_2 [FB3]
Load_StandardDeviation [FB1
Standard_Deviation [FB2]
Data [DB2]
Function_1_DB [DB5]
Function_2_DB [DB4]
Load_StandardDeviation_DB
Standard_Deviation_DB [DB3
...IA V15 EDU) ▸ EDU_StandardDeviation [IOT2000EDU] ▸ Program blocks ▸ Function_2 [FB3]
Block interface
IF... CASE... OF... FOR... TO DO... WHILE... DO... (*...*) REGION
REGIONS
Instructions
Testing
Tasks
Libraries

```

IF "Data".init_done THEN

    // Call the SFB65002 to communicate with IOT2000EDU ODK
    #EXEC_COM_instance(OBJHandle := "Data".OBJHandle
                       ,
                       Command := 2 // TO call 'case 2:' in Execute for "Sum_S
                       ,
                       InputData := "Data".Sum_Inputs
                       ,
                       OutputData := "Data".Sum_Output
                       ,
                       STATUS => #exec_status
    );
END_IF;
```

Figure 10-4 Assign "Sum_Step" input and output parameters

4. Load the TIA program into the IOT20x0 device.

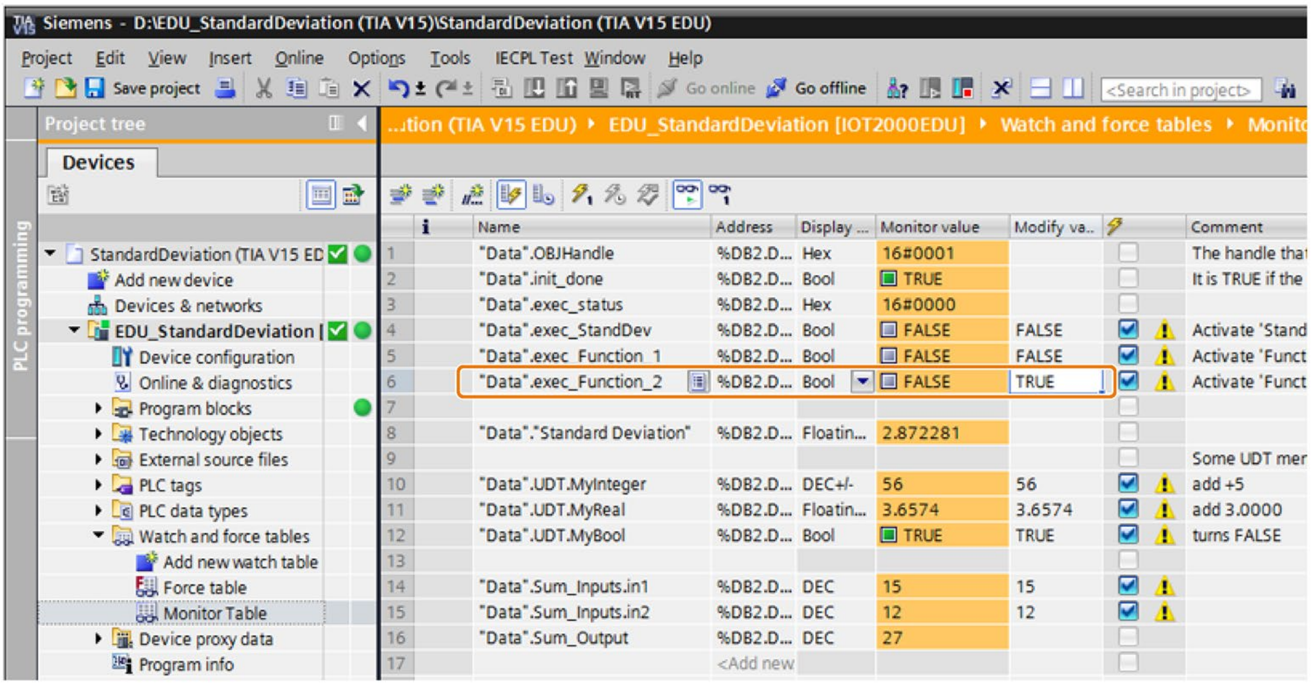
Note

Requirements for successful loading

- You have copied the SO file "libEDU_StandardDeviation.so" into the IOT20x0 device.
- You have defined the path of the SO file in the TIA Portal in the "Load_StandardDeviation" function block as follows:

  "*so:/home/root/libEDU_StandardDeviation.so"
- IOT2000EDU is running.

5. Test and monitor the "Function_2" function block via the watch table by setting the value "TRUE" in the "exec_Function_2" row of the "Control value" column.


Siemens - D:\EDU_StandardDeviation (TIA V15)\StandardDeviation (TIA V15 EDU)

...tion (TIA V15 EDU) ▸ EDU_StandardDeviation [IOT2000EDU] ▸ Watch and force tables ▸ Monit

Project tree — Devices:
- StandardDeviation (TIA V15 ED
  - Add new device
  - Devices & networks
  - EDU_StandardDeviation [
    - Device configuration
    - Online & diagnostics
    - Program blocks
    - Technology objects
    - External source files
    - PLC tags
    - PLC data types
    - Watch and force tables
      - Add new watch table
      - Force table
      - Monitor Table
    - Device proxy data
    - Program info

	i	Name	Address	Display ...	Monitor value	Modify va..	⚡		Comment
1		"Data".OBJHandle	%DB2.D...	Hex	16#0001		☐		The handle that
2		"Data".init_done	%DB2.D...	Bool	🟩 TRUE		☐		It is TRUE if the
3		"Data".exec_status	%DB2.D...	Hex	16#0000		☐		
4		"Data".exec_StandDev	%DB2.D...	Bool	▨ FALSE	FALSE	✓	⚠	Activate 'Stand
5		"Data".exec_Function_1	%DB2.D...	Bool	▨ FALSE	FALSE	✓	⚠	Activate 'Funct
6		"Data".exec_Function_2	%DB2.D...	Bool	▨ FALSE	TRUE	✓	⚠	Activate 'Funct
7							☐		
8		"Data"."Standard Deviation"	%DB2.D...	Floatin...	2.872281		☐		
9							☐		Some UDT mer
10		"Data".UDT.MyInteger	%DB2.D...	DEC+/-	56	56	✓	⚠	add +5
11		"Data".UDT.MyReal	%DB2.D...	Floatin...	3.6574	3.6574	✓	⚠	add 3.0000
12		"Data".UDT.MyBool	%DB2.D...	Bool	🟩 TRUE	TRUE	✓	⚠	turns FALSE
13							☐		
14		"Data".Sum_Inputs.in1	%DB2.D...	DEC	15	15	✓	⚠	
15		"Data".Sum_Inputs.in2	%DB2.D...	DEC	12	12	✓	⚠	
16		"Data".Sum_Output	%DB2.D...	DEC	27		☐		
17			<Add new				☐		

Figure 10-5 Test Sum_Step

6. To change the monitoring value of the output parameter, adjust the control value of the input parameters and click on the icon "⚡₁".

10.5.6 Data type conversion

Convert data types between the IOT2000EDU function library, Simulink Coder and TIA Portal. Make sure that the converted data types match.

The IOT2000EDU function library supports a maximum of 32 bits (4 bytes). If the data type in the program of the IOT2000EDU function library is larger than the data type used in the TIA Portal, data may be lost.

You can find information about data types in the "Data types (Page 56)“ section.

11 OUC communication

11.1 Connection establishment

IOT2000EDU supports "Open User Communication (OUC)" using the TCP/IP stack. With OUC functionality, IOT2000EDU can be used as a server or client, depending on the requirements of your program.

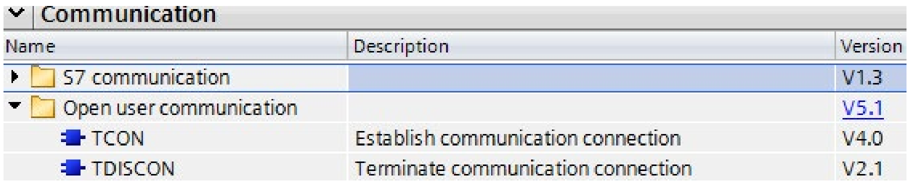
IOT2000EDU supports "OUC TCP" and UDP connection types.

Note

"ISO on TCP" is not supported.

IOT2000EDU supports the following OUC instructions for establishing the connection:

- "TCON" (SFC 133)
- "TDISCON" (SFC 134)



Communication

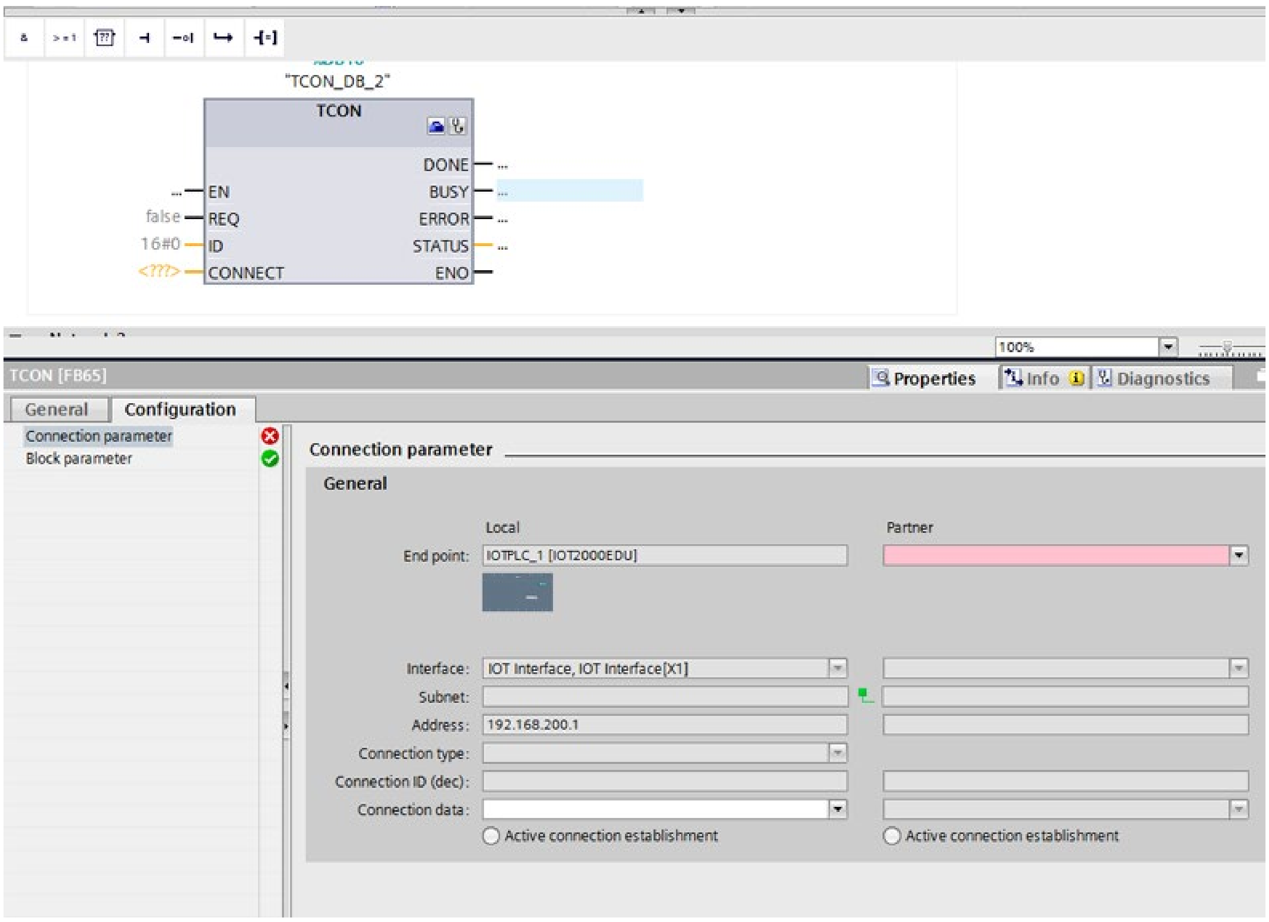
Name	Description	Version
S7 communication		V1.3
Open user communication		V5.1
TCON	Establish communication connection	V4.0
TDISCON	Terminate communication connection	V2.1

These instructions can be found in the TIA Portal in the "Instructions" task card under "Communication > Open User Communication".

Both TCP and UDP-based OUC connections require the "TCON" and "TDISCON" instructions.

11.2 Configuring TCON and TDISCON in the TIA Portal

You can set the input and output parameters of "TCON" and "TDISCON" either manually or using the TIA Portal configuration wizard.



TCON [FB65]

General | Configuration

Connection parameter
Block parameter

Connection parameter

General

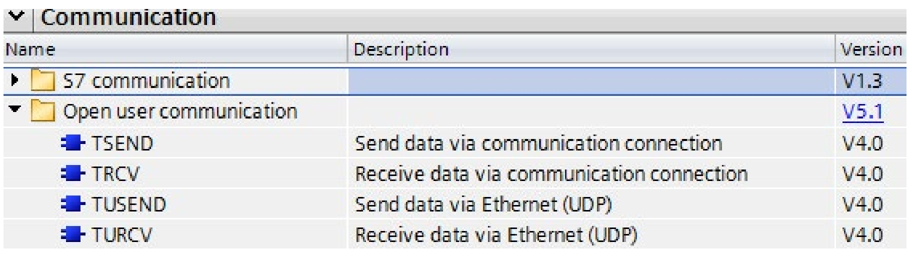
	Local	Partner
End point:	IOTPLC_1 [IOT2000EDU]	
Interface:	IOT Interface, IOT Interface[X1]	
Subnet:		
Address:	192.168.200.1	
Connection type:		
Connection ID (dec):		
Connection data:		
	Active connection establishment	Active connection establishment

11.3 Data exchange

Data can be sent in both directions when the connection is activated. This means data can be sent and received simultaneously.

The following communication instructions are available for data exchange:

- "TSEND" (SFC 131)
- "TRCV" (SFC 132)
- "TUSEND" (SFC 135)
- "TURCV" (SFC 136)

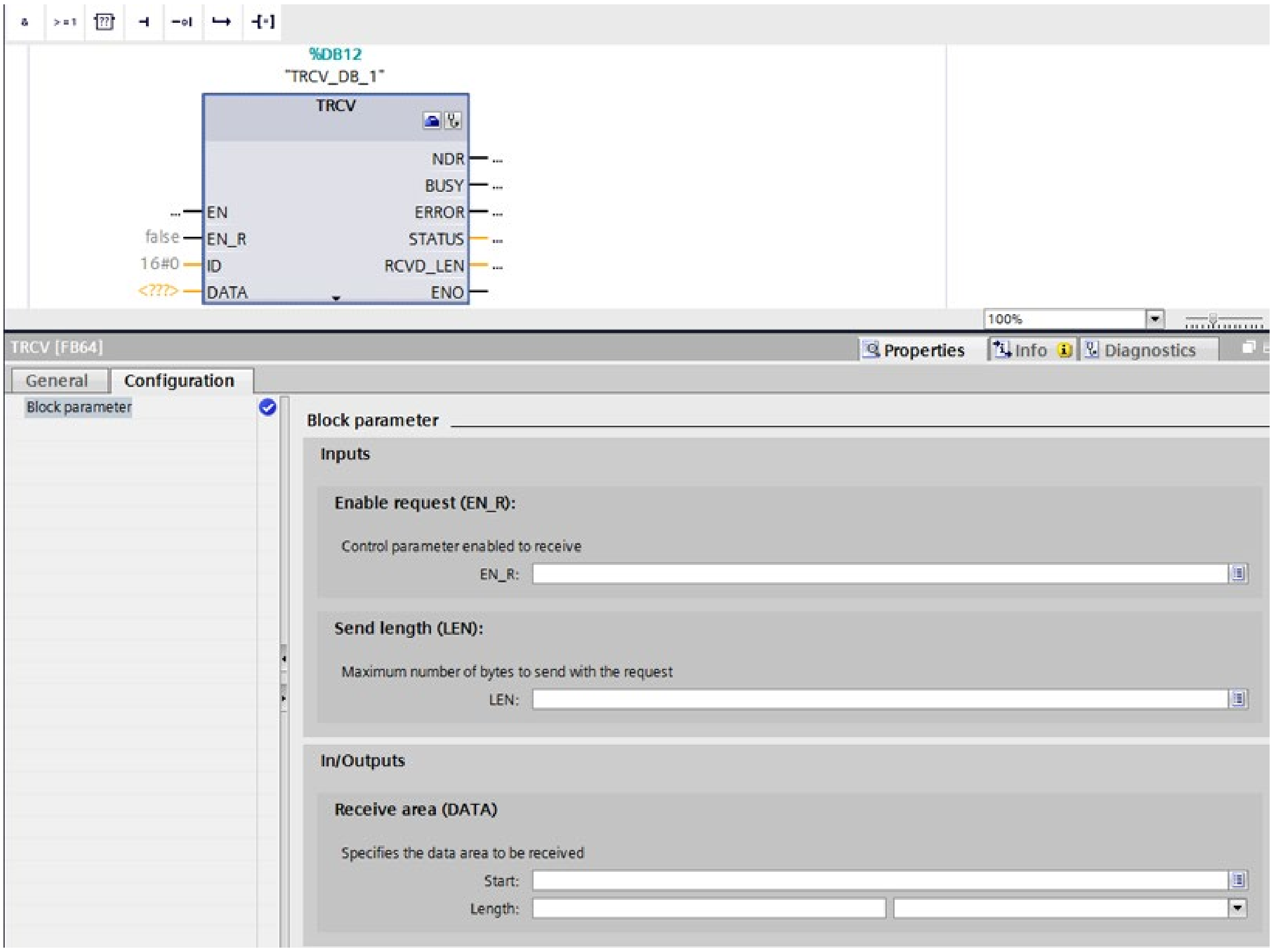


Communication

Name	Description	Version
S7 communication		V1.3
Open user communication		V5.1
TSEND	Send data via communication connection	V4.0
TRCV	Receive data via communication connection	V4.0
TUSEND	Send data via Ethernet (UDP)	V4.0
TURCV	Receive data via Ethernet (UDP)	V4.0

These instructions can be found in the TIA Portal in the "Instructions" task card under "Communication > Open User Communication".

11.4 Configuring TSEND, TRCV and TUSEND, TURCV in the TIA Portal



Block parameter

Inputs

Enable request (EN_R):

Control parameter enabled to receive

EN_R:

Send length (LEN):

Maximum number of bytes to send with the request

LEN:

In/Outputs

Receive area (DATA)

Specifies the data area to be received

Start:

Length:

"TSEND" and "TRCV" are connection instructions that only work with TCP-based OUC connections.

"TUSEND" and "TURCV" are connectionless instructions that only work with UDP-based OUC connections.

Note

IOT2000EDU supports up to 4 OUC connections simultaneously. Select the ports used by OUC carefully, because these ports should not be used by other network programs.

Ensuring code compatibility for IOT2000EDU and S7-1500

When programming your code, make sure that your program can run on both an IOT2000EDU and an S7-1500 controller. Note the following information:

Note

All four modules mentioned above return the code W#16#0000 even when called for the first time with REQ=1 and successful execution in RET_VAL.

If the program is to be executed on an S7-1500 controller, also ensure that RET_VAL=W#16#7001 is in your code.
You can find additional information on this in the TIA Portal help under "PLC programming > Instructions > Instructions (S7-1200, S7-1500) > Asynchronous instructions (S7-1200, S7-1500) > Difference between synchronous and asynchronous instructions".

Technical specifications A

A.1 Technical specifications

Technical specifications of the IOT2000EDU

The following technical specifications are in effect for the IOT2000EDU:

Article number	6ES7671-0LE00-0YB0
General information	
Product type designation	IOT2000EDU
Firmware version	V1.1
Memory	
Work memory	
• integrated (for program)	128 kbyte
• integrated (for data)	256 kbyte
CPU-blocks	
DB	
• Number, max.	200; Limited by RAM for data
• Size, max.	64 kbyte
FB	
• Number, max.	20; Limited by RAM for code
• Size, max.	64 kbyte
FC	
• Number, max.	20; Limited by RAM for code
• Size, max.	64 kbyte
OB	
• Number, max.	Limited only by RAM for code
• Number of free cycle OBs	1; OB 1
• Number of time alarm OBs	1; OB 10
• Number of delay alarm OBs	1; OB 20
• Number of cyclic interrupt OBs	9; OB 30-38
• Number of startup OBs	2; OB 100, 102
• Number of asynchronous error OBs	3
• Number of synchronous error OBs	1

Article number	6ES7671-0LE00-0YB0
Nesting depth	
• per priority class	16
• additional within an error OB	16
Counters, timers and their retentivity	
S7 counter	
• Number	2 048
IEC counter	
• Number	Limited by RAM for data/DB count
S7 times	
• Number	2 048
IEC timer	
• present	Yes
• Type	SFB
• Number	Limited by RAM for data/DB count
Data areas and their retentivity	
Flag	
• Number, max.	16 kbyte
• Number of clock memories	8; 8 clock memory bits, grouped into one clock memory byte
Local data	
• preset	32 kbyte
Digital channels	
• Inputs	20; Via Arduino UNO Shields
• Outputs	20; Via Arduino UNO Shields; of which 6 PWM outputs
Analog channels	
• Inputs	6; Via Arduino UNO Shields
Clock synchronization	
• supported	Yes; Synchronized to system clock of IoT2020/2040
Protocols	
Open IE communication	
• TCP/IP	Yes
– Data length, max.	16 kbyte
• UDP	Yes
– Number of connections, max.	4
– Data length, max.	1 472 byte

Article number	6ES7671-0LE00-0YB0
Web server	
• User-defined websites	No
• Number of HTTP clients	2
Communication functions	
PG/OP communication	Yes
S7 communication	
• supported	Yes; For engineering, HMI
Test commissioning functions	
Status block	Yes
Single step	Yes
Number of breakpoints	16
Status/control	
• Status/control variable	Yes
Diagnostic buffer	
• present	Yes
• Number of entries, max.	120
Hardware requirement	
Hardware required	IoT2020, IoT2040
Programming	
• Nesting levels	8
Programming language	
– LAD	Yes
– FBD	Yes
– STL	Yes
– SCL	Yes
– GRAPH	Yes

Additional information B

The following links provide additional information:

- IOT20x0 manual: https://support.industry.siemens.com/cs/document/109741658/simatic-iot2020-simatic-iot2040?dti=0&lc=en-WW (https://support.industry.siemens.com/cs/document/109741658/simatic-iot2020-simatic-iot2040?dti=0&lc=en-WW)
- Commissioning the IOT20x0: https://support.industry.siemens.com/tf/ww/en/posts/setting-up-the-simatic-iot2000/155642/?page=0&pageSize=10 (https://support.industry.siemens.com/tf/ww/en/posts/setting-up-the-simatic-iot2000/155642/?page=0&pageSize=10)
- IOT20x0 area in the Siemens Industry Support Forum: http://www.siemens.com/iot2000-forum (http://www.siemens.com/iot2000-forum)
- IOT2020 information page: https://siemens.com/iot2020 (https://siemens.com/iot2020)
- SD card example image: https://support.industry.siemens.com/cs/document/109741799/simatic-iot2000-sd-card-example-image (https://support.industry.siemens.com/cs/document/109741799/simatic-iot2000-sd-card-example-image?dti=0&lc=en-WW)
- Instructions for creating your own image: https://github.com/siemens/meta-iot2000 (https://github.com/siemens/meta-iot2000)
- Information on Yocto tags:
  - EXTRA_IMAGE_FEATURES: http://www.yoctoproject.org/docs/1.8/ref-manual/ref-manual.html#ref-features-image (http://www.yoctoproject.org/docs/1.8/ref-manual/ref-manual.html#ref-features-image)
  - PACKAGE_CLASS: http://www.yoctoproject.org/docs/1.8/ref-manual/ref-manual.html#var-PACKAGE_CLASSES (http://www.yoctoproject.org/docs/1.8/ref-manual/ref-manual.html#var-PACKAGE_CLASSES)
- MRAA library (version 1.6.1): https://github.com/intel-iot-devkit/mraa#installing-on-intel-32bit-yocto-based-opkg-image (https://github.com/intel-iot-devkit/mraa#installing-on-intel-32bit-yocto-based-opkg-image)
- Detail information on assignment of the GPIOs: https://support.industry.siemens.com/tf/WW/en/posts/how-is-the-assignment-of-the-gpios/155609?page=0&pageSize=10 (https://support.industry.siemens.com/tf/WW/en/posts/how-is-the-assignment-of-the-gpios/155609?page=0&pageSize=10)
- Information on the SIEMENS IOT2000 IO, Input/Output Module: https://support.industry.siemens.com/cs/document/109745681/iot2000-io-input-output-module?dti=0&lc=de-DE (https://support.industry.siemens.com/cs/document/109745681/iot2000-io-input-output-module?dti=0&lc=en-US)

Abbreviations C

CPU	Central Processing Unit
GPIO	General Purpose Input/Output
HSP	Hardware Support Package
IOT	Internet of Things
IOT2000EDU	SIMATIC S7 Software Controller, IOT2000EDU (EDU for education)
Opkg	Open PacKaGe management
OS	Operating System
PLC	Programmable Logic Controller
PWM	Pulse Width Modulation
SSH	Secure Shell
TIA	Totally Integrated Automation