SIEMENS

SIMATIC

S7-PLCSIM Advanced API

Programming Manual

Introduction 1

Product overview 2

Initialize API 3

Shut down API 4

Global functions (Native C++) 5

IBaseRuntimeManager 6

ISimulationRuntimeManager 7

IInstance 8

IRemoteRuntimeManager 9

Data types 10

Examples/applications 11

V7.0, 11/2024
A5E53246567-AC

Legal information

Warning notice system

This manual contains notices you have to observe in order to ensure your personal safety, as well as to prevent damage to property. The notices referring to your personal safety are highlighted in the manual by a safety alert symbol, notices referring only to property damage have no safety alert symbol. These notices shown below are graded according to the degree of danger.

⚠ DANGER

indicates that death or severe personal injury **will** result if proper precautions are not taken.

⚠ WARNING

indicates that death or severe personal injury **may** result if proper precautions are not taken.

⚠ CAUTION

indicates that minor personal injury can result if proper precautions are not taken.

NOTICE

indicates that property damage can result if proper precautions are not taken.

If more than one degree of danger is present, the warning notice representing the highest degree of danger will be used. A notice warning of injury to persons with a safety alert symbol may also include a warning relating to property damage.

Qualified Personnel

The product/system described in this documentation may be operated only by **personnel qualified** for the specific task in accordance with the relevant documentation, in particular its warning notices and safety instructions. Qualified personnel are those who, based on their training and experience, are capable of identifying risks and avoiding potential hazards when working with these products/systems.

Proper use of Siemens products

Note the following:

⚠ WARNING

Siemens products may only be used for the applications described in the catalog and in the relevant technical documentation. If products and components from other manufacturers are used, these must be recommended or approved by Siemens. Proper transport, storage, installation, assembly, commissioning, operation and maintenance are required to ensure that the products operate safely and without any problems. The permissible ambient conditions must be complied with. The information in the relevant documentation must be observed.

Trademarks

All names identified by ® are registered trademarks of Siemens Aktiengesellschaft. The remaining trademarks in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owner.

Disclaimer of Liability

We have reviewed the contents of this publication to ensure consistency with the hardware and software described. Since variance cannot be precluded entirely, we cannot guarantee full consistency. However, the information in this publication is reviewed regularly and any necessary corrections are included in subsequent editions.

Siemens Aktiengesellschaft
Digital Industries
Postfach 48 48
90026 NÜRNBERG
GERMANY

A5E53246567-AC
Ⓟ 10/2024 Subject to change

Copyright © Siemens 2024.
All rights reserved

Table of contents

1 Introduction 8

1.1 General information 8

1.2 S7-PLCSIM Advanced documentation guide 9

1.3 SIMATIC Technical Documentation 13

1.4 Tool support 14

1.5 Cybersecurity information 15

1.6 Administrator rights 15

2 Product overview 16

2.1 What is S7-PLCSIM Advanced? 16

2.2 Functions for licenses 17

2.3 System requirements 19

2.4 Restrictions due to antivirus programs 19

2.5 Access to instances 21

2.6 Interfaces of the Simulation Runtime 21

2.7 Overview of interfaces for native C++ 23

2.8 Overview of interfaces for managed code 27

2.9 Restrictions for file paths 30

2.10 Overview of data types for native C++ 30

2.11 Overview of data types for managed code 31

3 Initialize API 33

3.1 Load API library 33

3.2 Native C++ 34

3.2.1 InitializeApi() 34

3.2.2 RuntimeApiEntry_Initialize 35

3.3 .NET (C#) 37

3.3.1 Initialize 37

4 Shut down API 38

4.1 Native C++ 38

4.1.1 DestroyInterface() 38

4.1.2 RuntimeApiEntry_DestroyInterface 39

4.1.3 FreeApi() 40

4.1.4 ShutdownAndFreeApi() 41

4.2 .NET (C#) ... 42
4.2.1 Shut down API ... 42

5 Global functions (Native C++) ... 43

6 IBaseRuntimeManager ... 48

6.1 Simulation runtime instances ... 48

6.2 Interfaces for IBaseRuntimeManager ... 54

7 ISimulationRuntimeManager ... 62

7.1 Remote connections ... 62
7.1.1 RunAutodiscover() ... 66

7.2 Events for ISimulationRuntimeManager ... 67
7.2.1 OnRuntimeManagerLost events ... 68
7.2.2 OnAutodiscoverData events ... 70

8 IInstance ... 72

8.1 Interfaces for IInstances ... 72

8.2 Controller - Information and settings ... 80

8.3 Operating state ... 90

8.4 Stop simulation ... 100

8.5 Tag list ... 101

8.6 I/O access ... 106
8.6.1 Synchronizing inputs and outputs ... 106
8.6.2 Simulate peripheral I/O ... 107
8.6.3 I/O access via address - Reading ... 107
8.6.4 I/O access via address - Writing ... 113
8.6.5 I/O access via tag name - Reading ... 118
8.6.6 I/O access via tag name - Writing ... 139

8.7 Cycle control ... 163

8.8 Virtual and real time ... 168

8.9 Monitoring overflow ... 171

8.10 Acyclic services ... 172
8.10.1 Overview ... 172
8.10.2 ReadRecordDone/WriteRecordDone ... 174
8.10.3 AlarmNotification ... 175
8.10.4 ProcessEvent ... 178
8.10.5 PullOrPlugEvent ... 179
8.10.6 StatusEvent ... 181
8.10.7 ProfileEvent ... 182
8.10.8 UpdateEvent ... 183
8.10.9 GetConfiguredProcessEvent ... 184
8.10.10 RackOrStationFaultEvent ... 184

8.11 Events for IInstances ... 185
8.11.1 Events for operating state and cycle control ... 185
8.11.1.1 Synchronize simulation partner ... 186
8.11.1.1.1 Synchronize simulation partner cycle-controlled ... 186
8.11.1.1.2 Synchronize simulation partner time-controlled ... 190
8.11.1.2 OnOperatingStateChanged events ... 191
8.11.1.3 OnLedChanged events ... 195
8.11.1.4 OnSoftwareConfigurationChanged events ... 197
8.11.1.5 OnIPAddressChanged events ... 199
8.11.1.6 OnSyncPointReached events ... 201
8.11.1.7 OnHardwareConfigChanged events ... 203
8.11.2 Events for acyclic services ... 206
8.11.2.1 OnAlarmNotification events ... 208
8.11.2.2 OnProcessEvent events ... 209
8.11.2.3 OnPullOrPlugEvent events ... 210
8.11.2.4 OnStatusEvent events ... 211
8.11.2.5 OnProfileEvent events ... 212
8.11.2.6 OnUpdateEvent events ... 213
8.11.2.7 RackOrStationFault events ... 214

9 IRemoteRuntimeManager ... 215

9.1 Interfaces - Information and settings ... 215

9.2 Events for IRemoteRuntimeManager ... 217
9.2.1 OnConnectionLost events ... 217

10 Data types ... 220

10.1 DLL import functions (Native C++) ... 220
10.1.1 ApiEntry_Initialize ... 220
10.1.2 ApiEntry_DestroyInterface ... 221

10.2 Event callback functions (Native C++) ... 221
10.2.1 EventCallback_VOID ... 221
10.2.2 EventCallback_SRCC_UINT32_UINT32_INT32 ... 222
10.2.3 EventCallback_SRRSI_AD ... 222
10.2.4 EventCallback_IRRTM ... 223
10.2.5 EventCallback_II_SREC_ST_SROS_SROS ... 223
10.2.6 EventCallback_II_SREC_ST_UINT32_INT64_INT64_UINT32 ... 224
10.2.7 EventCallback_II_SREC_ST ... 224
10.2.8 EventCallback_II_SREC_ST_SRICC_UINT32_UINT32_UINT32_UINT32 ... 225
10.2.9 EventCallback_II_SREC_ST_SRLT_SRLM ... 225
10.2.10 EventCallback_II_SREC_ST_SDRI ... 226
10.2.11 EventCallback_II_SREC_ST_SDRI_BYTE ... 226
10.2.12 EventCallback_II_SREC_ST_UINT32_UINT32 ... 227
10.2.13 EventCallback_II_SREC_ST_UINT32_UINT32_EPET_UINT32 ... 227
10.2.14 EventCallback_II_SREC_ST_UINT32_EPPET_UINT32 ... 228
10.2.15 EventCallback_II_SREC_ST_UINT32_ERSFET ... 228
10.2.16 EventCallback_II_SREC_ST_UINT32 ... 229

10.3 Delegate definitions (managed code) ... 229
10.3.1 Delegate_Void ... 229
10.3.2 Delegate_SRCC_UINT32_UINT32_INT32 ... 230
10.3.3 Delegate_SRRSI_AD ... 230

10.3.4 Delegate_II_EREC_DT ... 231
10.3.5 Delegate_II_EREC_DT_EOS_EOS ... 231
10.3.6 Delegate_II_EREC_DT_ELT_ELM ... 232
10.3.7 Delegate_II_EREC_DT_UINT32_INT64_INT64_UINT32 ... 232
10.3.8 Delegate_IRRTM ... 233
10.3.9 Delegate_II_EREC_DT_SRICC_UINT32_UINT32_UINT32_UINT32 ... 233
10.3.10 Delegate_II_EREC_DT_SDRI ... 234
10.3.11 Delegate_II_EREC_DT_SDR ... 234
10.3.12 Delegate_SREC_ST_UINT32_EPPET_UINT32 ... 235
10.3.13 Delegate_SREC_ST_UINT32_UINT32_EPET_UINT32 ... 235
10.3.14 Delegate_SREC_ST_UINT32 ... 236
10.3.15 Delegate_SREC_ST_UINT32_UINT32 ... 236
10.3.16 Delegate_SREC_ST_UINT32_ERSFET ... 237

10.4 Definitions and constants ... 237

10.5 Unions (Native C++) ... 238
10.5.1 UIP ... 238
10.5.2 UDataValue ... 239

10.6 Structures ... 240
10.6.1 SDataValue ... 240
10.6.2 SDVBNI ... 241
10.6.3 SDataValueByAddress ... 242
10.6.4 SDataValueByAddressWithCheck ... 243
10.6.5 SDataValueByName ... 244
10.6.6 SDataValueByNameWithCheck ... 245
10.6.7 SConnectionInfo ... 245
10.6.8 SInstanceInfo ... 246
10.6.9 SDimension ... 246
10.6.10 STagInfo ... 247
10.6.11 SIP ... 248
10.6.12 SIPSuite4 ... 249
10.6.13 SOnSyncPointReachedResult ... 249
10.6.14 SDataRecordInfo ... 251
10.6.15 SDataRecord ... 252
10.6.16 SConfiguredProcessEvents ... 252
10.6.17 SDiagExtChannelDescription ... 253
10.6.18 SAutodiscoverData ... 254
10.6.19 SMAC address ... 255
10.6.20 SNetInterfaceInfo ... 255

10.7 Enumerations ... 255
10.7.1 ERuntimeErrorCode ... 256
10.7.2 EArea ... 259
10.7.3 EOperatingState ... 259
10.7.4 EOperatingMode ... 260
10.7.5 ECPUType ... 261
10.7.6 ECommunicationInterface ... 263
10.7.7 ENetworkMode ... 263
10.7.8 EPLCInterface ... 264
10.7.9 ELEDType ... 264
10.7.10 ELEDMode ... 265
10.7.11 EPrimitiveDataType ... 265

10.7.12 EDataType ... 267
10.7.13 ETagListDetails ... 270
10.7.14 ERuntimeConfigChanged ... 271
10.7.15 EInstanceConfigChanged ... 271
10.7.16 EPullOrPlugEventType ... 272
10.7.17 EProcessEventType ... 272
10.7.18 ESoftwareConfigChanged ... 273
10.7.19 EDirection ... 273
10.7.20 EDiagProperty ... 273
10.7.21 EDiagSeverity ... 274
10.7.22 ERackOrStationFaultType ... 274
10.7.23 ECycleTimeMonitoringMode ... 274
10.7.24 EAutodiscoverType ... 275

11 Examples/applications ... 276

11.1 Example: Communication via TCP/IP in Single Adapter Network Mode (promiscuous mode) ... 276

11.2 Example: Communication via TCP/IP in Multi-Adapter Network Mode (non-promiscuous mode) ... 277

Introduction

1

1.1 General information

Purpose of the documentation

This manual describes the API for the simulation software, SIMATIC S7-PLCSIM Advanced. You can use the API to simulate and test your SIMATIC STEP 7 programs on a simulated PLC instance. You can also use the API to connect the simulated PLC instance to a system or machine simulation (co-simulation).

Basic knowledge required

The API must only be used by qualified staff.
The following knowledge is required:

- Industrial Automation and Automation Technology
- Programming with STEP 7 (TIA Portal)
- SIMATIC CPUs and CPU programming
- PC-based automation using S7-1500 and WinCC Runtime Advanced
- Knowledge of programming with C++ or C#
- PC technology
- Windows operating system

Conventions

STEP 7: In this documentation, "STEP 7" is used as a synonym for all versions of the configuration and programming software "SIMATIC STEP 7 (TIA Portal)".

Also observe 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.

Scope

This manual is valid for the following product variants:

Article number	Product variants
6ES7823-1FA06-0YA5	SIMATIC S7-PLCSIM Advanced V7.0
6ES7823-1FE06-0YA5	SIMATIC S7-PLCSIM Advanced V7.0 Download
6ES7823-1FA06-0YE5	SIMATIC S7-PLCSIM Advanced V7.0 Upgrade V1.0 -> V7.0
6ES7823-1FE06-0YE5	SIMATIC S7-PLCSIM Advanced V7.0 Upgrade V1.0 -> V7.0 Download
6ES7823-1FE06-0YA7	SIMATIC S7-PLCSIM Advanced V7.0 POC
6ES7823-1FE00-0YN5	SIMATIC S7-PLCSIM Advanced Subscription Download

The articles each contain one license, which is valid for two instances.

NOTE

You can obtain online readme updates to the S7-PLCSIM manuals on the Internet (https://support.industry.siemens.com/cs/us/en/view/109824282).

You can find the following application examples for S7-PLCSIM Advanced on the Internet:

- SIMATIC S7-PLCSIM Advanced: Co-Simulation via API (https://support.industry.siemens.com/cs/ww/de/view/109739660/en)
- Digitalization with TIA Portal: Virtual commissioning with SIMATIC and Simulink (https://support.industry.siemens.com/cs/ww/en/document/109749187)

Recycling and disposal

For environmentally sustainable recycling and disposal of your old equipment, contact a certified electronic waste disposal service and dispose of the equipment according to the applicable regulations in your country.

Industry Mall

The Industry Mall (https://mall.industry.siemens.com) is the catalog and order system of Siemens AG for automation and drive solutions on the basis of Totally Integrated Automation (TIA) and Totally Integrated Power (TIP).

1.2 S7-PLCSIM Advanced documentation guide

The control panel buttons display the English S7-PLCSIM Advanced Function Manual and API Manual in PDF format. Download the free Adobe Reader if necessary to view PDFs. The installation folder of S7-PLCSIM Advanced includes German PDFs and English PDFs.

Other display formats are available at Siemens Industry Online Support (//https://support.industry.siemens.com/) (SIOS).

Accessing documentation on SIOS

Documentation for this product is available at Siemens Industry Online Support (//https://support.industry.siemens.com/). Formats include the following for display or download:

- PDF document
- Multimedia document (HTML5 Web site)
- Website with chapter structure (useful for providing a specific topic link to other people)

From the Web site, search for your product name. Filter the search results by manuals if you are only looking for documentation. For your product, select the edition that corresponds to the version of your product. You can switch the display language for the page and the documents from the menu.

The various document formats provide navigation and search features based on their documentation format.

Navigating an HTML5 site

When you open an HTML5 Web site, click Begin from the Start page.

The table of contents displays on the left. Each chapter collapses as you navigate from one chapter to the next to optimize screen space on devices such as phones and tablets. You can also click the button to hide the table of contents. Click to restore it.

Left and right arrow buttons allow to navigate back and forward through topics that you have previously viewed:

Previous and next page buttons allow you to navigate sequentially through topics and chapters in the order of the table of contents:

Searching in an HTML5 site

To search an HTML5 Web site, click the button on the right. When you enter your search text, you see a list of topics that contain the search text. Click any topic to view it. The page highlights occurrences of the matched search text.

To see where the topic exists in the table of contents, click the button to switch from the search view to the table of contents view. To switch back to your search, click again. The search functions much the same as any search from a Web browser. Ctrl+F searches within the displayed topic.

Changing the language of the HTML5 site

The S7-PLCSIM Advanced HTML5 documentation includes German and English. Click the button to switch to another language.



The documentation for the SIMATIC S7-1500 automation system, for the 1513/1516pro-2 PN, SIMATIC Drive Controller CPUs based on SIMATIC S7-1500 and the SIMATIC ET 200MP, ET 200SP, ET 200AL and ET 200eco PN distributed I/O systems is arranged into three areas.

This arrangement enables you to access the specific content you require.

You can download the documentation free of charge from the Internet (https://support.industry.siemens.com/cs/ww/en/view/109742705).

Basic information



The system manuals and Getting Started describe in detail the configuration, installation, wiring and commissioning of the SIMATIC S7-1500, SIMATIC Drive Controller, ET 200MP, ET 200SP, ET 200AL and ET 200eco PN systems. Use the corresponding operating instructions for 1513/1516pro-2 PN CPUs.

The STEP 7 online help supports you in the configuration and programming.

Examples:

- Getting Started S7-1500
- System manuals
- Operating instructions ET 200pro and 1516pro-2 PN CPU
- Online help TIA Portal

Device information



Equipment manuals contain a compact description of the module-specific information, such as properties, wiring diagrams, characteristics and technical specifications.

Examples:

- Equipment manuals for CPUs
- Equipment manuals for interface modules
- Equipment manuals for digital modules
- Equipment manuals for analog modules
- Equipment manuals for communication modules
- Equipment manuals for technology modules
- Equipment manuals for power supply modules
- Equipment manuals for BaseUnits

General information



The function manuals contain detailed descriptions on general topics relating to the SIMATIC Drive Controller and the S7-1500 automation system.

Examples:

- Function Manual Diagnostics
- Function Manual Communication
- Function Manuals Motion Control
- Function Manual Web Server
- Function Manual Cycle and Response Times
- PROFINET Function Manual
- PROFIBUS Function Manual

Product Information

Changes and supplements to the manuals are documented in a Product Information. The Product Information takes precedence over the device and system manuals.

You will find the latest Product Information on the Internet:

- S7-1500/ET 200MP (https://support.industry.siemens.com/cs/de/en/view/68052815)
- SIMATIC Drive Controller (https://support.industry.siemens.com/cs/de/en/view/109772684/en)
- Motion Control (https://support.industry.siemens.com/cs/de/en/view/109794046/en)
- ET 200SP (https://support.industry.siemens.com/cs/de/en/view/73021864)
- ET 200eco PN (https://support.industry.siemens.com/cs/ww/en/view/109765611)

Manual Collections

The Manual Collections contain the complete documentation of the systems put together in one file.

You will find the Manual Collections on the Internet:

- S7-1500/ET 200MP/SIMATIC Drive Controller (https://support.industry.siemens.com/cs/ww/en/view/86140384)
- ET 200SP (https://support.industry.siemens.com/cs/ww/en/view/84133942)
- ET 200AL (https://support.industry.siemens.com/cs/ww/en/view/95242965)
- ET 200eco PN (https://support.industry.siemens.com/cs/ww/en/view/109781058)

1.3 SIMATIC Technical Documentation

Additional SIMATIC documents will complete your information. You can find these documents and their use at the following links and QR codes.

The Industry Online Support gives you the option to get information on all topics. Application examples support you in solving your automation tasks.

Overview of the SIMATIC Technical Documentation

Here you will find an overview of the SIMATIC documentation available in Siemens Industry Online Support:

Industry Online Support International

Watch this short video to find out where you can find the overview directly in Siemens Industry Online Support and how to use Siemens Industry Online Support on your mobile device:

Quick introduction to the technical documentation of automation products per video

YouTube video: Siemens Automation Products - Technical Documentation at a Glance

mySupport

With "mySupport" you can get the most out of your Industry Online Support.

Registration	You must register once to use the full functionality of "mySupport". After registration, you can create filters, favorites and tabs in your personal workspace.
Support requests	Your data is already filled out in support requests, and you can get an overview of your current requests at any time.
Documentation	In the Documentation area you can build your personal library.
Favorites	You can use the "Add to mySupport favorites" to flag especially interesting or frequently needed content. Under "Favorites", you will find a list of your flagged entries.
Recently viewed articles	The most recently viewed pages in mySupport are available under "Recently viewed articles".
CAx data	The CAx data area gives you access to the latest product data for your CAx or CAe system. You configure your own download package with a few clicks: • Product images, 2D dimension drawings, 3D models, internal circuit diagrams, EPLAN macro files • Manuals, characteristics, operating manuals, certificates • Product master data

You can find "mySupport" on the Internet.

Application examples

The application examples support you with various tools and examples for solving your automation tasks. Solutions are shown in interplay with multiple components in the system - separated from the focus on individual products.

You can find the application examples on the Internet.

1.4 Tool support

Tools

The tools described below support you in all steps: from planning, over commissioning, all the way to analysis of your system.

TIA Selection Tool

The TIA Selection Tool tool supports you in the selection, configuration, and ordering of devices for Totally Integrated Automation (TIA).

As successor of the SIMATIC Selection Tools , the TIA Selection Tool assembles the already known configurators for automation technology into a single tool.

With the TIA Selection Tool , you can generate a complete order list from your product selection or product configuration.

You can find the TIA Selection Tool on the Internet.
(https://support.industry.siemens.com/cs/us/en/view/109767888)

SINETPLAN

SINETPLAN, the Siemens Network Planner, supports you in planning automation systems and networks based on PROFINET. The tool facilitates professional and predictive dimensioning of your PROFINET installation as early as in the planning stage. In addition, SINETPLAN supports you during network optimization and helps you to exploit network resources optimally and to plan reserves. This helps to prevent problems in commissioning or failures during productive operation even in advance of a planned operation. This increases the availability of the production plant and helps improve operational safety.

The advantages at a glance

- Network optimization thanks to port-specific calculation of the network load
- Increased production availability thanks to online scan and verification of existing systems
- Transparency before commissioning through importing and simulation of existing STEP 7 projects
- Efficiency through securing existing investments in the long term and the optimal use of resources

You can find SINETPLAN on the Internet (https://new.siemens.com/global/en/products/automation/industrial-communication/profinet/sinetplan.html).

1.5 Cybersecurity information

Siemens provides products and solutions with industrial cybersecurity 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 cybersecurity 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 cybersecurity measures that may be implemented, please visit
https://www.siemens.com/cybersecurity-industry.

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 customer's exposure to cyber threats.

To stay informed about product updates, subscribe to the Siemens Industrial Cybersecurity RSS Feed under
https://new.siemens.com/cert.

1.6 Administrator rights

NOTE

Administrator rights

For normal operation, the standard user rights are sufficient. You need administrator rights only when you install or repair S7-PLCSIM Advanced or change the network configuration.

Product overview

2

2.1 What is S7-PLCSIM Advanced?

You use S7-PLCSIM Advanced to simulate your CPU programs on a simulated PLC instance. You do not need any real controllers for simulation. You configure your CPU with STEP 7 in TIA Portal, program your application logic and then load the hardware configuration and the program into the simulated CPU. From STEP 7, you run your program logic, monitor the effects of simulated inputs and outputs, and adapt your programs.

In addition to communication via Softbus, which limits communication to a local PC or virtual machine, S7-PLCSIM Advanced offers a full Ethernet connection and thus also supports distributed I/O systems.

S7-PLCSIM Advanced also includes an application programming interface (API), which enables interaction with native C++/C# programs or simulation software. The API extends simulation capabilities and provides features such as the following:

- Simulation of hardware interrupts
- Stopping a simulation
- Synchronization with simulation partners

For these and other API functions, refer to the S7-PLCSIM Advanced API manual included with your release.

Application areas

Typical application areas of S7-PLCSIM Advanced are:

- Verification of the user program (TIA Portal)
- Testing of the STEP 7 program
- Software in the loop simulation for the virtual commissioning of machine tools/production machines, production cells, and production lines in a plant.
- Operator training through the connection of a real HMI

Advantages

S7-PLCSIM Advanced offers numerous advantages:

- Improve the quality of automation projects by early error detection
- Avoid costs for hardware in simulation environments
- Shorten commissioning time
- Reduce risk for commissioning
- Train operators earlier
- Increase production efficiency by optimizing program components

- Increase efficiency during replacement of machine components
- Increase efficiency during expansion of existing plants

2.2 Functions for licenses

API functions for licenses

S7-PLCSIM Advanced regularly checks whether a license is available. The following return values provide information about the status (for example, for C++):

- Return values for API function `PowerOn()` and callback function `OnOperatingStateChanged`
  - `SREC_OK` when a floating license is available.
  - `SREC_WARNING_TRIAL_MODE_ACTIVE` when an instance is started with the Trial License.
  - `SREC_WARNING_RUNNING_ON_TIA_PORTAL_TEST_SUITE` when no valid license for S7-PLCSIM Advanced is available, but a "TIA Portal Test Suite" license is available.

    S7-PLCSIM Advanced starts with this license. A download from the TIA Portal is possible, but the instance terminates without feedback if the download was not made from the TIA Portal Test Suite.
  - `SREC_NOT_EMPTY` when no valid license for S7-PLCSIM Advanced is available, but a "TIA Portal Test Suite" license is available.

    If this is the case, power-up from the Virtual SIMATIC Memory Card is not supported.
- Return value for callback function `OnOperatingStateChanged`
  - `SREC_LICENSE_NOT_FOUND` when the instance is automatically shut down after 21 days.

Components of the Simulation Runtime

The following components are relevant for handling the Simulation Runtime of S7-PLCSIM Advanced:

Table 2-1 Components of the Simulation Runtime

Components	Description
• "Siemens.Simatic.Simulation. Runtime.Manager.exe"	A Windows process that runs in the background. Main component of Runtime that manages all other Runtime components. The process is started automatically as soon as an application attempts to initialize the Runtime API. It is ended automatically as soon as there is no longer any application running that initialized the Runtime API.
• "Siemens.Simatic.Simulation. Runtime.Instance.exe"	The process of the instance that loads a DLL of a simulated PLC instance. Each simulated PLC instance generates its own process.
• "Siemens.Simatic.Simulation. Runtime.Api.x86.dll" • "Siemens.Simatic.Simulation. Runtime.Api.x64.dll"	API libraries that must load an application to use the Simulation Runtime. The libraries contain interfaces for native code and managed code. The "Runtime.Api.x86.dll" is loaded exclusively by 32-bit applications, and the "Runtime.Api.x64.dll" by 64-bit applications.

Components	Description
• "SimulationRuntimeApi.h"	Header file that describes all data types that require a native C++ application to use the API library.
• "Siemens.Simatic.PlcSim.Vplc1500.ODKClient.so.exe"	ODK client process for a CPU function library (original Shared Object)
• "Siemens.Simatic.PlcSim.Vplc1500.ODKClient.x86.exe"	ODK client process for a 32-bit application
• "Siemens.Simatic.PlcSim.Vplc1500.ODKClient.x64.exe"	ODK client process for a 64-bit application

External applications and Simulation Runtime

The following figure schematically presents the access of external applications to Simulation Runtime via the Runtime API. The Simulation Runtime Manager manages the Runtime instances. These load the libraries of the simulated PLC instances.

An external application can be, for example, another simulation software or a graphical user interface (GUI).

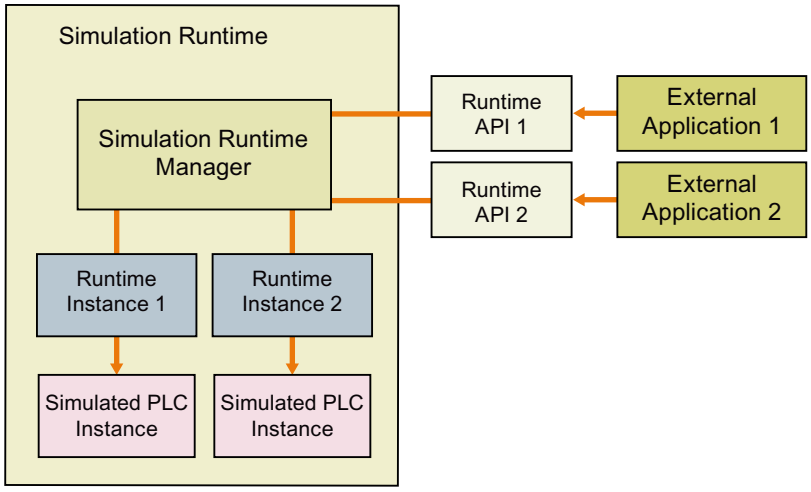


Figure 2-1 External applications and Simulation Runtime

2.3 System requirements

For S7-PLCSIM Advanced to operate efficiently, the following minimum requirements for computer hardware or for a virtual machine must be met.

	Hardware	Virtual machine
Processor	• At least one core Intel® Core™ i7 or comparable per instance started. • At least one additional core for the operating system • At least one additional core for the additional active applications	• One virtual CPU per started instance has to be assigned to the VM. • A corresponding number of processors has to be physically available on the host. • At least one additional core for the operating system • At least one additional core for the additional active applications • At least two cores, if STEP 7 (TIA Portal) is installed on the VM
RAM	• 1 GB per started instance • At least 4 GB for the Windows operating system • Additional RAM corresponding to the requirements of the remaining active applications	• 1 GB per started instance • At least 4 GB for the Windows operating system • Additional RAM corresponding to the requirements of the remaining active applications • At least 8 GB, if STEP 7 (TIA Portal) is installed on the VM
Free hard disk space	5 GB	5 GB
Screen resolution	Minimum 1024 x 768	Minimum 1024 x 768

Operating systems and virtualization platforms

S7-PLCSIM Advanced supports the same operating systems, virtualization, and hard disk encryption technology as the corresponding STEP 7 Basic version., with the exception of the Windows Home operating systems. S7-PLCSIM does not support the Windows Home operating systems. See the TIA Portal Information System for a full list of supported technologies.

Refer to online information for installing STEP 7 (TIA Portal) and S7-PLCSIM Advanced on a virtual machine.

See also

Installing STEP 7 and S7-PLCSIM Advanced on a VM
(https://support.industry.siemens.com/cs/ww/en/view/78788417)

2.4 Restrictions due to antivirus programs

NOTICE
Restrictions due to virus scanners and Advanced Threat Protection software (ATP software) Virus scanners and ATP software that monitor the behavior of processes and communication can have a significant influence on the performance of the runtime and communication of S7-PLCSIM Advanced and even prevent S7-PLCSIM Advanced instances from starting.

Solution

You can decrease restrictions during installation and runtime of S7-PLCSIM Advanced. To do so, define exceptions for the virus scanner for secure files and folders. Add the following folders to the exceptions:

- "C:\Program Files\Common Files\Siemens\PLCSIMADV\Drivers"
- "C:\Program Files (x86)\Common Files\Siemens\PLCSIMADV"
- "C:\Program Files (x86)\SIEMENS\Automation\PLCSIMADV\bin"

If the virus scanner only allows files as exceptions, add all files from the listed folders to the exceptions. The procedure is described in the instructions of the respective manufacturer.

NOTE

Refer to the latest readme (https://support.industry.siemens.com/cs/us/en/view/109824282) on the Siemens Industry Online Support site for updates. Select the Edition that corresponds to your release version.

Virus scanner and security software support

Refer to "General software and hardware requirements" in the TIA Portal information system for the supported virus scanners and security software.

Siemens recommends Windows Defender that is part of the Windows operating system.

Symantec Endpoint Protection 14.3

Note the following restriction for Symantec Endpoint Protection 14.3 when using TCP/IP Single Adapter or TCP/IP Multiple Adapter:

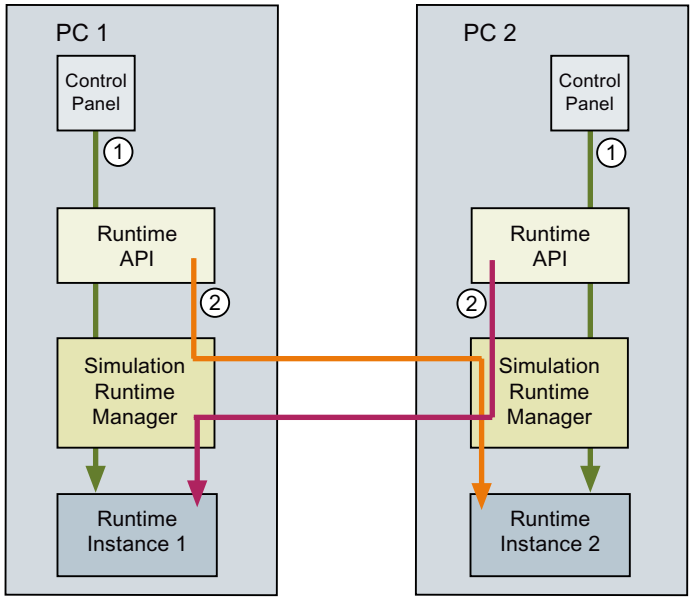
If you use Npcap with Symantec Endpoint Protection, select one of the following options:

- Uninstall Symantec Endpoint Protection.
- Use Symantec Endpoint Protection with Npcap V1.60 or earlier.

2.5 Access to instances

Access via the Control Panel and the API

You can access a local instance on the PC via the Control Panel. It does not matter on which PC an instance was created and started. With distributed communication, the Runtime API accesses the instance of the other PCs via the Simulation Runtime Manager.



① Access to a local instance via the Control Panel

② Access to a remote instance on the Runtime API

Figure 2-2 Access to instances with distributed communication

API functions

The following interfaces are available for native C++ (Page 23) and managed code (Page 27):

- Overview of interfaces for native C++ (Page 23)
- Overview of interfaces for managed code (Page 27)

2.6 Interfaces of the Simulation Runtime

The user interfaces of the Simulation Runtime include functions you use for tasks such as the following:

- Creating instances
- Changing the operating mode of a simulated PLC instance
- Exchanging I/O data

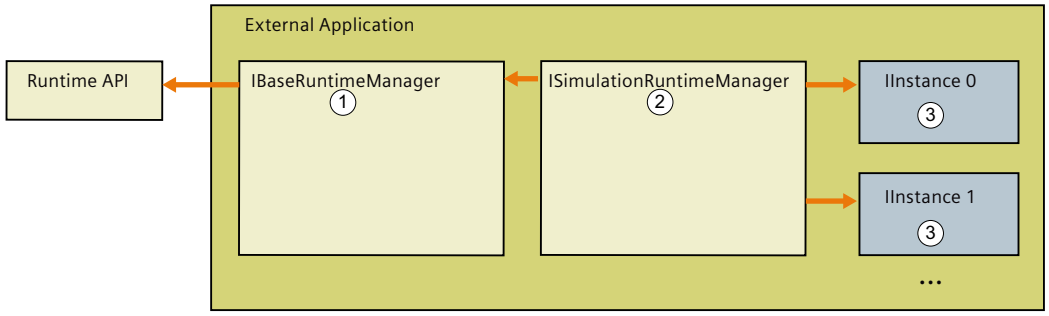
Simulation Runtime has the following user interfaces:

- IBaseRuntimeManager
- ISimulationRuntimeManager
- IInstances
- IRemoteRuntimeManager

ISimulationRuntimeManager and IRemoteRuntimeManager are derived from IBaseRuntimeManager.

API and external applications

The Runtime API makes the interfaces available to an external application.



① **IBaseRuntimeManager**
Base interface of the Runtime Manager. It is used to register new Runtime instances, to search through existing Runtime instances, and to receive an interface of a registered instance. Up to 16 instances can be registered in one Runtime Manager.

② **ISimulationRuntimeManager**
Derived from IBaseRuntimeManager. It is used to handle events, manage connections with other Runtime Managers, and check API and Runtime Manager status.

③ **IInstances**
Interface of a Runtime instance. It is used to change the operating state of a simulated PLC instance and to exchange I/O data. Each instance has a unique name and an ID.

Figure 2-3 API and external applications

Access to API functions and data types

Required functions and data types are available for native C++ and .NET (C#).

- Overview of interfaces for native C++ (Page 23)
- Overview of data types for native C++ (Page 30)
- Overview of interfaces for managed code (Page 27)
- Overview of data types for managed code (Page 31)

NOTE

The list of tables in this manual gives you direct access to the description of the individual functions and data types.

Transfer parameters for API functions

All API functions that return a value using the function parameters expect a user-allocated memory area as a transfer parameter. Null pointers are not permitted. Exceptions to this are the functions that return an interface of a simulated PLC instance:

- An IBaseRuntimeManager interface
- An ISimulationRuntimeManager interface
- An IRemoteRuntimeManager interface
- An IInstance interface

2.7 Overview of interfaces for native C++

Initializing and shutting down API

Table 2-2 Overview of initializing and shutting down API - Native C++

Actions	Functions
Initialize API (Page 34)	`InitializeApi` `RuntimeApiEntry_Initialize`
Shut down API (Page 38)	`DestroyInterface` `RuntimeApiEntry_DestroyInterface`
Logging off API library (Page 40)	`FreeApi` `ShutdownAndFreeApi`

Global functions

Table 2-3 Overview of global functions - Native C++

Actions	Functions

Global functions (Page 43)	`GetNameOfAreaSection()`
	`GetNameOfCPUType()`
	`GetNameOfCommunicationInterface()`
	`GetNameOfDataType()`
	`GetNameOfLEDMode()`
	`GetNameOfLEDType()`
	`GetNameOfOperatingMode()`
	`GetNameOfOperatingState()`
	`GetNameOfPrimitiveDataType()`
	`GetNameOfTagListDetails()`
	`GetNameOfErrorCode()`
	`GetNameOfRuntimeConfigChanged()`
	`GetNameOfInstanceConfigChanged()`
	`GetNameOfDiagSeverity()`
	`GetNameOfDirection()`
	`GetNameOfRackOrStationFaultType()`
	`GetNameOfProcessEventType()`
	`GetNameOfPullOrPlugEventType()`
	`GetNameOfCycleTimeMonitoringMode()`
	`GetNameOfDiagProperty()`
	`GetNameOfAutodiscoverType()`
	`GetNameOfSoftwareConfigChanged()`
	`GetNameOfNetworkMode()`

API IBaseRuntimeManager

Table 2-4 Overview of IBaseRuntimeManager functions - Native C++

Settings	Functions

Interface (Page 54)	`GetVersion()`
	`GetPort()`
	`GetStrictMotionTiming()`
	`SetStrictMotionTiming()`
	`GetNetworkMode()`
	`SetNetworkMode()`
	`GetNetInterfaces()`
	`SetNetInterfaceBindings()`
	`ResetNetInterfaceBindings()`
	`GetDefaultStoragePath()`
	`SetDefaultStoragePath()`

Simulation Runtime instances (Page 48)	`GetRegisteredInstancesCount()`
	`GetRegisteredInstanceInfoAt()`
	`RegisterInstance()`
	`RegisterCustomInstance()`
	`CreateInterface()`

API ISimulationRuntimeManager

Table 2-5 Overview of API ISimulationRuntimeManager functions - Native C++

Settings	Functions

Interface (Page 70)	`IsInitialized()`
	`IsRuntimeManagerAvailable()`
	`Shutdown()`

Remote connections (Page 62)	`OpenPort()`
	`ClosePort()`
	`GetRemoteConnectionsCount()`
	`GetRemoteConnectionInfoAt()`
	`RemoteConnect()`
	`RunAutodiscover()`

Table 2-6 Overview of API ISimulationRuntimeManager events - Native C++

Events	Functions

OnSoftwareConfigurationChanged (Page 197)	`RegisterOnSoftwareConfigurationChangedCallback()`
	`UnregisterOnSoftwareConfigurationChangedCallback()`
	`RegisterOnSoftwareConfigurationChangedEvent()`
	`UnregisterOnSoftwareConfigurationChangedEvent()`
	`WaitForOnSoftwareConfigurationChangedEvent()`

OnRuntimeManagerLost (Page 68)	`RegisterOnRuntimeManagerLostCallback()`
	`UnregisterOnRuntimeManagerLostCallback()`
	`RegisterOnRuntimeManagerLostEvent()`
	`UnregisterOnRuntimeManagerLostEvent()`
	`WaitForOnRuntimeManagerLostEvent()`

OnAutodiscover (Page 70)	`RegisterOnAutodiscoverCallback()` `UnregisterOnAutodiscoverCallback()`

API IInstance

Table 2-7 Overview of IInstance functions - Native C++

Settings	Functions

Interface (Page 72)	`GetID()`
	`GetName()`
	`GetCPUType()`
	`SetCPUType()`
	`GetCommunicationInterface()`
	`SetCommunicationInterface()`
	`GetInfo()`
	`UnregisterInstance()`
	`GetStrictMotionTiming()`
	`SetStrictMotionTiming()`

Controller (Page 80)	`GetControllerName()`
	`GetControllerShortDesignation()`
	`GetControllerIPCount()`
	`GetControllerIP()`
	`GetControllerIPSuite4()`
	`SetIPSuite()`
	`GetStoragePath()`
	`SetStoragePath()`
	`ArchiveStorage()`
	`RetrieveStorage()`
	`CleanupStoragePath()`

Operating state (Page 90)	`PowerOn()`
	`PowerOff()`
	`Run()`
	`Stop()`
	`GetOperatingState()`
	`MemoryReset()`

Tag list (Page 101)	`UpdateTagList()`
	`GetTagListStatus()`
	`GetTagInfoCount()`
	`GetTagInfos()`
	`CreateConfigurationFile()`

I/O access via address - Reading (Page 107)	`GetAreaSize()`
	`ReadBit()`
	`ReadByte()`
	`ReadBytes()`
	`ReadSignals()`

I/O access via address - Writing (Page 113)	`WriteBit()`
	`WriteByte()`
	`WriteBytes()`
	`WriteSignals()`

I/O access via tag name - Reading (Page 118)	`Read()`
	`ReadBool()`
	`ReadChar(), ReadWChar()`
	`ReadDouble()`
	`ReadFloat()`
	`ReadInt8(), ReadInt16(), ReadInt32(), ReadInt64()`
	`ReadUInt8(), ReadUInt16(), ReadUInt32(), ReadUInt64()`
	`ReadSignals()`
	`ReadString()`
	`ReadWString()`

I/O access via tag name - Writing (Page 139)	`Write()`
	`WriteBool()`
	`WriteChar(), WriteWChar()`
	`WriteDouble()`
	`WriteFloat()`
	`WriteInt8(), WriteInt16(), WriteInt32(), WriteInt64(),`
	`WriteUInt8(), WriteUInt16(), WriteUInt32(), WriteUInt64()`
	`WriteSignals()`
	`WriteString()`
	`WriteWString()`

Settings	Functions

Virtual time (Page 161)	`GetSystemTime()`
	`SetSystemTime()`
	`GetScaleFactor()`
	`SetScaleFactor()`

Cycle control (Page 163)	`GetOperatingMode()`
	`SetOperatingMode()`
	`SetSendSyncEventInDefaultModeEnabled()`
	`IsSendSyncEventInDefaultModeEnabled`
	`GetOverwrittenMinimalCycleTime_ns()`
	`SetOverwrittenMinimalCycleTime_ns()`
	`RunToNextSyncPoint()`
	`StartProcessing()`
	`SetCycleTimeMonitoringMode()`
	`GetCycleTimeMonitoringMode()`

Acyclic services (Page 172)	

Table 2-8 Overview of IInstance events - Native C++

Events	Functions

OnOperatingStateChanged (Page 191)	`RegisterOnOperatingStateChangedCallback()`
	`UnregisterOnOperatingStateChangedCallback()`
	`RegisterOnOperatingStateChangedEvent()`
	`UnregisterOnOperatingStateChangedEvent()`
	`WaitForOnOperatingStateChangedEvent()`

OnLedChanged (Page 195)	`RegisterOnLedChangedCallback()`
	`UnregisterOnLedChangedCallback()`
	`RegisterOnLedChangedEvent()`
	`UnregisterOnLedChangedEvent()`
	`WaitForOnLedChangedEvent()`

OnSoftwareConfigurationChanged (Page 197)	`RegisterOnSoftwareConfigurationChangedCallback()`
	`UnregisterOnSoftwareConfigurationChangedCallback()`
	`RegisterOnSoftwareConfigurationChangedEvent()`
	`UnregisterOnSoftwareConfigurationChangedEvent()`
	`WaitForOnSoftwareConfigurationChangedEvent()`

OnSyncPointReached (Page 201)	`RegisterOnSyncPointReachedCallback()`
	`UnregisterOnSyncPointReachedCallback()`
	`RegisterOnSyncPointReachedEvent()`
	`UnregisterOnSyncPointReachedEvent()`
	`WaitForOnSyncPointReachedEvent()`

OnHardwareConfigChanged (Page 203)	`RegisterOnHardwareConfigChangedCallback()`
	`UnregisterOnHardwareConfigChangedCallback()`
	`RegisterOnHardwareConfigChangedEvent()`
	`UnregisterOnHardwareConfigChangedEvent()`
	`WaitForOnHardwareConfigChangedEvent()`

API IRemoteRuntimeManager

Table 2-9 Overview of IRemoteRuntimeManager functions - Native C++

Settings	Functions

Interface (Page 215)	`GetIP()`
	`GetRemoteComputerName()`
	`Disconnect()`

Table 2-10 Overview of IRemoteRuntimeManager events - Native C++

Events	Functions

OnConnectionLost (Page 217)	`RegisterOnConnectionLostCallback()`
	`UnregisterOnConnectionLostCallback()`
	`RegisterOnConnectionLostEvent()`
	`UnregisterOnConnectionLostEvent()`
	`WaitForOnConnectionLostEvent()`

2.8 Overview of interfaces for managed code

Initializing and shutting down API

Table 2-11 Overview of initializing and shutting down API - .NET (C#)

Actions	Functions
Initialize API (Page 37)	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeManager`
Shut down API (Page 42)	

API IBaseRuntimeManager

Table 2-12 Overview of IBaseRuntimeManager functions - .NET (C#)

Settings	Functions

Interface (Page 54)	`Version { get; }`
	`Port { get; }`
	`StrictMotionTiming { get; set; }`
	`NetworkMode { get; set; }`
	`NetInterfaces { get; }`
	`SetNetInterfaceBindings();`
	`ResetNetInterfaceBindings();`
	`DefaultStoragePath { get; set; }`

Simulation Runtime instances (Page 48)	`RegisteredInstanceInfo { get; }`
	`RegisterInstance();`
	`RegisterCustomInstance();`
	`CreateInterface();`

API ISimulationRuntimeManager

Table 2-13 Overview of ISimulationRuntimeManager functions - .NET (C#)

Settings	Functions

Interface (Page 70)	`IsInitialized { get; }`
	`IsRuntimeManagerAvailable { get; }`
	`Shutdown()`
	`StrictMotionTiming { get; }`

Remote connections (Page 62)	`OpenPort()`
	`ClosePort()`
	`Port { get; }`
	`RemoteConnectionInfo { get; }`
	`RemoteConnect()`
	`RunAutodiscover()`

Table 2-14 Overview of ISimulationRuntimeManager events - .NET (C#)

Events	Functions
OnSoftwareConfigurationChanged	`UnregisterOnSoftwareConfigurationChangedEvent()` `WaitForOnSoftwareConfigurationChangedEvent()`

OnRuntimeManagerLost (Page 68)	`OnRuntimeManagerLost()`
	`RegisterOnRuntimeManagerLostEvent()`
	`UnregisterOnRuntimeManagerLostEvent()`
	`WaitForOnRuntimeManagerLostEvent()`

OnAutodiscover (Page 70)	`OnAutodiscoverData()`

API IInstance

Table 2-15 Overview of IInstance functions - .NET (C#)

Settings	Functions

Interface (Page 72)	`Dispose ()`
	`ID { get; }`
	`Name { get; }`
	`CommunicationInterface { get; }`
	`CPUType { get; set; }`
	`Info { get; }`
	`UnregisterInstance()`
	`StrictMotionTiming { get; set; }`

Controller - Information and settings (Page 80)	`ControllerName { get; }`
	`ControllerShortDesignation { get; }`
	`ControllerIPSuite4 { get; }`
	`SetIPSuite()`
	`StoragePath { get; set; }`
	`ArchiveStorage()`
	`RetrieveStorage()`
	`CleanupStoragePath()`

Operating state (Page 90)	`PowerOn()`
	`PowerOff()`
	`Run()`
	`Stop()`
	`OperatingState { get; }`
	`MemoryReset()`

Tag list (Page 101)	`UpdateTagList()`
	`GetTagListStatus()`
	`TagInfos { get; }`
	`CreateConfigurationFile()`

I/O access via address - Reading (Page 107)	`InputArea | MarkerArea | OutputArea { get; }`
	`AreaSize { get; }`
	`ReadBit()`
	`ReadByte()`
	`ReadBytes()`
	`ReadSignals()`

I/O access via address - Writing (Page 113)	`WriteBit()`
	`WriteByte()`
	`WriteBytes()`
	`WriteSignals()`

I/O access via tag name - Reading (Page 118)	`Read()`
	`ReadBool()`
	`ReadChar(), ReadWChar()`
	`ReadDouble()`
	`ReadFloat()`
	`ReadInt8(), ReadInt16(), ReadInt32(), ReadInt64()`
	`ReadUInt8(), ReadUInt16(), ReadUInt32(), ReadUInt64()`
	`ReadSignals()`
	`ReadString()`

Settings	Functions
	`ReadWString()`

I/O access via tag name - Writing (Page 139)	`Write()`
	`WriteBool()`
	`WriteChar(), WriteWChar()`
	`WriteDouble()`
	`WriteFloat()`
	`WriteInt8(), WriteInt16(), WriteInt32(), WriteInt64(),`
	`WriteUInt8(), WriteUInt16(), WriteUInt32(), WriteUInt64()`
	`WriteSignals()`
	`WriteString()`
	`WriteWString()`

Virtual time (Page 161)	`SystemTime { get; set; }` `ScaleFactor { get; set; }`

Cycle control (Page 163)	`OperatingMode { get; set; }`
	`IsSendSyncEventInDefaultModeEnabled { get; set; }`
	`OverwrittenMinimalCycleTime_ns { get; set; }`
	`RunToNextSyncPoint()`
	`StartProcessing()`
	`SetCycleTimeMonitoringMode()`
	`GetCycleTimeMonitoringMode()`

Acyclic services (Page 172)	

Table 2-16 Overview of IInstance events - .NET (C#)

Events	Functions

OnOperatingStateChanged (Page 191)	`OnOperatingStateChanged()`
	`RegisterOnOperatingStateChangedEvent()`
	`UnregisterOnOperatingStateChangedEvent()`
	`WaitForOnOperatingStateChangedEvent()`

OnLedChanged (Page 195)	`OnLedChanged()`
	`RegisterOnLedChangedEvent()`
	`UnregisterOnLedChangedEvent()`
	`WaitForOnLedChangedEvent()`

OnSoftwareConfigurationChanged	`UnregisterOnSoftwareConfigurationChangedEvent()` `WaitForOnSoftwareConfigurationChangedEvent()`

OnSyncPointReached (Page 201)	`OnSyncPointReached()`
	`RegisterOnSyncPointReachedEvent()`
	`UnregisterOnSyncPointReachedEvent()`
	`WaitForOnSyncPointReachedEvent()`

OnHardwareConfigChanged (Page 203)	`OnHardwareConfigChanged()`
	`RegisterOnHardwareConfigChangedEvent()`
	`UnregisterOnHardwareConfigChangedEvent()`
	`WaitForOnHardwareConfigChangedEvent()`

API IRemoteRuntimeManager

Table 2-17 Overview of IRemoteRuntimeManager functions - .NET (C#)

Settings	Functions

Interface (Page 215)	`Dispose()`
	`IP { get; }`
	`RemoteComputerName { get; }`
	`Disconnect()`
	`StrictMotionTiming { get; }`

Table 2-18 Overview IRemoteRuntimeManager events - .NET (C#)

Events	Functions

OnConnectionLost() (Page 217)	`OnConnectionLost()`
	`RegisterOnConnectionLostEvent()`
	`UnregisterOnConnectionLostEvent()`
	`WaitForOnConnectionLostEvent()`

2.9 Restrictions for file paths

The following restrictions apply for user interfaces which expect a path or a complete file name as the transfer parameter:

Restrictions for local paths	Write permissions to system-critical folders such as the Windows folder (%Windows%) or the program folders (%Program Files%, %Program Files (x86)%) are not allowed. In this case the C++ user interface returns the error code `SREC_WRONG_ARGUMENT`. In this case the managed user interface returns an exception with the error code `RuntimeErrorCode.WrongArgument`.
Restrictions for network paths	In order to be able to use network paths, you must incorporate them as a network drive. Otherwise, the C++ user interface returns the error code `SREC_WRONG_ARGUMENT`. The managed user interface returns an exception with the error code `RuntimeErrorCode.WrongArgument`.

2.10 Overview of data types for native C++

The following table shows which data types are available for the simulation in Runtime Manager.

Table 2-19 Overview of data types - Native C++

Data type	
DLL import functions (Page 220)	`ApiEntry_Initialize` `ApiEntry_DestroyInterface`

Event callback functions (Page 221)	`EventCallback_VOID`
	`EventCallback_SRCC_UINT32_UINT32_INT32`
	`EventCallback_SRRSI_AD`
	`EventCallback_IRRTM`
	`EventCallback_II_SREC_ST_SROS_SROS`
	`EventCallback_II_SREC_ST_UINT32_INT64_INT64_UINT32`
	`EventCallback_II_SREC_ST`
	`EventCallback_II_SREC_ST_SRICC_UINT32_UINT32_UINT32_UINT32`
	`EventCallback_II_SREC_ST_SRLT_SRLM`
	`EventCallback_II_SREC_ST_SDRI`
	`EventCallback_II_SREC_ST_SDRI_BYTE`
	`EventCallback_II_SREC_ST_UINT32_UINT32`
	`EventCallback_II_SREC_ST_UINT32_UINT32_EPET_UINT32`
	`EventCallback_II_SREC_ST_UINT32_EPPET_UINT32`
	`EventCallback_II_SREC_ST_UINT32_ERSFET`
	`EventCallback_II_SREC_ST_UINT32`
	`EventCallback_II_SSCCP`

Definitions and constants (Page 237)	

Data type	
Unions (Page 238)	UIP UDataValue
Structures (Page 240)	SDataValue SDVBNI SDataValueByAddress SDataValueByName SConnectionInfo SInstanceInfo SDimension STagInfo SIP SIPSuite4 SOnSyncPointReachedResult SDataValueByAddressWithCheck SDataValueByNameWithCheck SDataRecordInfo SDataRecord SConfiguredProcessEvents SDiagExtChannelDescription SAutodiscoverData SOnSoftwareConfigChangedParameter
Enumerations (Page 255)	ERuntimeErrorCode EArea ECommunicationInterface EOperatingState EOperatingMode ECPUType ENetworkMode EPLCInterface ELEDType ELEDMode EPrimitiveDataTypes EDataType ETagListDetails ERuntimeConfigChanged EInstanceConfigChanged EPullOrPlugEventType EProcessEventType EDirection EDiagProperty EDiagSeverity ERackOrStationFaultType ECycleTimeMonitoringMode EAutodiscoverType

2.11 Overview of data types for managed code

The following table shows which data types are available for the simulation in Runtime Manager.

Table 2-20 Overview of data types - .NET (C#)

Data type	
Delegate definitions (Page 229) - Event handler methods	Delegate_Void Delegate_SRCC_UINT32_UINT32_INT32 Delegate_SRRSI_AD Delegate_IRRTM Delegate_II_EREC_DT_EOS_EOS Delegate_II_EREC_DT_UINT32_INT64_INT64_UINT32 Delegate_II_EREC_DT Delegate_II_EREC_DT_SRICC_UINT32_UINT32_UINT32_UINT32 Delegate_II_EREC_DT_ELT_ELM Delegate_II_EREC_DT_SDRI Delegate_II_EREC_DT_SDR

Data type	

	`Delegate_SREC_ST_UINT32_UINT32`
	`Delegate_SREC_ST_UINT32_UINT32_EPET_UINT32`
	`Delegate_SREC_ST_UINT32_EPPET_UINT32`
	`Delegate_SREC_ST_UINT32_ERSFET`
	`Delegate_SREC_ST_UINT32`

Definitions and constants (Page 237)	

Structures (Page 240)	`SDataValue`
	`SDVBNI`
	`SDataValueByAddress`
	`SDataValueByName`
	`SConnectionInfo`
	`SInstanceInfo`
	`SDimension`
	`STagInfo`
	`SIP`
	`SIPSuite4`
	`SOnSyncPointReachedResult`
	`SDataValueByAddressWithCheck`
	`SDataValueByNameWithCheck`
	`SDataRecordInfo`
	`SDataRecord`
	`SConfiguredProcessEvents`
	`SDiagExtChannelDescription`
	`SAutodiscoverData`

Enumerations (Page 255)	`ERuntimeErrorCode`
	`EArea`
	`ECommunicationInterface`
	`EOperatingState`
	`EOperatingMode`
	`ECPUType`
	`ELEDType`
	`ELEDMode`
	`ENetworkMode`
	`EPrimitiveDataTypes`
	`EDataType`
	`ETagListDetails`
	`ERuntimeConfigChanged`
	`EInstanceConfigChanged`
	`EPullOrPlugEventType`
	`EProcessEventType`
	`EDirection`
	`EDiagProperty`
	`EDiagSeverity`
	`ERackOrStationFaultType`
	`ECycleTimeMonitoringMode`
	`EAutodiscoverType`

Initialize API

3

3.1 Load API library

Description

For S7-PLCSIM Advanced, the interfaces of the API V6.0 are not compatible with the interfaces of previous API versions. However, the Runtime Manager of S7-PLCSIM Advanced V6.0 is compatible with the API of previous S7-PLCSIM Advanced versions.

Earlier versions of the API are also installed during the installation of S7-PLCSIM Advanced V6.0.

The default path is:

- C:\Program Files (x86)\Common Files\Siemens\PLCSIMADV\API\1.0
- C:\Program Files (x86)\Common Files\Siemens\PLCSIMADV\API\2.0
- C:\Program Files (x86)\Common Files\Siemens\PLCSIMADV\API\2.1
- C:\Program Files (x86)\Common Files\Siemens\PLCSIMADV\API\3.0
- C:\Program Files (x86)\Common Files\Siemens\PLCSIMADV\API\4.0
- C:\Program Files (x86)\Common Files\Siemens\PLCSIMADV\API\4.1
- C:\Program Files (x86)\Common Files\Siemens\PLCSIMADV\API\5.0
- C:\Program Files (x86)\Common Files\Siemens\PLCSIMADV\API\6.0

NOTE

API versions

To access and use the latest features and bug fixes, use the latest API version.

The installation path of S7-PLCSIM Advanced is contained in the registry:

- Key: "HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Siemens\Shared Tools\PLCSIMADV_SimRT"
- Value: "Path"

To maintain the path to the API, add the character string of the following subdirectory at the end: "API\<API version>" (e.g. "API\6.0").

When you use this path the API library (DLL) is loaded directly from the installation directory.

Reference

Additional information can be found in:

- For Native C++ in section InitializeApi() (Page 34).
- For .NET via the call of the function "System.Reflection.Assembly.LoadFile(string)" in the online documentation for MSDN.

3.2 Native C++

3.2.1 InitializeApi()

Description

The `InitializeApi` function loads the API library (DLL) and initializes the API. The function loads the version of the DLL that is compatible with the architecture of your application and with the header file of the API ("SimulationRuntimeApi.h").

To load the DLL, the function `InitializeApi` searches in the following folders one after the other:

- Folder to which the parameter of the function leads (`in_SimulationRuntimeApiDllPath`)
- Folder, which also contains your application that calls this function
- Installation directory of S7-PLCSIM Advanced

If no DLL is available, the function accesses the next folder.

The function returns an interface to the Simulation Runtime Manager. Use this interface to create a new simulated PLC instance or to obtain access to an existing instance.

Table 3-1 InitializeApi() - Native C++

Syntax	`ERuntimeErrorCode InitializeApi(ISimulationRuntimeManager** out_SimulationRuntimeManagerInterface);` `ERuntimeErrorCode InitializeApi(WCHAR* in_SimulationRuntimeApiDllPath,ISimulationRuntimeManager** inout_SimulationRuntimeManagerInterface);`	
Parameters	• `ISimulationRuntimeManager** out_SimulationRuntimeManagerInterface`: Pointer to a Runtime Manager interface pointer. The pointer must be initialized with `NULL`. The interface is created within the function. See Data types (Page 220). • `WCHAR* in_SimulationRuntimeApiDllPath`: The path to the Runtime API library.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_WRONG_ARGUMENT`	The pointer to the Runtime Manager interface does not equal `NULL`.

Return values	SREC_WRONG_VERSION	• The required version of the interface is incompatible with the version used to compile the API. • The version of the API is not compatible with Runtime. See the topic on compatibility in the S7-PLCSIM Advanced Function Manual.
	SREC_CONNECTION_ERROR	Unable to establish a connection to the Runtime Manager.
	SREC_ERROR_LOADING_DLL	The API library cannot be loaded.
	SREC_RUNTIME_NOT_AVAILABLE	No Runtime Manager runs in this Windows user session.
	SREC_CONFIG_FILE_ERROR	Operation regarding the configuration file "UserInterfaceConfiguration.xml" has failed, for example, create, read, write.

Example C++	`// Include The Headerfile Of The API`
	`#include "SimulationRuntimeApi.h"`
	
	`// Prepare The Variables`
	`ERuntimeErrorCode result = SREC_INVALID_ERROR_CODE;`
	`ISimulationRuntimeManager* api = NULL;`
	
	`// Initialize The API And Get The RuntimeManager Interface`
	`result = InitializeApi(&api);`	

NOTE

If you no longer require the interface, delete the interface.

See DestroyInterface() (Page 38).

3.2.2 RuntimeApiEntry_Initialize

Description

Use the function `RuntimeApiEntry_Initialize` only if the API library (DLL) is to be loaded from a different folder than the folder of your application calling this function.

When the API is initialized, the API library is first loaded and the `Initialize` function is then imported and called.

The function returns an interface to the Simulation Runtime Manager. Use this interface to create a new simulated PLC instance to obtain access to an existing instance.

Table 3-2 RuntimeApiEntry_Initialize - Native C++

Syntax	`__declspec(dllexport) ERuntimeErrorCode RuntimeApiEntry_Initialize(ISimulationRuntimeManager** out_SimulationRuntimeManagerInterface,UINT32 in_InterfaceVersion);`	
Parameters	• `ISimulationRuntimeManager** out_SimulationRuntimeManagerInterface:` Pointer to a Runtime Manager interface pointer. The pointer must be initialized with `NULL`. The interface is created within the function. See Data types (Page 220). • `UINT32 in_InterfaceVersion:` Version of the API interface to be downloaded: `DAPI_DLL_INTERFACE_VERSION`.	
Return values	Runtime error code	Description
	SREC_OK	The function is successful.

Return values	SREC_WRONG_ARGUMENT	The pointer to the Runtime Manager interface does not equal NULL.
	SREC_WRONG_VERSION	• The required version of the interface is incompatible with the version used to compile the API. • The version of the API is not compatible with Runtime. See information about compatibility in the S7-PLCSIM Advanced Function Manual.
	SREC_CONNECTION_ERROR	Unable to establish a connection to the Runtime Manager.
	SREC_RUNTIME_NOT_AVAILABLE	No Runtime Manager runs in this Windows user session.
	SREC_CONFIG_FILE_ERROR	Operation regarding the configuration file "UserInterfaceConfiguration.xml" has failed, for example, create, read, write.

Example C++	`// Include The Headerfile Of The API`
	`#include "SimulationRuntimeApi.h"`
	
	`// Prepare The Variables`
	`ERuntimeErrorCode result = SREC_INVALID_ERROR_CODE;`
	`HMODULE dllHandle = NULL;`
	`ApiEntry_Initialize Initialize = NULL;`
	`ISimulationRuntimeManager* api = NULL;`
	
	`// Load The DLL And Import The "Initialize" Function (using the Windows API)`
	`dllHandle = LoadLibrary(DAPI_DLL_NAME_X86);`
	`if (dllHandle != NULL)`
	`{`
	`Initialize = (ApiEntry_Initialize)GetProcAddress(dllHandle, DAPI_ENTRY_INITIALIZE);`
	`}`
	
	`// Initialize The API And Get The RuntimeManager Interface`
	`if ( Initialize != NULL )`
	`{`
	`result = Initialize(&api, DAPI_DLL_INTERFACE_VERSION);`
	`}`	

NOTE

If you no longer require the interface, delete the interface.
SeeDestroyInterface() (Page 38).

3.3 .NET (C#)

3.3.1 Initialize

Description

The entry point to the API is the static class `Siemens.Simatic.Simulation.Runtime.SimulationRuntimeManager`.
The API is initialized when a function of this class is used for the first time.

Table 3-3 Initialize - .NET (C#)

Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationInitializationException`	
	Runtime error code	Description
	`ERuntimeErrorCode.Connection-Error`	Unable to establish a connection to the Runtime Manager.
	`ERuntimeErrorCode.WrongVer-sion`	The version of the API is not compatible with Runtime. See information about compatibility in the S7-PLCSIM Advanced Function Manual.
	`ERuntimeErrorCode.RuntimeNo-tAvailable`	No Runtime Manager runs in this Windows user session.
	`ERuntimeErrorCode.Config-FileError`	Operation regarding the configuration file "UserInterfaceConfiguration.xml" has failed, for example, create, read, write.

Shut down API

4

4.1 Native C++

Basic procedure for deleting the user interfaces

To delete all user interfaces, generally follow these steps:

1. Delete the interfaces IInstances and IRemoteRuntimeManager.
2. Call the `Shutdown()` function of the ISimulationRuntimeManager interface.
3. Delete the ISimulationRuntimeManager interface.
4. Unload the API library (DLL) with the Window API function `FreeLibrary()`.

Deleting the user interfaces via functions

Deleting the user interfaces is also possible via functions.

If the API was initialized using the `InitializeApi()` function, you delete the user interfaces using the following functions:

- FreeApi() (Page 40)
- ShutdownAndFreeApi() (Page 41)

4.1.1 DestroyInterface()

Description

`DestroyInterface()` is a function pointer that points to the `RuntimeApiEntry_DestroyInterface` function. It is only valid if you have successfully called the `InitializeApi` function.

The function unloads the memory of an ISimulationRuntimeManager, IRemoteRuntimeManager or IInstance interface.

If you call `DestroyInterface()` on an instance and the number of interfaces reaches zero, the instance becomes unregistered.

Table 4-1 DestroyInterface() - Native C++

Syntax	`ERuntimeErrorCode DestroyInterface(IBaseInterface* in_Interface);`	
Parameters	• `IBaseInterface* in_Interface:` The interface to be deleted.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_WRONG_ARGUMENT`	The pointer to the interface is `NULL`.

Example C++	`// Include The Headerfile Of The API`
	`#include "SimulationRuntimeApi.h"`
	
	`// The Interfaces`
	`ERuntimeErrorCode result;`
	`ISimulationRuntimeManager* api = NULL;`
	`IInstance* instance = NULL;`
	
	`// Init the DLL and create an instance`
	`result = InitializeApi(&api);`
	`result = api->RegisterInstance(&instance);`
	
	`// Destroy Instance Interfaces`
	`result = DestroyInterface(instance);`
	`instance = NULL;`	

4.1.2 RuntimeApiEntry_DestroyInterface

Description

Use the `RuntimeApiEntry_DestroyInterface` function only if the API library (DLL) is to be loaded from a different directory than the Startup folder of the application calling this function.

If the API was initialized using the `InitializeApi` function, you select the DestroyInterface() (Page 38) function.

The function unloads the memory of an ISimulationRuntimeManager, IRemoteRuntimeManager or IInstance interface.

Table 4-2 RuntimeApiEntry_DestroyInterface() - Native C++

Syntax	`__declspec(dllexport) ERuntimeErrorCode RuntimeApiEntry_DestroyInterface(IBaseInterface* in_Interface);`	
Parameters	• `IBaseInterface* in_Interface:` The interface to be deleted.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_WRONG_ARGUMENT`	The pointer to the interface is `NULL`.

Example C++	`// Include The Headerfile Of The API`
	`#include "SimulationRuntimeApi.h"`
	
	`// Prepare The Variables`
	`ERuntimeErrorCode result = SREC_INVALID_ERROR_CODE;`
	`HMODULE dllHandle = NULL;`
	`ApiEntry_DestroyInterface Destroy = NULL;`
	`IInstance* instance = NULL;`
	
	`// Load The DLL And Import The "DestroyInterface" Function (using the Windows API)`
	`dllHandle = LoadLibraryA(DAPI_DLL_NAME_X86);`
	`if ( dllHandle != NULL )`
	`{`
	`Destroy = (ApiEntry_ DestroyInterface)GetProcAddress(dllHandle,`
	` DAPI_ENTRY_DESTROY_INTERFACE);`
	`}`
	…
	`// Frees the memory of an IInstance interface`
	`result = Destroy(instance);`	

4.1.3 FreeApi()

Description

The `FreeApi()` function unloads the library of the Runtime API.

This function can only be called after the successful call of the `InitializeApi` If the `InitializeApi` function was not called, the library must be unloaded using the Windows API function `FreeLibrary()`.

Table 4-3 FreeApi() - Native C++

Syntax	`ERuntimeErrorCode FreeApi();`	
Parameters	None	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_API_NOT_INITIALIZED`	The `InitializeApi` function was not called successfully.

Example C++	`// Include The Headerfile Of The API`
	`#include "SimulationRuntimeApi.h"`
	
	`// The Interfaces`
	`ERuntimeErrorCode result;`
	`ISimulationRuntimeManager* api = NULL;`
	`IInstance* instance = NULL;`
	
	`// Init the API`
	`result = InitializeApi(&api);`
	
	…
	
	`// Shutdown The API`
	`api->Shutdown();`
	`result = DestroyInterface(api);`
	`api = NULL;`
	`result = FreeApi();`	

4.1.4 ShutdownAndFreeApi()

Description

The `ShutdownAndFreeApi()` function shuts down the Runtime API, deletes the IRuntimeManager interface and unloads the library of the Runtime API.

This function can only be called after the successful call of the `InitializeApi` If the `InitializeApi` function was not called, the library must be unloaded using the Windows API function `FreeLibrary()`.

Table 4-4 ShutdownAndFreeApi() - Native C++

Syntax	`ERuntimeErrorCode ShutdownAndFreeApi(ISimulationRuntimeManager* in_SimulationRuntimeManagerInterface);`	
Parameters	• `ISimulationRuntimeManager* in_SimulationRuntimeManagerInterface`: The interface of the Runtime Manager to be deleted.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_API_NOT_INITIALIZED`	The `InitializeApi` function was not called successfully.
	`SREC_WRONG_ARGUMENT`	The pointer to the interface is `NULL`.

Example C++	`// Include The Headerfile Of The API`
	`#include "SimulationRuntimeApi.h"`
	
	`// The Interfaces`
	`ERuntimeErrorCode result;`
	`ISimulationRuntimeManager* api = NULL;`
	`IInstance* instance = NULL;`
	
	`// Init the API`
	`result = InitializeApi(&api);`
	
	…
	
	`// Shutdown The API`
	`result = ` **`ShutdownAndFreeApi`**`(api);`
	`api = NULL;`	

4.2 .NET (C#)

4.2.1 Shut down API

You can terminate the .NET components of the API for the IInstance and IRemoteRuntimeManager interfaces by calling the Dispose (Page 72) function.

These interfaces can also be cleared automatically with the .NET Garbage Collector.

Manually clearing the API

To manually clear the API, follow these steps:

1. Delete all interfaces.

   You can find more information at Interfaces for IInstances (Page 72).

2. Call the Shutdown() (Page 70) function of the ISimulationRuntimeManager interface.

Global functions (Native C++)

5

The global functions `GetNameOf...` return the name of the enumeration entry (`const WCHAR*`).

GetNameOfAreaSection()

Table 5-1 GetNameOfAreaSection() - Native C++

Syntax	`const WCHAR* GetNameOfAreaSection(EArea in_AreaSection);`
Parameters	`EArea in_AreaSection`: Enumeration entry
Return values	`const WCHAR*`: Name of the enumeration entry

GetNameOfCPUType()

Table 5-2 GetNameOfCPUType() - Native C++

Syntax	`const WCHAR* GetNameOfCPUType(ECPUType in_CPUType);`
Parameters	`ECPUType in_CPUType`: Enumeration entry
Return values	`const WCHAR*`: Name of the enumeration entry

GetNameOfCommunicationInterface()

Table 5-3 GetNameOfCommunicationInterface() - Native C++

Syntax	`const WCHAR* GetNameOfCommunicationInterface(ECommunicationInterface in_CommunicationInterface);`
Parameters	`ECommunicationInterface in_CommunicationInterface`: Enumeration entry
Return values	`const WCHAR*`: Name of the enumeration entry

GetNameOfDataType()

Table 5-4 GetNameOfDataType() - Native C++

Syntax	`const WCHAR* GetNameOfDataType(EDataType in_DataType);`
Parameters	`EDataType in_DataType`: Enumeration entry
Return values	`const WCHAR*`: Name of the enumeration entry

GetNameOfErrorCode()

Table 5-5 GetNameOfErrorCode() - Native C++

Syntax	`const WCHAR* GetNameOfErrorCode(ERuntimeErrorCode in_ErrorCode);`
Parameters	`ERuntimeErrorCode in_ErrorCode`: Enumeration entry
Return values	`const WCHAR*`: Name of the enumeration entry

GetNameOfLEDMode()

Table 5-6 GetNameOfLEDMode() - Native C++

Syntax	`const WCHAR* GetNameOfLEDMode(ELEDMode in_LEDMode);`
Parameters	`ELEDMode in_LEDMode`: Enumeration entry
Return values	`const WCHAR*`: Name of the enumeration entry

GetNameOfLEDType()

Table 5-7 GetNameOfLEDType() - Native C++

Syntax	`const WCHAR* GetNameOfLEDType(ELEDType in_LEDType);`
Parameters	`ELEDType in_LEDType`: Enumeration entry.
Return values	`const WCHAR*`: Name of the enumeration entry

GetNameOfOperatingMode()

Table 5-8 GetNameOfOperatingMode() - Native C++

Syntax	`const WCHAR* GetNameOfOperatingMode(EOperatingMode in_OperatingMode);`
Parameters	`EOperatingMode in_OperatingMode`: Enumeration entry
Return values	`const WCHAR*`: Name of the enumeration entry

GetNameOfOperatingState

Table 5-9 GetNameOfOperatingState() - Native C++

Syntax	`const WCHAR* GetNameOfOperatingState(EOperatingState in_OperatingState);`
Parameters	`EOperatingState in_OperatingState`: Enumeration entry
Return values	`const WCHAR*`: Name of the enumeration entry

GetNameOfPrimitiveDataType

Table 5-10 GetNameOfPrimitiveDataType() - Native C++

Syntax	`const WCHAR* GetNameOfPrimitiveDataType(EPrimitiveDataType in_DataType);`
Parameters	`EPrimitiveDataType in_DataType`: Enumeration entry
Return values	`const WCHAR*`: Name of the enumeration entry

GetNameOfTagListDetails

Table 5-11 GetNameOfTagListDetails() - Native C++

Syntax	`const WCHAR* GetNameOfTagListDetails(ETagListDetails in_TagListDetails);`
Parameters	`ETagListDetails in_TagListDetails`: Enumeration entry
Return values	`const WCHAR*`: Name of the enumeration entry

GetNameOfRuntimeConfigChanged()

Table 5-12 GetNameOfRuntimeConfigChanged() - Native C++

Syntax	`const WCHAR* GetNameOfRuntimeConfigChanged(ERuntimeConfigChanged in_RuntimeConfigChanged);`
Parameters	`ERuntimeConfigChanged in_RuntimeConfigChanged`: Enumeration entry
Return values	`const WCHAR*`: Name of the enumeration entry

GetNameOfInstanceConfigChanged()

Table 5-13 GetNameOfInstanceConfigChanged() - Native C++

Syntax	`const WCHAR* GetNameOfInstanceConfigChanged(EInstanceConfigChanged in_InstanceConfigChanged);`
Parameters	`EInstanceConfigChanged in_InstanceConfigChanged`: Enumeration entry
Return values	`const WCHAR*`: Name of the enumeration entry

GetNameOfDirection()

Table 5-14 GetNameOfDirection() - Native C++

Syntax	`const WCHAR* GetNameOfDirection(EDirection in_Direction);`
Parameter	`EDirection in_Direction`: Enumeration entry
Return values	`const WCHAR*`: Name of the enumeration entry

GetNameOfDiagSeverity()

Table 5-15 GetNameOfDiagSeverity() - Native C++

Syntax	`const WCHAR* GetNameOfDiagSeverity(EDiagSeverity in_DiagSeverity);`
Parameter	`EDiagSeverity in_DiagSeverity`: Enumeration entry
Return values	`const WCHAR*`: Name of the enumeration entry

GetNameOfRackOrStationFaultType()

Table 5-16 GetNameOfRackOrStationFaultType() - Native C++

Syntax	`const WCHAR* GetNameOfRackOrStationFaultType(ERackOrStationFaultType in_RackOrStationFaultType);`
Parameter	`ERackOrStationFaultType in_RackOrStationFaultType`: Enumeration entry
Return values	`const WCHAR*`: Name of the enumeration entry

GetNameOfProcessEventType()

Table 5-17 GetNameOfProcessEventType() - Native C++

Syntax	`const WCHAR*(GetNameOfProcessEventType(EProcessEventType in_ProcessEventType);`
Parameters	`EProcessEventType in_ProcessEventType`: Enumeration entry
Return values	`const WCHAR*`: Name of the enumeration entry

GetNameOfPullOrPlugEventType()

Table 5-18 GetNameOfPullOrPlugEventType() - Native C++

Syntax	`const WCHAR* GetNameOfPullOrPlugEventType(EPullOrPlugEventType in_PullOrPlugEventType);`
Parameters	`EPullOrPlugEventType in_PullOrPlugEventType`: Enumeration entry
Return values	`const WCHAR*`: Name of the enumeration entry

GetNameOfCycleTimeMonitoringMode()

Table 5-19 GetNameOfCycleTimeMonitoringMode() - Native C++

Syntax	`const WCHAR* GetNameOfCycleTimeMonitoringMode(ECycleTimeMonitoringMode in_CycleTimeMonitoringMode);`
Parameters	`ECycleTimeMonitoringMode in_CycleTimeMonitoringMode`: Enumeration entry
Return values	`const WCHAR*`: Name of the enumeration entry

GetNameOfDiagProperty()

Table 5-20 GetNameOfDiagProperty() - Native C++

Syntax	`const WCHAR* GetNameOfDiagProperty(EDiagProperty in_DiagProperty);`
Parameters	`EDiagProperty in_DiagProperty`: Enumeration entry
Return values	`const WCHAR*`: Name of the enumeration entry

GetNameOfAutodiscoverType()

Table 5-21 GetNameOfAutodiscoverType() - Native C++

Syntax	`const WCHAR* GetNameOfAutodiscoverType(EAutodiscoverType in_AutodiscoverType);`
Parameters	`EAutodiscoverType in_AutodiscoverType`: Enumeration entry
Return values	`const WCHAR*`: Name of the enumeration entry

GetNameOfSoftwareConfigChanged()

Table 5-22 GetNameOfSoftwareConfigChanged() - Native C++

Syntax	`const WCHAR* GetNameOfSoftwareConfigChanged(ESoftwareConfigChanged in_SoftwareConfigChanged);`
Parameters	`ESoftwareConfigChanged in_SoftwareConfigChanged`: Enumeration entry
Return values	`const WCHAR*`: Name of the enumeration entry

GetNameOfNetworkMode()

Table 5-23 GetNameOfNetworkMode() - Native C++

Syntax	`const WCHAR* GetNameOfNetworkMode(ENetworkMode in_NetworkMode);`
Parameters	`ENetworkMode in_NetworkMode`: Enumeration entry
Return values	`const WCHAR*`: Name of the enumeration entry

IBaseRuntimeManager

6

6.1 Simulation runtime instances

CreateInterface()

Creates and returns an interface of an already registered instance of a simulated PLC.

The instance could have been registered via the application or another application that uses the Simulation Runtime API.

Table 6-1 CreateInterface() - Native C++

Syntax	`ERuntimeErrorCode CreateInterface(WCHAR* in_InstanceName, IInstance** out_InstanceInterface);` `ERuntimeErrorCode CreateInterface(INT32 in_InstanceID, IInstance** out_InstanceInterface);`	
Parameters	• `INT32 in_InstanceID:` The ID of the registered instance from which you want to receive the interface. • `WCHAR* in_InstanceName:` The name of the registered instance from which you want to receive the interface. • `IInstance** out_InstanceInterface:` Pointer to a Simulation Runtime interface pointer. The pointer must be initialized with `NULL`. The interface is created within the function.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The interface is disconnected from the remote Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_WRONG_ARGUMENT`	The name, the ID or the IInstance- pointer is invalid.
	`SREC_DOES_NOT_EXIST`	The instance is not registered in Runtime Manager.

Example C++	`ISimulationRuntimeManager * api = NULL;`
	`ERuntimeErrorCode result = InitializeApi(&api);`
	
	`IInstance* psa1 = NULL;`
	`IInstance* psa2 = NULL;`
	`if (result == SREC_OK)`
	`{`
	` result = api->`**`CreateInterface`**`(0, &psa1);`
	`result = api->`**`CreateInterface`**`(0, &psa2); // psa2 will be the same as psa1`
	`}`	

Example C++	`ISimulationRuntimeManager * api = NULL;`
	`ERuntimeErrorCode result = InitializeApi(&api);`
	
	`IInstance* psa = NULL;`
	`if (result == SREC_OK)`
	`{`
	` result = api->`**`CreateInterface`**`(L"My SimulationRuntime Instance", &psa);`
	`}`	

NOTE
Native C++
If you no longer require the interface, delete the interface.
See DestroyInterface() (Page 38)

Table 6-2 CreateInterface() - .NET (C#)

Syntax	`IInstance CreateInterface(string in_InstanceName);` `IInstance CreateInterface(INT32 in_InstanceID);`	
Parameters	• `INT32 in_InstanceID:` The ID of the registered instance from which you want to receive the interface. • `string in_InstanceName:` The name of the registered instance from which you want to receive the interface.	
Return values	If the function is successful, an interface of a simulated PLC instance; otherwise, a Null pointer.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The interface is disconnected from the remote Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.WrongArgument`	The name or the ID is invalid.
	`ERuntimeErrorCode.DoesNotExists`	The instance is not registered in Runtime Manager.

GetRegisteredInstanceInfoAt()

Returns information about an already registered instance.

You can use the ID or name to create an interface of this instance (see `CreateInterface()`).

Table 6-3 GetRegisteredInstanceInfoAt() - Native C++

Syntax	`ERuntimeErrorCode GetRegisteredInstanceInfoAt(UINT32 in_Index, SInstanceInfo* out_InstanceInfo);`	
Parameters	• `UINT32 in_Index:` Index of the created instance from which you want to receive the information. The index must be less than the value you receive when you call `GetRegisteredInstanceCount()`. • `SInstanceInfo* out_InstanceInfo:` The information with name and ID of the instance.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The interface is disconnected from the remote Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INDEX OUT_OF_RANGE`	There is no instance information for this index.

RegisteredInstanceInfo { get; }

Returns information about an already registered instance. You can use the ID or name of this instance to create an interface of this instance, see `CreateInterface()`.

Table 6-4 RegisterInstanceInfo { get; } - .NET (C#)

Syntax	`SInstanceInfo[] RegisteredInstanceInfo { get; }`	
Parameters	None	
Return values	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The interface is disconnected from the remote Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.

GetRegisteredInstancesCount()

Returns the number of instances that are registered in Runtime Manager. If the function fails, the return value is 0.

Table 6-5 GetRegisteredInstancesCount() - Native C++

Syntax	`UINT32 GetRegisteredInstancesCount();`
Parameters	None
Return values	`UINT32`: Number of available instances.

RegisterInstance()

Registers a new instance of a simulated PLC instance in Runtime Manager. Creates and returns an interface of this instance.

Table 6-6 RegisterInstance() - Native C++

Syntax	`ERuntimeErrorCode RegisterInstance(IInstance** out_InstanceInterface);`
	`ERuntimeErrorCode RegisterInstance(WCHAR* in_InstanceName, IInstance** out_InstanceInterface);`
	`ERuntimeErrorCode RegisterInstance(ECPUType in_CPUType, IInstance** out_InstanceInterface);`
	`ERuntimeErrorCode RegisterInstance(ECPUType in_CPUType, WCHAR* in_InstanceName, IInstance** out_InstanceInterface);`
	`ERuntimeErrorCode RegisterInstance(WCHAR* in_ArticleNumber, WCHAR* in_InstanceName, IInstance** out_InstanceInterface);`

Parameters	• `ECPUType in_CPUType`: Defines which CPU type is simulated at the start of the instance. The default setting is `"SRCT_1500_Unspecified"`. When a different CPU type is loaded via STEP 7 or from the Virtual SIMATIC Memory Card, this CPU type applies. • `WCHAR* in_InstanceName`: Name to be assigned to the instance. Every instance must have a unique name. If no name is assigned when registering a new instance, the instance is given the name `"Instance_#"` (# is the ID of the instance). If this name already exists, the name `"Instance_#.#"` is used, in which the second # is a counter that is incremented until the name is unique. The length of the name must be less than `DINSTANCE_NAME_LENGTH` and must not include Unicode characters. See Data types (Page 220).

	• `IInstance** out_InstanceInterface`: Pointer to a Simulation Runtime interface pointer. The pointer **must** be initialized with `NULL`. The interface is created within the function. • `WCHAR* in_ArticleNumber`: Article number for a specific PLC device. The article number must be 19 characters long.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_CONFIG_FILE_ERROR`	Error found in configuration file.
	`SREC_INTERFACE_REMOVED`	The interface is disconnected from the remote Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_WRONG_ARGUMENT`	One or more of the following parameters are invalid: • The instance name • The IInstance pointer • The length of `in_ArticleNumber`
	`SREC_LIMIT_REACHED`	There are already 16 instances registered in Runtime Manager.
	`SREC_ALREADY_EXISTS`	An instance with this name already exists.

Example C++	`ISimulationRuntimeManager * api = NULL;`
	`ERuntimeErrorCode result = InitializeApi(&api);`
	
	`// Example: How To Create And Register An Instance`
	`// And To Get An Interface Of The Instance The Same Time`
	`IInstance* psa = NULL;`
	`if (result == SREC_OK)`
	`{`
	`result = api->RegisterInstance(&psa);`
	`}`	

NOTE

Native C++

If you no longer require the interface, delete the interface.

See DestroyInterface() (Page 38).

Table 6-7 RegisterInstance() - .NET (C#)

Syntax	`IInstance RegisterInstance();`
	`IInstance RegisterInstance(string in_InstanceName);`
	`IInstance RegisterInstance(ECPUType in_CPUType);`
	`IInstance RegisterInstance(ECPUType in_CPUType string in_InstanceName);`
	`IInstance RegisterInstance(string in_ArticleNumber, string in_InstanceName);`

Parameters	• `ECPUType in_CPUType`: Defines which CPU type is simulated at the start of the instance. The default setting is `"ECPUType.Unspecified"`. When a different CPU type is loaded via STEP 7 or from the Virtual SIMATIC Memory Card, this CPU type applies. • `string in_InstanceName`: Name to be assigned to the instance. Every instance must have a unique name. If no name is assigned when registering a new instance, the instance is given the name `"Instance_#"` (# is the ID of the instance). If this name already exists, the name `"Instance_#.#"` is used, in which the second # is a counter that is incremented until the name is unique. The length of the name must be less than `DINSTANCE_NAME_LENGTH`. See Data types (Page 220).

	• `string in_ArticleNumber:` Article number for a specific PLC device. The article number must be 19 characters long.	
Return values	If the function is successful, an interface of a simulated PLC. Otherwise, a Null pointer.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.ConfigFileError`	Error found in configuration file.
	`ERuntimeErrorCode.InterfaceRemoved`	The interface is disconnected from the remote Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.WrongArgument`	One or more of the following parameters are invalid: • The instance name • The IInstance pointer • The length of `in_ArticleNumber`
	`ERuntimeErrorCode.LimitReached`	There are already 16 instances registered in Runtime Manager.
	`ERuntimeErrorCode.AlreadyExists`	An instance with this name already exists.

RegisterCustomInstance()

Registers a new simulated PLC instance in the Runtime Manager. Creates and returns an interface of this instance.

Table 6-8 RegisterCustomInstance() - Native C++

Syntax	`ERuntimeErrorCode RegisterCustomInstance(WCHAR* in_VplcDll, IInstance** out_InstanceInterface);` `ERuntimeErrorCode RegisterCustomInstance(WCHAR* in_VplcDll, WCHAR* in_InstanceName, IInstance** out_InstanceInterface);`	
Parameters	• `WCHAR* in_VplcDll:` The complete path to the DLL of the simulated PLC instance that Siemens.Simatic.Simulation.Runtime.Instance.exe loads at PowerOn. • `WCHAR* in_InstanceName:` Name to be assigned to the instance. Every instance must have a unique name. If no name is assigned when registering a new instance, the instance is given the name `"Instance_#"` (# is the ID of the instance). If this name already exists, the name `"Instance_#.#"` is used, in which the second # is a counter that is incremented until the name is unique. The length of the name must be less than `DINSTANCE_NAME_LENGTH`. See Data types (Page 220). • `IInstance** out_InstanceInterface:` Pointer to a Simulation Runtime interface pointer. The pointer must be initialized with `NULL`. The interface is created within the function.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The interface is disconnected from the remote Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_WRONG_ARGUMENT`	The DLL name, the instance name or the IInstance pointer is invalid.

Return values	SREC_LIMIT_REACHED	There are already 16 instances registered in Runtime Manager.
	SREC_ALREADY_EXISTS	An instance with this name already exists.

Example C++	`ISimulationRuntimeManager * api = NULL;`
	`ERuntimeErrorCode result = InitializeApi(&api);`
	
	`// Example: How To Create And Register An Instance`
	`// And To Get An Interface Of The Instance The Same Time`
	`IInstance* psa = NULL;`
	`if (result == SREC_OK)`
	`{`
	`result = api->`**`RegisterCustomInstance`**`(L"C:\\Temp\\vplc.dll", &psa);`
	`}`	

NOTE

Native C++

If you no longer require the interface, delete the interface.

See DestroyInterface() (Page 38).

Table 6-9 RegisterCustomInstance() - .NET (C#)

Syntax	`IInstance RegisterCustomInstance(string in_VplcDll);` `IInstance RegisterCustomInstance(string in_VplcDll, string in_InstanceName);`	
Parameters	• `string in_VplcDll:` The complete path to the DLL of the simulated PLC instance that Siemens.Simatic.Simulation.Runtime.Instance.exe loads at PowerOn. • `string in_InstanceName:` Name to be assigned to the instance. Every instance must have a unique name. If no name is assigned when registering a new instance, the instance is given the name "`Instance_#`" (# is the ID of the instance). If this name already exists, the name "`Instance_#.#`" is used, in which the second # is a counter that is incremented until the name is unique. The length of the name must be less than `DINSTANCE_NAME_LENGTH`. See Data types (Page 220).	
Return values	If the function is successful, an interface of a simulated PLC instance; otherwise, a Null pointer.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The interface is disconnected from the remote Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.WrongArgument`	The name or the ID is invalid.
	`ERuntimeErrorCode.LimitReached`	There are already 16 instances registered in Runtime Manager.
	`ERuntimeErrorCode.AlreadyExists`	An instance with this name already exists.

6.2 Interfaces for IBaseRuntimeManager

GetPort() / Port { get; }

Returns the open port of the PC on which the Runtime Manager is running. If the function fails, the return value is 0.

Table 6-10 GetPort() - Native C++

Syntax	`UINT16 GetPort();`
Parameters	None
Return values	`UINT16`: Open port of the PC on which the remote Runtime Manager is running.

Table 6-11 Port { get; } - .NET (C#)

Syntax	`UInt16 Port { get; }`
Parameters	None
Return values	`UInt16`: Open port of the PC on which the remote Runtime Manager is running.

GetVersion() / Version { get; }

Returns the version of the Runtime Manager. If the function fails, version 0.0 is returned.

Table 6-12 GetVersion() - Native C++

Syntax	`UINT32 GetVersion();`
Parameters	None
Return values	`UINT32`: Remote Runtime Manager Version (`HIWORD` = Major, `LOWORD` = Minor)

Table 6-13 Version { get; } - .NET (C#)

Syntax	`UInt32 Version { get; }`
Parameters	None
Return values	`Uint32`: Remote Runtime Manager Version (`HIWORD` = Major, `LOWORD` = Minor)

GetStrictMotionTiming() / StrictMotionTiming { get; }

Returns the current global setting for the "Strict Motion Timing" feature that has an effect on newly created instances. The "Strict Motion Timing" feature is located on the S7-PLCSIM Advanced Control Panel.

Table 6-14 GetStrictMotionTiming() - Native C++

Syntax	`ERuntimeErrorCode GetStrictMotionTiming(bool* enabled);`	
Parameters	bool* enabled: Receives the current setting. `true`: Active `false`: Inactive	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_CONFIG_FILE_ERROR`	The setting could not be read from the configuration file UserInterfaceConfiguration.xml.

Table 6-15 StrictMotionTiming { get; } - .NET (C#)

Syntax	`bool StrictMotionTiming { get; }`	
Parameters	None	
Return values	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.ConfigFileError`	The setting could not be read from the configuration file UserInterfaceConfiguration.xml.

SetStrictMotionTiming() / StrictMotionTiming { set; }

Sets the global setting for the "Strict Motion Timing" feature that has an effect on newly created instances. The "Strict Motion Timing" feature is located on the S7-PLCSIM Advanced Control Panel.

Table 6-16 SetStrictMotionTiming() - Native C++

Syntax	`ERuntimeErrorCode SetStrictMotionTiming(bool enable);`	
Parameters	bool enable: The value to be set. `true`: Active `false`: Inactive	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_ALREADY_EXISTS`	An instance is registered. No instance must be registered to change the setting.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_ACCESS_DENIED`	No write rights for the configuration file.
	`SREC_CONFIG_FILE_ERROR`	The setting could not be written to the configuration file UserInterfaceConfiguration.xml.

Table 6-17 StrictMotionTiming { set; } - .NET (C#)

Syntax	`bool StrictMotionTiming { set; }`	
Parameters	None	
Return values	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.AlreadyExists`	An instance is registered. No instance must be registered to change the setting.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.AccessDenied`	No write rights for the configuration file.
	`ERuntimeErrorCode.ConfigFileError`	The setting could not be written to the configuration file UserInterfaceConfiguration.xml.

GetNetworkMode

Returns the current network mode settings that are used in the network adapter for distributed communication.

Table 6-18 GetNetworkMode() - Native C++

Syntax	`ERuntimeErrorCode IBaseRuntimeManager::GetNetworkMode(ENetworkMode* out_Mode);`	
Parameters	`ENetworkMode* out_Mode` Receives the network mode as #`ENetworkMode` that is currently set in the driver of the S7-PLCSIM Advanced Virtual Switch	
Return values	Runtime error code	Description
	`SREC_OK`	The function call is successful.

SetNetworkMode()

Sets the network mode as used in the network adapter for distributed communication.

Check that the binding on the network interfaces is correct.

Table 6-19 SetNetworkMode() - Native C++

Syntax	`ERuntimeErrorCode SetNetworkMode(const ENetworkMode in_mode);`	
Parameters	`ENetworkMode in_Mode` Type of network communication mode to be set	
Return values	Runtime error code	Description
	`SREC_OK`	The function call is successful.
	`SREC_VIRTUAL_SWITCH_MISCONFIGURED`	The driver of the S7-PLCSIM Advanced Virtual Switch is not running.
	`SREC_INSTANCE_ALREADY_RUNNING`	Network mode cannot be changed because there is already a running instance.
	`SREC_PCAP_DRIVER_NOT_RUNNING`	The Nmap Packet Capture Driver (Npcap) is not active on the system. Remedy: 1. Start the command line in administrator mode. 2. Execute this command: net start npcap

NetworkMode { get; set; };

Returns or sets the network mode used on the network adapter for distributed communication.

Check that the binding on the network interfaces is correct.

Table 6-20 NetworkMode { get; } - .NET (C#)

Syntax	`ENetworkMode NetworkMode { get; };`
Return value	`ENetworkMode` Reveives the network mode as #`ENetworkMode` that is currently set in the driver of the S7-PLCSIM Advanced Virtual Switch
Exceptions	None

Table 6-21 NetworkMode { set; } - .NET (C#)

Syntax	`ENetworkMode NetworkMode { set; };`	
Parameters	None	
Exceptions	Runtime error code	Description
	`ERuntimeErrorCode.VirtualSwitchMis-configured`	The S7-PLCSIM Advanced Virtual Switch driver is not running.
	`ERuntimeErrorCode.InstanceAlreadyR-unning`	Network mode cannot be changed because there is already a running instance.
	`ERuntimeErrorCode.PCAPDriverNotRun-ning`	The Nmap Packet Capture Driver (Npcap) is not active on the system. Remedy: 1. Start the command line in administrator mode. 2. Execute this command: `net start npcap`

GetNetInterfaces() / NetInterfaces { get; }

Provides information on the PC interfaces, which you can use for `SetNetInterfaceMapping`.

`GetNetInterfaces()` does not return wireless adapters.

Table 6-22 GetNetInterfaces() - Native C++

Syntax	`ERuntimeErrorCode GetNetInterfaces(SNetInterfaceInfo* inout_InterfaceInfo, UINT* inout_InterfaceCount);`	

Parameters	`SNetInterfaceInfo* inout_InterfaceInfo:`
	Address of the first item of the SNetInterfaceInfo array, which contains information about PC network adapters. If the value is NULL, only the number of network adapters has been requested.
	`UINT* inout_InterfaceCount:`
	For the input: Number of SNetInterfaceInfo items assigned by the user.
	For the output: Number of network adapters (array items)	

Return values	Runtime error code	Description
	`SREC_OK`	The function call is successful.

Return values	`SREC_INDEX_OUT_OF_RANGE`	The number of array units is lower than the number of network adapters.
	`SREC_INTERNAL_ERROR`	The system call has caused an internal error.

Example C++	`#include <iostream>`
	`#include "SimulationRuntimeApi.h"`
	
	`ISimulationRuntimeManager* rtm = nullptr;`
	`ERuntimeErrorCode result = SREC_INVALID_ERROR_CODE;`
	
	`//init connection`
	`InitializeApi(&rtm);`
	
	`//get count of network interfaces`
	`UINT count;`
	`result = rtm->GetNetInterfaces(nullptr, &count);`
	
	`//get actual network interfaces`
	`std::unique_ptr<SNetInterfaceInfo[]> netInterfaces =`
	`std::make_unique<SNetInterfaceInfo[]>(count);`
	`result = rtm->GetNetInterfaces(netInterfaces.get(), &count);`
	
	`ShutdownAndFreeApi(rtm);`	

Table 6-23 NetInterfaces { get; } - .NET (C#)

Syntax	`SNetInterfaceInfo[] NetInterfaces { get; }`	
Return values	Runtime error code	Description
	`SNetInterfaceInfo[]`	Information about the PC network interfaces
Exceptions	`ERuntimeErrorCode.InternalError`	The system call has caused an internal error.

SetNetInterfaceBindings()

Sets the binding of an S7-PLCSIM Advanced Virtual Switch to PC interfaces based on the network mode and the mapping configuration of the network interfaces for the instance.

You must call the function from a parent session (administrator mode), otherwise you cannot access it, and the message `SREC_ACCESS_DENIED` appears. This function is not valid for the Softbus NetworkMode.

Table 6-24 SetNetInterfaceBindings() - Native C++

Syntax	`ERuntimeErrorCode` `SetNetInterfaceBindings(DWORD in_PCInterfaceIndex = 0);`	
Parameters	`DWORD in_PCInterfaceIndex = 0`: • When Single Adapter Network Mode (promiscuous mode) is set, you can specify the index of the PC interfaces for network communication. To enable only local PC communication, use the value (0). • When Multi-Adapter Network Mode (non-promiscuous mode) is set, this parameter is not used and the binding is set according to the configured mapping of the interfaces.	
Return values	Runtime error code	Description
	`SREC_OK`	The function call is successful.

Return values	SREC_INVALID_CONFIGURATION	The configuration is invalid because an invalid adapter is mapped on an instance. For example, a sim_hwdb.ini file of a registered instance uses the MAC address of an invalid adapter such as a PLCSIM vAdapter, a disabled adapter, a wireless adapter, or an adapter that does not support PLCSIM vSwitch.
	SREC_ACCESS_DENIED	You do not have authorization to connect to the PC network interfaces.
	SREC_WRONG_COMMUNICATION_INTERFACE	The PC interface does not exist or is disabled.
	SREC_INTERNAL_ERROR	The system call has caused an internal error. Enable the traces to obtain detailed information.
	SREC_INSTANCE_ALREADY_RUNNING	In Single Adapter Network Mode (promiscuous mode), bindings cannot change while an instance is running.
	SREC_VIRTUAL_SWITCH_MISCONFIGURED	The configuration of an S7-PLCSIM Advanced Virtual Switch is not correct or the driver is not functioning properly.
	SREC_NOT_SUPPORTED	You cannot call the function in the Softbus NetworkMode.

Table 6-25 SetNetInterfaceBindings() - .NET (C#)

Syntax	`void SetNetInterfaceBindings();` `void SetNetInterfaceBindings(DWORD in_PCInterfaceIndex);`	
Exceptions	Runtime error code	Description
	ERuntimeErrorCode.InvalidConfiguration	The configuration is invalid because, for example, a non-existent adapter was used.
	ERuntimeErrorCode.AccessDenied	You do not have authorization to connect to the PC network interfaces.
	ERuntimeErrorCode.WrongCommunicationInterface	The PC interface does not exist or is disabled.
	ERuntimeErrorCode.InternalError	The system call has caused an internal error. Enable the traces to obtain detailed information.
	ERuntimeErrorCode.InstanceAlreadyRunning	In Single Adapter Network Mode (promiscuous mode), bindings cannot change while an instance is running.
	ERuntimeErrorCode.VirtualSwitchMisconfigured	The configuration of an S7-PLCSIM Advanced Virtual Switch is not correct or the driver is not functioning properly.
	ERuntimeErrorCode.NotSupported	You cannot call the function in the Softbus NetworkMode.

ResetNetInterfaceBindings()

Deactivates the binding of an S7-PLCSIM Advanced Virtual Switch to PC interfaces. The binding on the Siemens PLCSIM Virtual Ethernet Adapter remains.

You must call the function from a parent session (administrator mode), otherwise you cannot access it, and the message SREC_ACCESS_DENIED appears. For local runtime manager, the

application using the API must be in an elevated session; for remote use, the remote runtime manager must be in the elevated session. This function is not valid for the Softbus NetworkMode.

Table 6-26 ResetNetInterfaceBindings() - Native C++

Syntax	`ERuntimeErrorCode ResetNetInterfaceBindings();`	
Parameters	None	
Return values	Runtime error code	Description
	`SREC_OK`	The function call is successful.
	`SREC_ACCESS_DENIED`	You do not have authorization to connect to the PC network interfaces.
	`SREC_INSTANCE_ALREADY_RUNNING`	In Single Adapter Network Mode (promiscuous mode), the bindings cannot change while an instance is running.
	`SREC_INTERNAL_ERROR`	The system call has caused an internal error. Enable the traces to obtain detailed information.
	`SREC_NOT_SUPPORTED`	You cannot call the function in the Softbus NetworkMode.

ResetNetInterfaceBindings() - .NET (C#)

Syntax	`void ResetNetInterfaceBindings();`	
Exceptions	Runtime error code	Description
	`ERuntimeErrorCode.AccessDenied`	You do not have authorization to connect to the PC network interfaces.
	`ERuntimeErrorCode.InternalError`	The system call has caused an internal error. Enable the traces to obtain detailed information.
	`ERuntimeErrorCode.InstanceAlreadyR-unning`	In Single Adapter Network Mode (promiscuous mode), the bindings cannot change while an instance is running.
	`ERuntimeErrorCode.NotSupported`	You cannot call the function in the Softbus NetworkMode.

GetDefaultStoragePath()

Returns the full path of the default folder in which the instance stores its data.

Table 6-27 GetDefaultStoragePath() - Native C++

Syntax	`ERuntimeErrorCode GetDefaultStoragePath(WCHAR* out_DefaultStoragePath);`	
Parameters	• `WCHAR* out_DefaultStoragePath` Output parameter that is filled with the path of the default folder.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.

SetDefaultStoragePath()

Sets the full path of the default folder in which the instance stores its data.

Table 6-28 SetDefaultStoragePath() - Native C++

Syntax	`ERuntimeErrorCode SetDefaultStoragePath(const WCHAR* in_DefaultStoragePath);`	
Parameters	• const WCHAR* in_DefaultStoragePath Input parameter that specifies the path of the default folder.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_WRONG_ARGUMENT`	The path contains invalid characters.
	`SREC_CONFIG_FILE_ERROR`	The configuration file UserInterfaceConfiguration.xml could not be read.

DefaultStoragePath { get; set; }

Returns or sets the full path of the default folder in which the instance stores its data.

Table 6-29 DefaultStoragePath { get; set; } - .NET (C#)

Syntax	string DefaultStoragePath { get; set; }	
Return values	Runtime error code	Description
	`OK`	The function is successful.
	`WrongArgument`	The path contains invalid characters.
	`ConfigFileError`	The configuration file UserInterfaceConfiguration.xml could not be read.

ISimulationRuntimeManager

7

7.1 Remote connections

OpenPort()

Opens a port to which another Runtime Manager can connect.

Table 7-1 OpenPort() - Native C++

Syntax	`ERuntimeErrorCode OpenPort(UINT16 in_Port);`	
Parameters	• `UINT16 in_Port:` The port. The value must be greater than 1024.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_ALREADY_EXISTS`	A port is already open.
	`SREC_WRONG_ARGUMENT`	The port is invalid.
	`SREC_CONNECTION_ERROR`	The port cannot be opened.

Table 7-2 OpenPort() - .NET (C#)

Syntax	`void OpenPort(UInt16 in_Port);`	
Parameters	• `UInt16 in_Port:` The port. The value must be greater than 1024.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.AlreadyExists`	A port is already open.
	`ERuntimeErrorCode.WrongArgument`	The port is invalid.
	`ERuntimeErrorCode.ConnectionError`	The port cannot be opened.

ClosePort()

Closes an open port and all open connections that another Runtime Manager has created to this open port.

Table 7-3 ClosePort() - Native C++

Syntax	`ERuntimeErrorCode ClosePort();`	
Parameters	None	
Return values	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_WARNING_INVALID_CALL`	No port is open.

Table 7-4 ClosePort() - .NET (C#)

Syntax	`void ClosePort();`	
Parameters	None	
Return values	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.Timeout`	The function does not return on time.

GetRemoteConnectionsCount()

Supplies the number of open remote connections.

Table 7-5 GetRemoteConnectionsCount() - Native C++

Syntax	`UINT32 GetRemoteConnectionsCount();`
Parameters	None
Return values	`UINT32`: Number of open remote connections.

GetRemoteConnectionInfoAt()

Returns information about an open connection.

Table 7-6 GetRemoteConnectionInfoAt()- Native C++

Syntax	`ERuntimeErrorCode GetRemoteConnectionInfoAt(UINT32 in_Index, SConnectionInfo* out_ConnectionInfo);`	
Parameters	• `UINT32 in_Index:` Index of the connection information that is expected. • `SConnectionInfo* out_ConnectionInfo:` The connection information for this index.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INDEX_OUT_OF_RANGE`	Connection information for this index does not exist.

RemoteConnectionInfo { get; }

Returns an array of information about all open connections.

Table 7-7 RemoteConnectionInfo { get; } - .NET (C#)

Syntax	`SConnectionInfo[] RemoteConnectionInfo { get; }`	
Parameters	None	
Return values	`SConnectionInfo[]`: An array of information about all open connections.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.Timeout`	The function does not return on time.

RemoteConnect()

Creates a new connection to a remote Runtime Manager or uses an existing connection to create an IRemoteRuntimeManager interface.

Table 7-8 RemoteConnect() - Native C++

Syntax	`ERuntimeErrorCode RemoteConnect(UINT8 in_IP3, UINT8 in_IP2, UINT8 in_IP1, UINT8 in_IP0, UINT16 in_Port, IRemoteRuntimeManager**out_RemoteRuntimeManagerInterface);` `ERuntimeErrorCode RemoteConnect(UIP in_IP, UINT16 in_Port, IRemoteRuntimeManager** out_RunTimeManagerInterface);`
Parameters	• `UINT8 in_IP3:` First part of the IP address of the remote PC. • `UINT8 in_IP2:` Second part of the IP address of the remote PC. • `UINT8 in_IP1:` Third part of the IP address of the remote PC. `UINT8 in_IP0:` Last part of the IP address of the remote PC. • `UIP in_IP:` IP address of the remote PC.

	• `UINT16 in_Port:` The port that is open on the remote PC. • `IRemoteRuntimeManager** out_RemoteRuntimeManagerInterface:` Pointer to a remote Runtime Manager interface pointer. The pointer **must** be initialized with `NULL`. The interface is created in the function.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_CONNECTION_ERROR`	The connection to the remote Runtime Manager cannot be established.
	`SREC_WRONG_ARGUMENT`	IP, port or IInstance pointer is invalid.
	`SREC_WRONG_VERSION`	The version of the API is not compatible with Runtime. See information on compatibility in the S7-PLCSIM Advanced Function Manual.

Example C++	`ISimulationRuntimeManager* api = NULL;`
	`ERuntimeErrorCode result = InitializeApi(&api);`
	
	`IRemoteRuntimeManager * client = NULL;`
	
	`if (result == SREC_OK)`
	`{`
	`result = api->`**`RemoteConnect`**`(192,203,145,144, 4444, &client);`
	`}`	

NOTE

Native C++

If you no longer require the interface, delete the interface.

See DestroyInterface() (Page 38).

Table 7-9 RemoteConnect() - .NET (C#)

Syntax	`IRemoteRuntimeManager RemoteConnect(string in_ConnectionString);`
	`IRemoteRuntimeManager RemoteConnect(SIP in_IP, UInt16 in_Port);`
	`IRemoteRuntimeManager RemoteConnect(Byte in_IP3, Byte in_IP2, Byte in_IP1, Byte in_IP0, UInt16 in_Port);`	

Parameters	• `Byte in_IP3:` First part of the IP address of the remote PC. • `Byte in_IP2:` Second part of the IP address of the remote PC. • `Byte in_IP1:` Third part of the IP address of the remote PC. • `Byte in_IP0:` Last part of the IP address of the remote PC. • `string in_ConnectionString:` A string in the form of "<IP3>.<IP2>.<IP1>.<IP0>:<Port>" Example: "182.203.145.144:4444". • `SIP in_IP:` IP address of the remote PC. • `UInt16 in_Port:` The port that is open on the remote PC.	
Return values	`IRemoteRuntimeManager:` Interface to the remote Runtime Manager.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description

Exceptions	`ERuntimeErrorCode.ConnectionError`	Connection to the remote Runtime Manager cannot be established.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.WrongArgument`	IP or port is invalid.
	`ERuntimeErrorCode.WrongVersion`	The version of the API is not compatible with Runtime. See information on compatibility in the S7-PLCSIM Advanced Function Manual.

7.1.1 RunAutodiscover()

Description

This function identifies all Runtime Managers that are on the network and that are ready to establish a remote connection.

NOTE

The function identifies Runtime Managers as of S7-PLCSIM Advanced V5.0.

Requirements

- The Runtime Manager must be running and allowing remote connections.
- The firewall of the remote PC must not block traffic on the selected UDP port.
- Devices in the local network (such as routers, switches, firewalls) must not block multicast packets of the selected class.

RunAutodiscover()

The function starts the identification of the Runtime Manager in the network.

Table 7-10 RunAutodiscover() - Native C++

Syntax	`ERuntimeErrorCode RunAutodiscover(UINT32 in_Timeout = 2000);`	
Parameters	• `UINT32 in_Timeout` A timeout value in milliseconds that defines how long the local Runtime Manager waits for responses from the Remote Manager. A value between 500 ms and 30000 ms is valid. Default: 2000 ms.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.

Return values	SREC_WRONG_ARGUMENT	The timeout value is outside the permissible range.
	SREC_AUTODISCOVER_ALREADY_RUNNING	A RunAutodiscover() call is already running in the background. Wait for the message SRRSI_DISCOVER_FINISHED in the callback function. See EAutodiscoverType (Page 275).
	SREC_TIMEOUT	Communication errors in the local Runtime Manager.

Table 7-11 RunAutodiscover() - .NET (C#)

Syntax	void RunAutodiscover(UInt32 in_Timeout = 2000);	
Parameters	• UInt32 in_Timeout A timeout value in milliseconds that defines how long the local Runtime Manager waits for responses from the Remote Manager. A value between 500 ms and 30000 ms is valid. Default: 2000 ms.	
Exceptions	Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException	
	Runtime error code	Description
	ERuntimeErrorCode.WrongArgument	The timeout value is outside the permissible range.
	ERuntimeErrorCode.Autodiscover-AlreadyRunning	A RunAutodiscover() call is already running in the background. Wait for the message AutodiscoverFinished in the callback function. See EAutodiscoverType (Page 275).
	ERuntimeErrorCode.Timeout	Communication errors in the local Runtime Manager.

7.2 Events for ISimulationRuntimeManager

Events for runtime instances and remote connections

The following events are triggered for the ISimulationRuntimeManager interface:

Table 7-12 Events for ISimulationRuntimeManager

Event	Cause
AUTOHOTSPOT	The Runtime Manager configuration has changed: • A new instance is registered. • An instance is removed. • A connection to a client is established. The Control Panel uses such an event to update the list of available instances.
OnRuntimeManagerLost (Page 68)	The connection to the Runtime Manager is interrupted.
OnAutodiscoverData events (Page 70)	The network searches for Runtime Managers which are ready to establish a remote connection.

7.2.1 OnRuntimeManagerLost events

OnRuntimeManagerLost

Registers or unregisters an event handler method.

Table 7-13 OnRuntimeManagerLost - .NET (C#)

Syntax	`event Delegate_Void OnRuntimeManagerLost;`
Parameters	None. See Delegate_Void (Page 229).
Return values	None
Exceptions	None
Note	The event handler method runs in a separate thread.

RegisterOnRuntimeManagerLostCallback()

When the event occurs, the registered callback function is called. Multiple callback functions can be registered for the event.

Table 7-14 RegisterOnRuntimeManagerLostCallback() - Native C++

Syntax	`void RegisterOnRuntimeManagerLostCallback(EventCallback_VOID in_CallbackFunction);`
Parameters	• `EventCallback_VOID in_CallbackFunction:` A callback function that subscribes to the event. See EventCallback_VOID (Page 221).
Return values	None
Note	The event handler method runs in a separate thread.

RegisterOnRuntimeManagerLostEvent()

When the event occurs, the registered event object is set to the signaled state. Only one event object can be registered for the event. Registration of a new event object causes the previous event object to be deleted.

Table 7-15 RegisterOnRuntimeManagerLostEvent() - Native C++

Syntax	`void RegisterOnRuntimeManagerLostEvent();` `void RegisterOnRuntimeManagerLostEvent(HANDLE* in_Event);`
Parameters	• `None:` An internal event handle is registered. • `HANDLE* in_Event:` A user-specific event handle is registered.
Return values	None

Table 7-16 RegisterOnRuntimeManagerLostEvent() - .NET (C#)

Syntax	`void RegisterOnRuntimeManagerLostEvent();`
Parameters	None
Return values	None

UnregisterOnRuntimeManagerLostCallback()

Unregisters the callback function. When the event occurs, no callback function is called.

Table 7-17 UnregisterOnRuntimeManagerLostCallback() - Native C++

Syntax	`void UnregisterOnRuntimeManagerLostCallback();`
Parameters	Optional parameter of the same type as for the Register*() function If you use the parameter, it uninstalls the specific function; otherwise, the last registered callback function is unregistered.
Return values	None

UnregisterOnRuntimeManagerLostEvent()

Unregisters the event object.

Table 7-18 UnregisterOnRuntimeManagerLostEvent() - Native C++

Syntax	`void UnregisterOnRuntimeManagerLostEvent();`
Parameters	None
Return values	None

Table 7-19 UnregisterOnRuntimeManagerLostEvent() - .NET (C#)

Syntax	`void UnregisterOnRuntimeManagerLostEvent();`
Parameters	None
Return values	None

WaitForOnRuntimeManagerLostEvent()

The function will block the program until the registered event object is set to the signaled state or the timeout interval is exceeded.

Table 7-20 WaitForOnRuntimeManagerLostEvent() - Native C++

Syntax	`bool WaitForOnRuntimeManagerLostEvent();` `bool WaitForOnRuntimeManagerLostEvent(UINT32 in_Time_ms);`
Parameters	• `None:` The time limit is set to `INFINITE`. • `UINT32 in_Time_ms:` Value for the time limit in milliseconds.
Return values	• `true:` If the event object was set to the signaled state. • `false:` If no event was received during the defined timeout interval.

Table 7-21 WaitForOnRuntimeManagerLostEvent() - .NET (C#)

Syntax	`bool WaitForOnRuntimeManagerLostEvent();` `bool WaitForOnRuntimeManagerLostEvent(UInt32 in_Time_ms);`
Parameters	• `None:` The time limit is set to `INFINITE`. • `UInt32 in_Time_ms:` Value for the time limit in milliseconds.
Return values	• `true:` If the event object was set to the signaled state. • `false:` If no event was received during the defined timeout interval.

7.2.2 OnAutodiscoverData events

OnAutodiscoverData

Registers or unregisters an event handler method.

Table 7-22 OnAutodiscoverData - .NET (C#)

Syntax	`event Delegate_SRRSI_AD OnAutodiscoverData`
Parameters	None. See Delegate_SRRSI_AD (Page 230).
Return values	None
Exceptions	None
Note	The event handler method runs in a separate thread.

RegisterOnAutodiscoverCallback()

When the event occurs, the registered callback function is called. Multiple callback functions can be registered for the event.

Table 7-23 RegisterOnAutodiscoverCallback() - Native C++

Syntax	`void RegisterOnAutodiscoverCallback(EventCallback_SRRSI_AD in_CallbackFunction);`
Parameters	• `EventCallback_SRRSI_AD in_CallbackFunction:` Pointer to a user-defined callback function. See EventCallback_SRRSI_AD (Page 222).
Return values	None

UnregisterOnAutodiscoverCallback()

Unregisters the callback function.

Table 7-24 UnregisterOnAutodiscoverCallback() - Native C++

Syntax	`void UnregisterOnAutodiscoverCallback();`
Parameters	Optional parameter of the same type as for the Register*() function If you use the parameter, it uninstalls the specific function; otherwise, the last registered callback function is unregistered.
Return values	None

IsInitialized() / IsInitialized { get; }

Returns a value that indicates whether the API was successfully initialized.

Table 7-25 IsInitialized() - Native C++

Syntax	`bool IsInitialized();`
Parameters	None
Return values	• `false`: If the API was not initialized. • `true`: If the API was initialized.

Table 7-26 IsInitialized { get; } - .NET (C#)

Syntax	`bool IsInitialized { get; }`
Parameters	None
Return values	• `false`: If the API was not initialized. • `true`: If the API was initialized.

IsRuntimeManagerAvailable() / IsRuntimeManagerAvailable { get; }

The function returns `false` when the connection to Runtime Manager is interrupted. This happens only when the Runtime Manager process is closed.

Subscribe to the `OnRuntimeManagerLost()` event to find out whether the connection is interrupted. See Events (Page 67).

Table 7-27 IsRuntimeManagerAvailable() - Native C++

Syntax	`bool IsRuntimeManagerAvailable();`
Parameters	None
Return values	• `false`: If the connection is interrupted. • `true`: If the connection is active.

Table 7-28 IsRuntimeManagerAvailable { get; } - .NET (C#)

Syntax	`bool IsRuntimeManagerAvailable{ get; }`
Parameters	None
Return values	• `false`: If the connection is interrupted. • `true`: If the connection is active.

Shutdown()

Ends communication with Runtime Manager and clears the interfaces.

Call this function in the following cases:

- Immediately before the API library (DLL) is unregistered (native C++).
- When your application is no longer using Runtime Manager.

Table 7-29 Shutdown() - Native C++

Syntax	`ERuntimeErrorCode Shutdown()`	
Parameters	None	
Return values	Runtime error code	Description
	`SREC_OK`	The function call is successful.

Table 7-30 Shutdown() - .NET (C#)

Syntax	`void Shutdown()`
Parameters	None
Return values	None

IInstance

8

8.1 Interfaces for IInstances

Dispose()

Deletes the managed interface and unloads the native components of the user interfaces.

Table 8-1 Dispose() - .NET (C#)

Syntax	`void Dispose()`
Parameters	None
Return values	None

GetID() / ID { get; }

Returns the instance ID. The ID is assigned by Runtime Manager when the instance is registered.

Table 8-2 GetID() - Native C++

Syntax	`INT32 GetID();`
Parameters	None
Return values	`INT32`: Instance ID

Table 8-3 ID { get; } - .NET (C#)

Syntax	`UInt32 ID { get; }`
Parameters	None
Return values	`Uint32`: Instance ID
Exceptions	None

GetName() / Name { get; }

Returns the name of the instance.

Table 8-4 GetName() - Native C++

Syntax	`ERuntimeErrorCode GetName(WCHAR inout_Name[], UINT32 in_ArrayLength);`	
Parameters	• `WCHAR inout_Name[]`: A user-allocated storage for the name of the instance. The field length should be at least as long as `DINSTANCE_NAME_MAX_LENGTH`. See Definitions and constants (Page 237). • `UINT32 in_ArrayLength`: Field length (Wide character)	
Return values	Runtime error code	Description

Return values	SREC_OK	The function is successful.
	SREC_INTERFACE_REMOVED	The instance is not registered in Runtime Manager.
	SREC_WRONG_ARGUMENT	The name does not fit in the storage.

Example C++	`ISimulationRuntimeManager * api = NULL;`
	`ERuntimeErrorCode result = InitializeApi(&api);`
	`IInstance* psa = NULL;`
	`if (result == SREC_OK)`
	`{`
	`  result = api->RegisterInstance(&psa);`
	`}`
	
	`WCHAR name[DINSTANCE_NAME_MAX_LENGTH];`
	`if (result == SREC_OK)`
	`{`
	`  result = psa->`**`GetName`**`(name, DINSTANCE_NAME_MAX_LENGTH);`
	`}`	

Table 8-5 Name { get; } - .NET (C#)

Syntax	`string Name { get; }`	
Parameters	None	
Return values	Name of the instance.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.

GetCPUType()

Returns the CPU type of the simulated PLC instance.

Table 8-6 GetCPUType() - Native C++

Syntax	`ECPUType GetCPUType();`
Parameters	None
Return values	An enumeration element that defines the CPU type. See ECPUType (Page 261).

SetCPUType()

Sets the CPU type of the simulated PLC instance. A change of CPU type occurs only when the controller is restarted.

NOTE

When you create a new instance and set the CPU type, S7-PLCSIM Advanced uses precise simulation to simulate the PLC after `PowerOn`. The API loads the quantity structure based on the selected type. You can only overload the CPU type using a hardware-compatible PLC.

Table 8-7 SetCPUType() - Native C++

Syntax	`void SetCPUType(ECPUType in_Value);`
Parameters	• `ECPUType in_Value:` Defines which CPU type is simulated at the start of the instance. When a different CPU type is loaded via STEP 7 or from the Virtual Memory Card, this CPU type applies.
Return values	None

CPUType { get; set; }

Returns or sets the CPU type of the simulated PLC instance. A change of CPU type occurs only when the controller is restarted.

When a different CPU type is loaded via STEP 7 or from the Virtual Memory Card, this CPU type applies.

NOTE

When you create a new instance and set the CPU type, S7-PLCSIM Advanced uses precise simulation to simulate the PLC after `PowerOn`. The API loads the quantity structure based on the selected type. You can only overload the CPU type using a hardware-compatible PLC.

Table 8-8 CPUType { get; set; } - .NET (C#)

Syntax	`ECPUType CPUType { get; set; }`
Parameters	None
Return values	An enumeration element that defines the CPU type.
Exceptions	None

GetArticleNumber() / ArticleNumber { get; }

Returns the set ArticleNumber of the simulated instance. If you only set the CPU type, the method returns the article number of the specified CPU type.

To use this method, you must first create a 20-character array for the article number using the constant `HWDB_B1_ORDER_ID_LEN` in SimulationRuntimeAPI.h. Pass a pointer to the array into the method. The method fills the array with a string containing the article number plus a zero in position 19.

Table 8-9 GetArticleNumber() - Native C++

Syntax	`GetArticleNumber(WCHAR *out_ArticleNumber);`
Parameters	`WCHAR *out_ArticleNumber`
Return values	None

Table 8-10 ArticleNumber{ get; } - .NET (C#)

Syntax	`System::String^ ArticleNumber { System::String^ get(); }`
Parameters	None
Return values	`System::String^` containing the article number

GetCommunicationInterface()

This property is now read-only. To set the network mode, refer to the NetworkMode section.

Returns the communication interface of the simulated controller: Local communication (Softbus) or TCPIP.

Table 8-11 GetCommunicationInterface() - Native C++

Syntax	`ECommunicationInterface GetCommunicationInterface();`
Parameters	None
Return values	• `SRCI_NONE` Cannot be selected. Is returned if the instance interface is no longer valid. • `SRCI_SOFTBUS` Is returned if the simulated controller uses the Softbus. • `SRCI_TCPIP` Is returned if the simulated controller communicates over the virtual adapter.

CommunicationInterface { get; }

This property is now read-only. To set the network mode, refer to the NetworkMode section (Page 70).

Returns the communication interface of the simulated controller: Local communication (Softbus) or TCPIP.

Table 8-12 CommunicationInterface { get; } - .NET (C#)

Syntax	`ECommunicationInterface CommunicationInterface { get; }`
Parameters	None
Return values	• `ECommunicationInterface.None` Cannot be selected. Is returned if the instance interface is no longer valid. • `ECommunicationInterface.Softbus` Is returned if the simulated controller uses the Softbus. • `ECommunicationInterface.TCPIP` Is returned if the simulated controller communicates over the virtual adapter.
Exceptions	None

GetInfo() / Info { get; }

Returns a structure that provides information about the instance.

Table 8-13 GetInfo() - Native C++

Syntax	`SInstanceInfo GetInfo();`
Parameters	None
Return values	`SInstanceInfo:` A structure that provides information about the instance. See SInstanceInfo (Page 246).

Table 8-14 Info { get; } - .NET (C#)

Syntax	`SInstanceInfo Info { get; }`
Parameters	None
Return values	`SInstanceInfo`: A structure that provides information about the instance.
Exceptions	None

GetNetInterfaceMapping()

Returns an interface index of the PC interface to which a specific CPU interface is mapped. Use the `GetNetInterfaces()` function to get a list of PC interfaces with interface index. The value 0 means that no mapping has been set.

Note that GetNetInterfaceMapping() only returns a mapping table and not the actual address while the instance is running.

NOTE

If two network adapters have the same MAC address, the `GetNetInterfaceMapping()` function might not return the adapter that you previously set using the `SetNetInterfaceMapping()` function. This issue can occur specifically with adapters that use the 'MAC Address Pass-through' feature, leading to potential conflicts in adapter identification and mapping.

Table 8-15 GetNetInterfaceMapping() - Native C++ und .NET (C#)

Syntax C++	`INT32 GetNetInterfaceMapping(EPLCInterface in_PLCInterfaceID);`	
Syntax .NET (C#)	`int GetNetInterfaceMapping(EPLCInterface in_PLCInterfaceID);`	
Parameters	`EPLCInterface in_PLCInterfaceID`: ID of the PLC interface for which you want to get the mapping.	
Return values	Value	Description
	Greater than 0	Index of mapped computer interface
	0	No mapping is defined for the PLC interface or the NetworkMode is Softbus
	-1	Cannot get the mapping

SetNetInterfaceMapping()

Sets interface mappings between CPU interface and PC interface. Ethernet packets that a certain PC interface receives are forwarded to the assigned CPU interface. Use the `GetNetInterfaces()` function to get a list of PC interfaces.

Mapping of PLCSIM Virtual Ethernet adapters or wireless adapters is not possible. The function is not valid for the Softbus NetworkMode.

Table 8-16 SetNetInterfaceMapping() - Native C++

Syntax	`ERuntimeErrorCode SetNetInterfaceMapping(EPLCInterface in_PLCInterfaceID, DWORD in_PCInterfaceIndex);`
	`ERuntimeErrorCode SetNetInterfaceMapping(EPLCInterface in_PLCInterfaceID, const WCHAR* in_PCInterfaceName);`
	`ERuntimeErrorCode SetNetInterfaceMapping(EPLCInterface in_PLCInterfaceID, const SMACAddress* in_MACAddressParam);`	

Parameters	`EPLCInterface in_PLCInterfaceID:`
	ID of the CPU interface for which a MAC address is to be set.
	`DWORD* in_PCInterfaceIndex:`
	Index of the PC interface; if this value is 0, the mapping is deleted.
	`const WCHAR* in_PCInterfaceName:`
	Name of the PC interface
	`const UMACAddress* in_MACAddressParam:`
	Hardware MAC address of the PC interface in the form FF-FF-FF-FF-FF-FF, where the octet separator is a "-".
	You get these values with the function `GetNetInterfaces()`.	

Return values	Runtime error code	Description
	`SREC_OK`	The function call is successful.
	`SREC_WRONG_COMMUNICATION_INTERFACE`	This error is returned in TCPIP modes in case the specified/mapped computer interface in the sim_hwdb.ini file is disabled or it is the PLCSIM Virtual Adapter.
	`SREC_ALREADY_EXISTS`	The PC interface has already been assigned to another CPU interface.
	`SREC_CONFIG_FILE_ERROR`	An error occurred when writing the configuration to the file "sim_hwdb.ini".
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_INSTANCE_ALREADY_RUNNING`	Bindings cannot change while the instance is running.
	`SREC_NOT_SUPPORTED`	You cannot call the function in Softbus NetworkMode.

Table 8-17 SetNetInterfaceMapping() - .NET (C#)

Syntax	`SetNetInterfaceMapping(EPLCInterface in_PLCInterfaceID, uint in_PCInterfaceIndex);`
	`SetNetInterfaceMapping(EPLCInterface in_PLCInterfaceID, const String in_PCInterfaceName);`
	`SetNetInterfaceMapping(EPLCInterface in_PLCInterfaceID, const SMACAddress in_MACAddressParam);`	

Exceptions	Runtime error code	Description
	`ERuntimeErrorCode.WrongCommunicationInterface`	This error is returned in TCPIP modes in case the specified/mapped computer interface in the sim_hwdb.ini file is disabled or it is the PLCSIM Virtual Adapter.
	`ERuntimeErrorCode.AlreadyExists`	The PC interface has already been assigned to another CPU interface.

Exceptions	`ERuntimeErrorCode.ConfigFileError`	An error occurred when writing the configuration to the file "sim_hwdb.ini".
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.InstanceAlreadyRunning`	Bindings cannot change while the instance is running.
	`ERuntimeErrorCode.NotSupported`	You cannot call the function in Softbus NetworkMode.

CheckNetInterfaceMapping()

The function checks the validity of the interface mapping in order to start the instance. All PC instances already running are checked for possible duplicate mappings. The function also checks whether the binding to the PC network interfaces is valid. The function is not valid for the Softbus NetworkMode.

Table 8-18 CheckNetInterfaceMapping() - Native C++

Syntax	`ERuntimeErrorCode CheckNetInterfaceMapping();`	
Parameters	None	
Return values	Runtime error code	Description
	`SREC_OK`	The function call is successful.
	`SREC_INVALID_CONFIGURATION`	The configuration is invalid. Examples: • The current instance has a mapped PC network interface that is already in use by another running instance. • A mapped PLCSIM Virtual Ethernet Adapter exists.
	`SREC_VIRTUAL_SWITCH_MISCONFIGURED`	The binding of an S7-PLCSIM Advanced Virtual Switch on a PC network interface is not correct, or the driver of an S7-PLCSIM Advanced Virtual Switch is not functioning properly. If the binding is not correct, call the `SetNetInterfaceBindings()` function or set the binding manually before starting the instance.
	`SREC_WARNING_NO_EXTERNAL_COMMUNICATION`	You have set the Multi-Adapter Network Mode (non-promiscuous mode) without interface mapping or Softbus as communication interface.
	`SREC_INTERNAL_ERROR`	The system call has caused an internal error.
	`SREC_WRONG_COMMUNICATION_INTERFACE`	This error is returned in TCPIP modes in case the specified/mapped computer interface in the sim_hwdb.ini file is disabled or it is the PLCSIM Virtual Adapter.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_PCAP_DRIVER_NOT_RUNNING`	The PCAP driver is not running.
	`SREC_NOT_SUPPORTED`	You cannot call the function in the Softbus NetworkMode.

Table 8-19 CheckNetInterfaceMapping() - .NET (C#)

Syntax	`ERuntimeErrorCode CheckNetInterfaceMapping();`	
Return values	Runtime error code	Description
	`ERuntimeErrorCode.OK`	The function call is successful.
	`ERuntimeErrorCode.InvalidConfiguration`	The configuration is invalid. Examples: • The current instance has a mapped PC network interface that is already in use by another running instance. • A mapped PLCSIM Virtual Ethernet Adapter exists.
	`ERuntimeErrorCode.VirtualSwitchMisconfigured`	The binding of an S7-PLCSIM Advanced Virtual Switch on a PC network interface is not correct, or the driver of an S7-PLCSIM Advanced Virtual Switch is not functioning properly. If the binding is not correct, call the `SetNetInterfaceBindings()` function or set the binding manually before starting the instance.
	`ERuntimeErrorCode.WarningNoExternalCommunication`	You have set the Multi-Adapter Network Mode (non-promiscuous mode) without interface mapping or Softbus as communication interface.
	`ERuntimeErrorCode.InternalError`	The system call has caused an internal error.
	`ERuntimeErrorCode.WrongCommunicationInterface`	This error is returned in TCPIP modes in case the specified/mapped computer interface in the sim_hwdb.ini file is disabled or it is the PLCSIM Virtual Adapter.
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.PCAPDriverNotRunning`	The PCAP driver is not running.
	`ERuntimeErrorCode.NotSupported`	You cannot call the function in the Softbus NetworkMode.

UnregisterInstance()

Unregisters this instance from Runtime Manager.

NOTE

Loss of the interfaces

Other applications that are connected to this instance will lose their interface to this instance.

NOTE

Do not unregister an instance or free the API while either the control panel or the API is starting an instance. Wait until the PowerOn action for this instance is complete.

Table 8-20 UnregisterInstance() - Native C++

Syntax	`ERuntimeErrorCode UnregisterInstance();`	
Parameters	None	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.

Table 8-21 UnregisterInstance() - .NET (C#)

Syntax	`void UnregisterInstance();`	
Parameters	None	
Return values	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.

8.2 Controller - Information and settings

GetControllerName() / ControllerName { get; }

Returns the downloaded name of the simulated PLC instance.

Table 8-22 GetControllerName() - Native C++

Syntax	`ERuntimeErrorCode GetControllerName(WCHAR inout_Name[], UINT32 in_ArrayLength);`	
Parameters	• `WCHAR inout_Name[]:` A user-allocated storage for the name. • `UINT32 in_ArrayLength:` The length of the storage.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INDEX_OUT_OF_RANGE`	The name does not fit in the storage.

Table 8-23 ControllerName { get; } - .NET (C#)

Syntax	`string ControllerName { get; }`	
Parameters	None	
Return values	`string:` The downloaded name of the simulated PLC instance.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.

GetControllerShortDesignation() / ControllerShortDesignation { get; }

Returns the downloaded short designation of the simulated PLC instance.

Table 8-24 GetControllerShortDesignation() - Native C++

Syntax	`ERuntimeErrorCode GetControllerShortDesignation(WCHAR inout_ShortDesignation[], UINT32 in_ArrayLength);`	
Parameters	• `WCHAR inout_ShortDesignation[]:` A user-allocated storage for the short designation. • `UINT32 in_ArrayLength:` The length of the storage.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INDEX_OUT_OF_RANGE`	The name does not fit in the storage.

Table 8-25 ControllerShortDesignation { get; } - .NET (C#)

Syntax	`string ControllerShortDesignation { get; }`	
Parameters	None	
Return values	`string:` The downloaded short designation of the simulated PLC instance.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.

GetControllerIPCount()

Returns the number of configured IP addresses of the simulated PLC instance. If the function fails, the return value is 0.

Table 8-26 GetControllerIPCount() - Native C++

Syntax	`UINT32 GetControllerIPCount();`
Parameters	None
Return values	`INT32`: Number of configured IP addresses of the simulated PLC instance.

GetControllerIP() / ControllerIP { get; }

Returns a configured IP address of the instance.

Table 8-27 GetControllerIP() - Native C++

Syntax	`UIP GetControllerIP();` `UIP GetControllerIP(UINT32 in_Index);`
Parameters	• `WCHAR in_Index`: The index of the IP address you want to receive. The index must be less than the value you receive from `GetControllerIPCount()`. The default setting is 0.
Return values	`UIP`: IP address of the simulated PLC instance. If the function fails, the return value is 0.

Table 8-28 ControllerIP { get; } - .NET (C#)

Syntax	`string[] ControllerIP { get; }`
Parameters	None
Return values	`string`: All downloaded IP addresses of the simulated PLC instance. If the function fails, the field is empty.
Exceptions	None

GetControllerIPSuite4() / ControllerIPSuite4 { get; }

Returns the IP suite instance. If the "Softbus" communication interface is used, the subnet mask and default gateway are 0.

Table 8-29 GetControllerIPSuite4() Native C++

Syntax	`SIPSuite4 GetControllerIPSuite4();` `SIPSuite4 GetControllerIPSuite4(UINT32 in_Index);`
Parameters	• `WCHAR in_Index`: The index of the IP address you want to receive. The index must be less than the value you receive from `GetControllerIPCount()`. The default setting is 0.
Return values	`SIPSuite4`: The IP suite of the simulated PLC instance. If the function fails, the return values are 0.

Table 8-30 ControllerIPSuite4 { get; } - .NET (#)

Syntax	`SIPSuite4[] ControllerIPSuite4 { get; };`
Parameters	None
Return values	`SIPSuite4[]`: All downloaded IP suites of the simulated PLC instance. If the function fails, the field is empty.
Exceptions	None

SetIPSuite()

Sets the IP suite of the network interface of a simulated PLC instance.

To change the subnet mask or gateway, you must also change the IP address. If you do not change the IP address, the API does not apply your other changes.

Table 8-31 SetIPSuite() - Native C++

Syntax	`ERuntimeErrorCode SetIPSuite(UINT32 in_InterfaceID, SIPSuite4 in_IPSuite, bool in_IsRemanent);`	
Parameters	• `UINT32 in_InterfaceID:` The ID of the network interface. • `SIPSuite4 in_IPSuite:` The IP suite that is to be assigned to the network interface. The IP suite contains the IP address, the subnet mask and the standard gateway. If the communication interface is "Softbus", the subnet mask and standard gateway are ignored. • `bool in_IsRemanent:` If `true`, the IP suite is saved after restart of the simulated PLC instance. If the communication interface is "Softbus", this flag is ignored.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.
	`SREC_DOES_NOT_EXIST`	There is no network interface with this ID.
	`SREC_INVALID_OPERATING_STATE`	The simulated PLC instance has not yet completed the boot process or is already in the shutdown phase.

Table 8-32 SetIPSuite() - .NET (C#)

Syntax	`void SetIPSuite(UInt32 in_InterfaceID, SIPSuite4 in_IPSuite, bool in_IsRemanent);`	
Parameters	• `UInt32 in_InterfaceID:` The ID of the network interface. • `SIPSuite4 in_IPSuite:` If the communication interface is "Softbus", the subnet mask and standard gateway are ignored. • `bool in_IsRemanent:` If `true`, the IP suite is saved after restart of the simulated PLC instance. If the communication interface is "Softbus", this flag is ignored.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.
	`ERuntimeErrorCode.DoesNotExist`	There is no network interface with this ID.
	`ERuntimeErrorCode.InvalidOperating-State`	The simulated PLC instance has not yet completed the boot process or is already in the shutdown phase.

GetStoragePath()

Returns the full path of the folder in which the instance stores its data.

Table 8-33 GetStoragePath() - Native C++

Syntax	`ERuntimeErrorCode GetStoragePath(WCHAR inout_StoragePath[], UINT32 in_ArrayLength);`	
Parameters	• `WCHAR inout_StoragePath[]:` A user-allocated storage for the storage path. The length of the array should be at least as long as `DSTORAGE_PATH_MAX_LENGTH`. See Data types (Page 220). • `UINT32 in_ArrayLength:` Length of the array (Wide character)	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INDEX_OUT_OF_RANGE`	The path does not fit in the storage.

SetStoragePath()

Sets the full path of the folder in which the instance stores its data. This can also be a network share.

Set the path before you start the instance. A change to the path takes effect only when the controller is restarted.

If no path is set, the default setting:
<My Documents>\Siemens\Simatic\Simulation\Runtime\Persistence\<Instance Name> is used.

Table 8-34 SetStoragePath() - Native C++

Syntax	`ERuntimeErrorCode SetStoragePath(WCHAR* in_StoragePath);`	
Parameters	• `WCHAR* in_StoragePath:` Full name of the storage path. The length of the name must be less than `DSTORAGE_PATH_MAX_LENGTH` and must not include Unicode characters. See Data types (Page 220).	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INDEX_OUT_OF_RANGE`	The length of the path exceeds the limit.
	`SREC_WRONG_ARGUMENT`	The path contains invalid characters.

StoragePath { get; set; }

Returns or sets the full path of the folder in which the instance stores its retentive data. This can also be a network share.

Set the path before you start the instance. A change to the path takes effect only when the controller is restarted.

If no path is set, the default setting:
<My Documents>\Siemens\Simatic\Simulation\Runtime\Persistence\<Instance Name> is used.

Table 8-35 StoragePath { get; set; } - .NET (C#)

Syntax	`string StoragePath { get; set; }`	
Parameters	None	
Return values	`string`: The configured storage path.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.IndexOutOfRange`	The length of the path exceeds the limit.
	`ERuntimeErrorCode.WrongArgument`	The path contains invalid characters.

ArchiveStorage()

The user program, the hardware configuration and the retentive data is stored in a file. The Virtual SIMATIC Memory Card. `ArchiveStorage()` stores this file as a ZIP file. The simulated PLC instance must be in OFF operating state for this. You must call the `PowerOn()` function at least once to avoid errors.

Table 8-36 ArchiveStorage() - Native C++

Syntax	`ERuntimeErrorCode ArchiveStorage(WCHAR* in_FullFileName);`	
Parameters	• `WCHAR in_FullFileName:` The full path to the ZIP file. The path relates to directories of the computer where the API is being called.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INVALID_OPERATING_STATE`	The instance is not in OFF operating state.
	`SREC_INVALID_ARCHIVE_PATH`	The archive path is invalid.
	`SREC_CREATE_DIRECTORIES_FAILED`	The folder for the ZIP file could not be created.
	`SREC_ARCHIVE_STORAGE_FAILED`	The ZIP file could not be created.
	`SREC_STORAGE_TRANSFER_ERROR`	Error during network data transfer. Memory data between client and server computers do not match.
	`SREC_NO_STORAGE_PATH_SET`	No storage path is set. You must execute the SetStoragePath() function beforehand.
	`SREC_ARCHIVE_STORAGE_NOT_CREATED`	The storage file could not be created. You must call the `PowerOn()` function at least once.

Table 8-37 ArchiveStorage() - .NET (C#)

Syntax	`void ArchiveStorage(string in_FullFileName);`	
Parameters	• `string in_FullFileName:` The full path to the ZIP file. The path relates to folders of the computer where the API is being called.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InvalidOperating-State`	The instance is not in OFF operating state.
	`ERuntimeErrorCode.InvalidArchivePath`	The archive path is invalid.
	`ERuntimeErrorCode.CreateDirectories-Failed`	The folder for the ZIP file could not be created.
	`ERuntimeErrorCode.ArchiveStorageN-otCreated`	The ZIP file could not be created.
	`ERuntimeErrorCode.StorageTransferEr-ror`	Error during network data transfer. Memory data between client and server computers do not match.

RetrieveStorage()

`RetrieveStorage()` creates a Virtual SIMATIC Memory Card from the archived ZIP file. The simulated PLC instance must be in OFF operating state for this.

Table 8-38 RetrieveStorage() - Native C++

Syntax	`ERuntimeErrorCode RetrieveStorage(WCHAR* in_FullFileName);`	
Parameters	• `WCHAR* in_FullFileName:` The full path to the ZIP file. The path relates to folders of the computer where the API is being called.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INVALID_OPERATING_STATE`	The instance is not in OFF operating state.
	`SREC_INVALID_ARCHIVE_PATH`	The archive path is invalid.
	`SREC_DELETE_EXISTING_STORAGE_FAILED`	The old storage cannot be deleted.
	`SREC_RETRIEVE_STORAGE_FAILURE`	The ZIP file cannot be unzipped.
	`SREC_STORAGE_TRANSFER_ERROR`	Error during network data transfer. Memory data between client and server computers do not match.
	`SREC_NO_STORAGE_PATH_SET`	No storage path is set. The user has to execute the SetStoragePath() function beforehand.

Table 8-39 RetrieveStorage() - .NET (C#)

Syntax	`void RetrieveStorage(string in_FullFileName);`	
Parameters	• `string in_FullFileName:` The full path to the ZIP file. The path relates to folders of the computer where the API is being called.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InvalidOperating-State`	The instance is not in OFF operating state.
	`ERuntimeErrorCode.InvalidArchivePath`	The archive path is invalid.
	`ERuntimeErrorCode.DeleteExisting-StorageFailed`	The old storage cannot be deleted.
	`ERuntimeErrorCode.RetrieveStorage-Failure`	The ZIP file cannot be unzipped.
	`ERuntimeErrorCode.StorageTransferEr-ror`	Error during network data transfer. Memory data between client and server computers do not match.

NOTE

Editing an instance created with S7-PLCSIM Advanced using S7-PLCSIM Advanced API V3.0 with the functions ArchiveStorage() and RetrieveStorage()

When you edit an instance created with S7-PLCSIM Advanced V4.0 or higher with the API functions ArchiveStorage() and RetrieveStorage() of V3.0, API V3.0 does not save the entire card content of the virtual memory card. The retain.pms and sim_hwdb.ini files are missing.

This means that the password to protect confidential configuration data is not saved and will be lost.

Solution:

You have a variety options:

- Set the password to protect the confidential configuration data before each simulation again using STEP 7, the SIMATIC Automation Tool, or TIA Portal Openness.
- Establish a remote connection of the instance via the Runtime Manager even when the instance communicates locally. In this case, the password is saved to the virtual memory card.
- Do not assign a password to protect confidential configuration data.

CleanupStoragePath()

The function deletes the folder with the Virtual SIMATIC Memory Card of a local instance or a remote instance. To do this, the function checks whether required and invalid files are available. Even if the folder is missing, the function is considered successful.

To make sure that the correct folder is deleted, the function checks whether there are any files that need to be present in the Virtual SIMATIC Memory Card:

- ".\sim_hwdb.ini"
- ".\SIMATIC_MC\SIMATIC.S7S\"
- ".\SIMATIC_MC\RData\"

To permanently delete the folder, only the following folders with files are also allowed:

- ".\CrashDump\"
- ".\Traces\"
- ".\retain.pms"

The instance must be in OFF operating state ("`PowerOff`").

Table 8-40 CleanupStoragePath() - Native C++

Syntax	`ERuntimeErrorCode CleanupStoragePath();`	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INVALID_OPERATING_STATE`	The instance is not in OFF operating state.
	`SREC_DELETE_EXISTING_STORAGE_FAILED`	The folder with the memory cannot be deleted.
	`SREC_INVALID_STORAGE`	The memory is invalid. It contains files or folders that are not permitted.

Table 8-41 CleanupStoragePath() - .NET (C#)

Syntax	`void CleanupStoragePath();`	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InvalidOperating-State`	The instance is not in OFF operating state.
	`ERuntimeErrorCode.DeleteExisting-StorageFailed`	The folder with the memory cannot be deleted.
	`SREC_INVALID_STORAGE`	The memory is invalid. It contains files or folders that are not permitted.

GetStrictMotionTiming() / StrictMotionTiming { get; }

Returns the current instance setting for the "Strict Motion Timing" feature. The "Strict Motion Timing" feature is located on the S7-PLCSIM Advanced Control Panel.

Table 8-42 GetStrictMotionTiming() - Native C++

Syntax	`ERuntimeErrorCode GetStrictMotionTiming(bool* enabled);`	
Parameters	bool* enabled: Receives the current setting. `true`: Active `false`: Inactive	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_TIMEOUT`	The function does not return on time.

Table 8-43 StrictMotionTiming { get; } - .NET (C#)

Syntax	`bool StrictMotionTiming { get; }`	
Parameters	None	
Return values	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.Timeout`	The function does not return on time.

SetStrictMotionTiming() / StrictMotionTiming { set; }

Sets the instance setting for the "Strict Motion Timing" feature. The "Strict Motion Timing" feature is located on the S7-PLCSIM Advanced Control Panel.

Table 8-44 SetStrictMotionTiming() - Native C++

Syntax	`ERuntimeErrorCode GetStrictMotionTiming(bool* enabled);`	
Parameters	bool enabled: The value to be set. `true`: Active `false`: Inactive	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INVALID_OPERATING_STATE`	The instance is in Power On. The instance must not be in Power On to change the setting.

Table 8-45 StrictMotionTiming { set; } - .NET (C#)

Syntax	`bool StrictMotionTiming { set; }`	
Parameters	None	
Return values	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InvalidOperating-State`	The instance is in Power On. The instance must not be in Power On to change the setting.

8.3 Operating state

PowerOn()

The function creates the process for the Simulation Runtime instance and starts the firmware of the simulated PLC instance.

The PowerOn() state is reached without a timeout value:

1. As soon as an instance was created
2. Before the user program has been started on the CPU

Table 8-46 PowerOn() - Native C++

Syntax	`ERuntimeErrorCode PowerOn();` `ERuntimeErrorCode PowerOn(UINT32 in_Timeout_ms);`	
Parameters	• `UInt32 in_Timeout_ms:` A timeout value in milliseconds. – If no timeout value is set, the function returns as soon as an instance is created and before the user program has been started on the CPU. Subscribe to the `OnOperatingStateChanged()` event to find out when the operation has been completed. – If the value is greater than 0 (a timeout value of at least 8500 ms is recommended), the function returns when the operation has been completed or after a timeout. – Expected operating states when this function is successful: { `SROS_STOP` or `SROS_RUN, SROS_FREEZE, SROS_HOLD` }	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The expected operating state does not occur on time.
	`SREC_ERROR_LOADING_DLL`	The "Siemens.Simatic.Simulation.Runtime.Instance.exe" cannot load the "Siemens.Simatic.PlcSim.Vplc1500.dll".
	`SREC_STORAGE_PATH_ALREADY_IN_USE`	The selected path for this instance is already being used by another instance.
	`SREC_NO_STORAGE_PATH_SET`	No storage path is set. The user has to execute the SetStoragePath() function beforehand.
	`SREC_WARNING_ALREADY_EXISTS`	Warning: The instance is started.
	`SREC_VIRTUAL_SWITCH_MISCONFIGURED`	• The S7-PLCSIM Advanced Virtual Switch is configured incorrectly. • The binding of an S7-PLCSIM Advanced Virtual Switch on a PC network interface is not correct, or the driver of an S7-PLCSIM Advanced Virtual Switch is not functioning properly. If the binding is not correct, call the `SetNetInterfaceBindings()` function or set the binding manually before starting the instance.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is no longer running.

Return values	SREC_UNSUPPORTED_PCAP_DRIVER	The PCAP driver used is not supported. S7-PLCSIM Advanced supports Npcap as of V0.9995. TCP/IP communication is not possible.
	SREC_WARNING_TRIAL_MODE_ACTIVE	Warning: No license available. You can use the instance without restrictions with the Trial License. Afterwards, the instance is shut down.
	SREC_WARNING_RUNNING_ON_TIA_PORTAL_-TEST_SUITE	Warning: No valid license is available, but a "TIA Portal Test Suite" license. S7-PLCSIM Advanced starts with this license. A download from the TIA Portal is possible, but the instance terminates without feedback if the download was not made from the TIA Portal Test Suite.
	SREC_NOT_EMPTY	Warning: No valid license for S7-PLCSIM Advanced is available, but a "TIA Portal Test Suite" license. If this is the case, power-up from the Virtual SIMATIC Memory Card is not supported.
	SREC_COMMUNICATION_INTERFACE_NOT_AV-AILABLE	**For local communication via Softbus** S7-PLCSIM Advanced cannot connect to the Softbus. **Solution** • Try again to establish the connection. • Close S7-PLCSIM Advanced and the TIA Portal and restart the applications. • Reboot the PC. • Repair the S7-PLCSIM Advanced installation. **For TCP/IP communication** Another application is connected to the Softbus on your PC. **Solution** • Close all SIMATIC applications, e.g. TIA Portal, WinCC, PLCSIM. • Reboot the PC. • Repair the S7-PLCSIM Advanced installation.
	SREC_ACCESS_DENIED	No write access in the Storage Directory.
	SREC_PCAP_DRIVER_NOT_RUNNING	The Nmap Packet Capture Driver (Npcap) is not active on the system. **Remedy** 1. Start the command line in administrator mode. 2. Execute the command "net start npcap".
	SREC_WRONG_COMMUNICATION_INTERFACE	• You have tried to supply the second instance with the communication interface that is different from the first instance that is already switched on. – **Example:** 1. PowerOn() of the first instance with Softbus is successful. 2. PowerOn() of the second instance with TCP/IP; the error code is returned. • The specified/mapped computer interface in the sim_hwdb.ini file is disabled or it is the PLCSIM Virtual Adapter.

Return values	SREC_WARNING_PASSWORD_PROTECTION_ERROR	The password protection prevented the memory from being opened. View the restrictions with activated password protection.
	SREC_INVALID_CONFIGURATION	The configuration is invalid because, for example, the current instance has a mapped PC network interface that is already being used by another running instance.
	SREC_WARNING_NO_EXTERNAL_COMMUNICATION	You have set the Multi-Adapter Network Mode (non-promiscuous mode) without interface mapping or Softbus as communication interface.
	SREC_INTERNAL_ERROR	The system call has caused an internal error.
	SREC_LICENSE_NOT_FOUND	This error occurs under one of the following conditions: • There is no valid license available in the Automation License Manager • The Automation License Manager is corrupt
	SREC_CONFIGURED_DEVICE_NOT_FOUND	The article number you provided is invalid.

Table 8-47 PowerOn() - .NET (C#)

Syntax	`ERuntimeErrorCode PowerOn();` `ERuntimeErrorCode PowerOn(UInt32 in_Timeout_ms);`	
Parameters	• `UInt32 in_Timeout_ms:` A timeout value in milliseconds. – If no timeout value is set, the function returns as soon as an instance is created and before the user program has been started on the CPU. Subscribe to the `OnOperatingStateChanged()` event to find out when the operation has been completed. – If the value is greater than 0 (a timeout value of at least 8500 ms is recommended), the function returns when the operation has been completed or after a timeout. – Expected operating states when this function is successful: { `EOperatingState.Run` or `EOperatingState.Stop, EOperatingState.Freeze, EOperatingState.Hold`}	
Return values	Runtime error code	Description
	`ERuntimeErrorCode.OK`	The function is successful.
	`ERuntimeErrorCode.WarningAlreadyExists`	Warning: The instance is started.
	`ERuntimeErrorCode.UnsupportedPcapDriver`	The PCAP driver used is not supported. S7-PLCSIM Advanced supports Npcap as of V0.9995. TCP/IP communication is not possible.
	`ERuntimeErrorCode.WarningTrialModeActive`	Warning: No license available. You can use the instance without restrictions with the Trial License. Afterwards, the instance is shut down.
	`ERuntimeErrorCode.WarningRunningOnTiaPortalTestSuite`	Warning: No valid license is available, but a "TIA Portal Test Suite" license. S7-PLCSIM Advanced starts with this license. A download from the TIA Portal is possible, but the instance terminates without feedback if the download was not made from the TIA Portal Test Suite.

Return values	ERuntimeErrorCode.NotEmpty	Warning: No valid license for S7-PLCSIM Advanced is available, but a "TIA Portal Test Suite" license. If this is the case, power-up from the Virtual SIMATIC Memory Card is not supported.
	ERuntimeErrorCode.CommunicationInterfaceNotAvailable	**For local communication via Softbus** S7-PLCSIM Advanced cannot connect to the Softbus. **Solution** • Try again to establish the connection. • Close S7-PLCSIM Advanced and the TIA Portal and restart the applications. • Reboot the PC. • Repair the S7-PLCSIM Advanced installation. **For TCP/IP communication** Another application is connected to the Softbus on your PC. **Solution** • Close all SIMATIC applications, e.g. TIA Portal, WinCC, PLCSIM. • Reboot the PC. • Repair the S7-PLCSIM Advanced installation.
	ERuntimeErrorCode.AccessDenied	No write access in the Storage Directory.
	ERuntimeErrorCode.PCAPDriverNotRunning	The Nmap Packet Capture Driver (Npcap) is not active on the system. **Remedy** 1. Start the command line in administrator mode. 2. Execute the command "net start npcap".
	ERuntimeErrorCode.WrongCommunicationInterface	• You have tried to supply the second instance with the communication interface that is different from the first instance that is already switched on. – Example: 1. PowerOn() of the first instance with Softbus is successful. 2. PowerOn() of the second instance with TCP/IP; the error code is returned. • The specified/mapped computer interface in the sim_hwdb.ini file is disabled or it is the PLCSIM Virtual Adapter.
	ERuntimeErrorCode.WarningPasswordForProtectionError	The password protection prevented the memory from being opened. View the restrictions with activated password protection.
	ERuntimeErrorCode.WarningNoExternalCommunication	You have set the Multi-Adapter Network Mode (non-promiscuous mode) without interface mapping or Softbus as communication interface.
Exceptions	Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException	
	Runtime error code	Description
	ERuntimeErrorCode.InterfaceRemoved	The instance is not registered in Runtime Manager.
	ERuntimeErrorCode.Timeout	The expected operating state does not occur on time.

Exceptions	`ERuntimeErrorCode.ErrorLoadingDll`	The "Siemens.Simatic.Simulation. Runtime.Instance.exe" cannot load the "Siemens.Simatic.PlcSim.Vplc1500.dll".
	`ERuntimeErrorCode.StoragePath-AlreadyInUse`	The selected path for this instance is already being used by another instance.
	`ERuntimeErrorCode.NoStoragePathSet`	The path could not be created. The length of the `DSTORAGE_PATH_MAX_LENGTH` characters might be exceeded.
	`ERuntimeErrorCode.VirtualSwitchMis-configured`	• The S7-PLCSIM Advanced Virtual Switch is configured incorrectly. • The binding of an S7-PLCSIM Advanced Virtual Switch on a PC network interface is not correct, or the driver of an S7-PLCSIM Advanced Virtual Switch is not functioning properly. If the binding is not correct, call the `SetNetInterfaceBindings()` function or set the binding manually before starting the instance.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is no longer running.
	`ERuntimeErrorCode.InvalidConfigura-tion`	The configuration is invalid because, for example, the current instance has a mapped PC network interface that is already being used by another running instance.
	`ERuntimeErrorCode.InternalError`	The system call has caused an internal error.
	`ERuntimeErrorCode.LicenseNotFound`	This error occurs under one of the following conditions: • There is no valid license available in the Automation License Manager • The Automation License Manager is corrupt
	`ERuntimeErrorCode.ConfiguredDevi-ceNotFound`	The article number you provided is invalid.

PowerOff()

Shuts down the Simulation Runtime and closes its process.

Table 8-48 PowerOff() - Native C++

Syntax	`ERuntimeErrorCode PowerOff();` `ERuntimeErrorCode PowerOff(UINT32 in_Timeout_ms);`	
Parameters	• `UINT32 in_Timeout_ms:` A timeout value in milliseconds. – If no timeout value is set, the function returns immediately. Subscribe to the `OnOperatingStateChanged()` event to find out when the operation has been completed. – If the value is greater than 0 (a timeout value of at least 800 ms is recommended), the function returns when the operation has been completed or after a timeout. Expected operating state when this function is successful: `{ SROS_OFF }`	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.

Return values	SREC_INTERFACE_REMOVED	The instance is not registered in Runtime Manager.
	SREC_TIMEOUT	The expected operating state does not occur on time.
	SREC_INSTANCE_NOT_RUNNING	The process of the simulated PLC instance is not running.

Table 8-49 PowerOff() - .NET (C#)

Syntax	void PowerOff(); void PowerOff(UInt32 in_Timeout_ms);	
Parameters	• UInt32 in_Timeout_ms: A timeout value in milliseconds. – If no timeout value is set, the function returns immediately. Subscribe to the OnOperatingStateChanged() event to find out when the operation has been completed. – If the value is greater than 0 (a timeout value of at least 800 ms is recommended), the function returns when the operation has been completed or after a timeout. Expected operating state when this function is successful: { EOperatingState.Stop }	
Return values	None	
Exceptions	Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException	
	Runtime error code	Description
	ERuntimeErrorCode.InterfaceRemoved	The instance is not registered in Runtime Manager.
	ERuntimeErrorCode.Timeout	The expected operating state does not occur on time.
	ERuntimeErrorCode.InstanceNotRunning	The process of the simulated PLC instance is not running.

Run()

Changes the simulated PLC instance to the RUN operating state.

Table 8-50 Run() - Native C++

Syntax	ERuntimeErrorCode Run(); ERuntimeErrorCode Run(UINT32 in_Timeout_ms);	
Parameters	• UINT32 in_Timeout_ms: A timeout value in milliseconds. – If no timeout value is set, the function returns immediately. Subscribe to the OnOperatingStateChanged() event to find out when the operation has been completed. – If the value is greater than 0 (a timeout value of at least 20 ms is recommended), the function returns when the operation has been completed or after a timeout. Expected operating states if this function is successful: { SROS_STOP , SROS_RUN }	
Return values	Runtime error code	Description
	SREC_OK	The function is successful.
	SREC_INTERFACE_REMOVED	The instance is not registered in Runtime Manager.

Return values	SREC_TIMEOUT	The expected operating state does not occur on time.
	SREC_INSTANCE_NOT_RUNNING	The process of the simulated PLC instance is not running.
	SREC_IS_EMPTY	One of the following two conditions applies here: **Condition:** After creating and starting an instance, you have tried to switch the simulated PLC instance to the RUN operating state without downloading a STEP 7 project. **Condition:** After creating and enabling an instance with download of a STEP 7 project with enabled password for the protection of confidential configuration data, you have moved the virtual memory card from one system to another. You have tried to start the instance on the new system. The following error code is returned after the start: "SREC_WARNING_PASSWORD_PROTECTION_ERROR" Afterwards, you have tried to switch the simulated PLC instance to the RUN operating state.

Table 8-51 Run() - .NET (C#)

Syntax	`void Run();` `void Run(Uint32 in_Timeout_ms);`	
Parameters	• `UInt32 in_Timeout_ms:` A timeout value in milliseconds. – If no timeout value is set, the function returns immediately. Subscribe to the `OnOperatingStateChanged()` event to find out when the operation has been completed. – If the value is greater than 0 (a timeout value of at least 20 ms is recommended), the function returns when the operation has been completed or after a timeout. Expected operating states when this function is successful: `{ EOperatingState.Run }`	
Exceptions	Runtime error code	Description
	ERuntimeErrorCode.IsEmpty	One of the following two conditions applies here: **Condition:** After creating and starting an instance, you have tried to switch the simulated PLC instance to the RUN operating state without downloading a STEP 7 project. **Condition:** After creating and enabling an instance with download of a STEP 7 project with enabled password for the protection of confidential configuration data, you have moved the virtual memory card from one system to another. You have tried to start the instance on the new system. The following error code is returned after the start: "SREC_WARNING_PASSWORD_PROTECTION_ERROR" Afterwards, you have tried to switch the simulated PLC instance to the RUN operating state.
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description

Exceptions	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The expected operating state does not occur on time.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.

RUN operating state after enabling an instance

When you expect the RUN operating state after enabling an instance, check whether the instance is already in RUN by subscribing to the event `OnOperatingStateChanged()`. As an alternative, call `Run()` immediately after enabling the API function to switch the instance to the RUN operating state.

Stop()

Changes the simulated PLC instance to the STOP operating state.

Table 8-52 Stop() - Native C++

Syntax	`ERuntimeErrorCode Stop();` `ERuntimeErrorCode Stop(UINT32 in_Timeout_ms);`	
Parameters	• `UINT32 in_Timeout_ms:` A timeout value in milliseconds. – If no timeout value is set, the function returns immediately. Subscribe to the `OnOperatingStateChanged()` event to find out when the operation has been completed. – If the value is greater than 0 (a timeout value of at least 200 ms is recommended), the function returns when the operation has been completed or after a timeout. Expected operating state when this function is successful: `{ SROS_STOP }`	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The expected operating state does not occur on time.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.

Table 8-53 Stop() - .NET (C#)

Syntax	`void Stop();` `void Stop(UInt32 in_Timeout_ms:);`	
Parameters	• `UInt32 in_Timeout_ms:` A timeout value in milliseconds. – If no timeout value is set, the function returns immediately. Subscribe to the `OnOperatingStateChanged()` event to find out when the operation has been completed. – If the value is greater than 0 (a timeout value of at least 200 ms is recommended), the function returns when the operation has been completed or after a timeout. Expected operating state if this function is successful: `{ EOperatingState.Stop }`	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description

Exceptions	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The expected operating state does not occur on time.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.

GetOperatingState() / OperatingState { get; }

Returns the operating state of the simulated PLC instance. When the operating state changes, the OnOperatingStateChanged() (Page 191) event is triggered. For details about the operating state, see Data types (Page 259).

Table 8-54 GetOperatingState() - Native C++

Syntax	`EOperatingState GetOperatingState();`
Parameters	None
Return values	• `SROS_INVALID_OPERATING_STATE:` If the function fails. • `SROS_OFF:` If the Simulation Runtime instance is not running. • `SROS_BOOTING:` If `PowerOn()` was called while in this state and the simulated PLC instance is not yet ready to start the user program. • `SROS_STOP:` If the simulated PLC instance is in STOP state. • `SROS_STARTUP:` If the user program is currently changing from STOP to RUN. • `SROS_RUN:` If the user program is running. • `SROS_FREEZE:` If the user program is being stopped (Freeze status). • `SROS_HOLD:` If the user program is set to HOLD when the breakpoint is reached. • `SROS_SHUTTING_DOWN:` If `PowerOff()` was called but the simulated PLC instance is still in the Shutdown phase.

Table 8-55 OperatingState { get; } - .NET (C#)

Syntax	`EOperatingState OperatingState { get; }`
Parameters	None
Return values	• `EOperatingState.InvalidOperatingState:` If the function fails. • `EOperatingState.Off:` If the Simulation Runtime instance is not running. • `EOperatingState.Booting:` If `PowerOn()` was called while in this state and the simulated PLC instance is not yet ready to start the user program. • `EOperatingState.Stop:` If the simulated PLC instance is in STOP state. • `EOperatingState.Startup:` If the user program is currently changing from STOP to RUN. • `EOperatingState.Run:` If the user program is running. • `EOperatingState.Freeze:` If the user program is being stopped (Freeze status).

	• `EOperatingState.Hold`: If the user program is set to HOLD when the breakpoint is reached. • `EOperatingState.ShuttingDown`: If `PowerOff()` was called but the simulated PLC instance is still in the Shutdown phase.

MemoryReset()

Shuts down the simulated PLC instance, closes its processes and performs a restart.

Closing the processes automatically performs a general reset.

Table 8-56 MemoryReset() - Native C++

Syntax	`ERuntimeErrorCode MemoryReset();` `ERuntimeErrorCode MemoryReset(UINT32 in_Timeout_ms);`	
Parameters	• `UINT32 in_Timeout_ms:` A timeout value in milliseconds. – If no timeout value is set, the function returns immediately. Subscribe to the `OnOperatingStateChanged()` event to find out when the operation has been completed. – If the value is greater than 0 (a value of 60000 is recommended), the function returns when the operation has been completed or after a timeout. Expected operating states if this function is successful: `{ SROS_STOP , SROS_RUN }`	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The expected operating state does not occur on time.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.

Table 8-57 MemoryReset() - .NET (C#)

Syntax	`void MemoryReset();` `void MemoryReset(UInt32 in_Timeout_ms);`	
Parameters	• `UInt32 in_Timeout_ms:` A timeout value in milliseconds. – If no timeout value is set, the function returns immediately. Subscribe to the `OnOperatingStateChanged()` event to find out when the operation has been completed. – If the value is greater than 0 (a value of 60000 is recommended), the function returns when the operation has been completed or after a timeout. Expected operating states when this function is successful: `{ EOperatingState.Run, EOperatingState.Stop }`	
Return values	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The expected operating state does not occur on time.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.

8.4 Stop simulation

Freeze state of the simulated PLC instance

To stop a simulation and synchronize simulation partners, you can set a simulated PLC instance to a freeze state via the API. When the instance has reached a synchronization point, the instance sends the event `OnSyncPointReached` to the API clients.

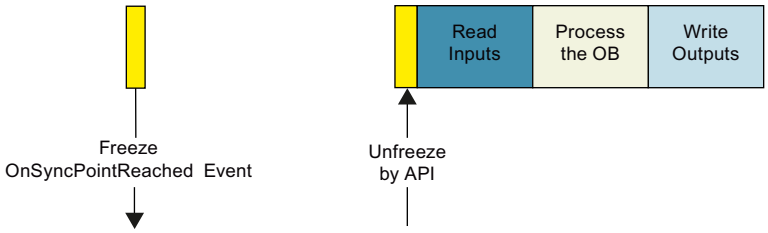


Figure 8-1 Freeze state of the simulated PLC instance

The following occurs in the freeze state:

- The virtual time is stopped.
- The OBs currently being executed are stopped.

  The procedure corresponds to the behavior of a program that has reached a breakpoint.
- The user program is no longer executed.
- The simulated PLC instance is still accessible from the TIA Portal.

NOTE

Freeze state during downloading

To complete a download in freeze state, the simulated PLC instance must pass through a suspended or terminated freeze state.

NOTE

Freeze-state ≠ operating state

The freeze state is an internal operating state of the simulated PLC instance. It does not correspond to RUN/STOP mode of a CPU. In the freeze state, the instance maintains the last operating state.

- The LED display on the Control Panel and on the Web server accordingly shows RUN or STOP for the instance.
- The instance shows the operating state `SROS_FREEZE` / `Freeze`, see EOperatingState (Page 259).

NOTE

Reaction of an S7-PLCSIM Advanced API to a freeze state

If you use a SingleStep operating mode, you must register a callback function for the `OnSyncPointReached` event. Calling the mode-specific `RunToNextSyncPoint` method ensures that the freeze state of the simulated PLC instance is exited again.

Synchronization points

A synchronization point always exists **before** inputs are read in, for example at the cycle control point or at the beginning of a cyclic OB.

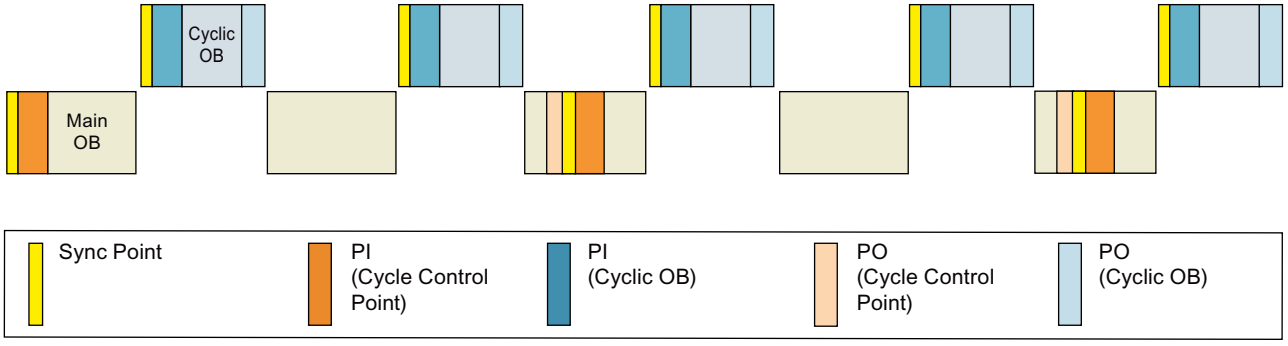


Figure 8-2 Overview of the synchronization points

Trigger freeze state

To trigger the freeze state, following modes are available for the simulated PLC instance:

- SingleStep operating modes

  See Synchronize simulation partner cycle-controlled (Page 186).
- TimespanSynchronized operating modes

  See Synchronize simulation partner time-controlled (Page 190).

In Default operating mode, the simulated PLC instance does not change into a freeze state.

API functions

- Settings for the cycle control (Page 163)
- GetOperatingMode() (Page 163)
- SetOperatingMode() (Page 163)
- OperatingMode { get; set; } (Page 163)
- EOperatingMode (Page 260)

8.5 Tag list

NOTE

Elements with data types not known to the API (`EDataType.Unknown`) are not included in the tag list.

UpdateTagList()

The function reads the tags from the simulated PLC instance and writes them to the shared storage arranged by name.

If the tag is an array or a structure, there are multiple entries.

In the case of a structure, there is an entry for the structure itself and an additional entry for each structure element.

Entry_1: "StructName"

Entry_2: "StructName.ElementName_1"

..

Entry_N: "StructName.ElementName_n"

In the case of an array, in this example a two-dimensional array, there is an entry for the array itself and an additional entry for each array element.

Entry_1: "ArrayName"

Entry_2: "ArrayName[a,b]", {a} and {b} correspond to the first index of the respective dimension)

..

Entry_N: "ArrayName[x,y]", {x} and {y} correspond to the last index of the respective dimension)

Memory for up to 500000 entries (not PLC tags) is reserved for the list. If the list becomes too large, the function returns the error/exception "NOT_ENOUGH_MEMORY".

If there are problems with the maximum number of entries and not all tags are needed, two filters can be used when refreshing the tag table.

Table 8-58 UpdateTagList() - Native C++

Syntax	`ERuntimeErrorCode UpdateTagList();`
	`ERuntimeErrorCode UpdateTagList(ETagListDetails in_TagListDetails);`
	`ERuntimeErrorCode UpdateTagList(ETagListDetails in_TagListDetails, bool in_IsHMIVisibleOnly);`
	`ERuntimeErrorCode UpdateTagList(ETagListDetails in_TagListDetails, bool in_IsHMIVisibleOnly, WCHAR* in_DataBlockFilterList);`

Parameters	• `ETagListDetails in_TagListDetails`: Every combination of the following four areas: **IO**: Inputs and Outputs **M**: Bit memory **CT**: Counters and Timers **DB**: Data Blocks The default setting is **IOMCTDB**. **Example: IOM** reads only the tags from the area Inputs / Outputs and Bit memory. • `bool in_IsHMIVisibleOnly`: If `true`, only tags marked with "HMI Visible" are read. The default setting is `true`. • `WCHAR* in_DataBlockFilterList`: A string that includes the name of all data blocks that are supposed to be available in the tag table. The string must be in quotation marks. **Example:** ""\"DB_1\", \"DB_2\" \"DB_3\"|\"DB_4\"\"DB_5\"" All characters within the quotation marks are interpreted as a DB name. If the data block does not exist in the PLC program, it is not added to the tag table memory. No error is triggered in the process.

	For this list to be taken into consideration, `in_DataBlockFilterList` has to be unequal to NULL and `in_TagListDetails` has to contain "DB".	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.
	`SREC_NOT_ENOUGH_MEMORY`	More than 500000 entries are requested.
	`SREC_WARNING_ALREADY_EXISTS`	The tag table is current.
	`SREC_WRONG_ARGUMENT`	The syntax of `in_DataBlockFilterList` is invalid. The list has to be 3 characters long; the first and last character have to be a quotation mark.
Example C++		

```
ISimulationRuntimeManager * api = NULL;
ERuntimeErrorCode result = InitializeApi(&api);

IInstance* psa = NULL;
if (result == SREC_OK)
{
result = api->RegisterInstance(&psa);
}

if (result == SREC_OK)
{
// Example: How to filter data blocks
// DB Name: TEST_DB, DB Number: 2
result = psa->UpdateTagList(SRTLD_MDB, false, L"\"TEST_DB\"");
}
```

Table 8-59 UpdateTagList() - .NET (C#)

Syntax	

```
void UpdateTagList();
void UpdateTagList(
 ETagListDetails in_TagListDetails
);
void UpdateTagList(
 ETagListDetails in_TagListDetails,
bool in_IsHMIVisibleOnly
);
ERuntimeErrorCode UpdateTagList(
 ETagListDetails in_TagListDetails,
  bool in_IsHMIVisibleOnly,
  string in_DataBlockFilterList
);
```

Parameters	• `ETagListDetails in_TagListDetails`: Every combination of the following four areas: **IO**: Inputs and Outputs **M**: Bit memory **CT**: Counters and Timers **DB**: Data Blocks The default setting is **IOMCTDB**. **Example: IOM** reads only the tags from the area Inputs / Outputs and Bit memory. • `bool in_IsHMIVisibleOnly`: If `true`, only tags marked with "HMI Visible" are read. The default setting is `true`. • `string in_DataBlockFilterList`:

	A string that includes the name of all data blocks that are supposed to be available in the tag table. The string must be in quotation marks. **Example:** ""\"DB_1\", \"DB_2\" \"DB_3\"|\"DB_4\"\"DB_5\"" All characters within the quotation marks are interpreted as a DB name. If the data block does not exist in the PLC program, it is not added to the tag table memory. No error is triggered in the process. For this list to be taken into consideration, `in_DataBlockFilterList` has to be unequal to NULL and `in_TagListDetails` has to contain "DB".	
Return values	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.
	`ERuntimeErrorCode.NotEnoughMemory`	More than 500000 entries are requested.
	`ERuntimeErrorCode.WrongArgument`	The syntax of `in_DataBlockFilterList` is invalid. The list has to be 3 characters long; the first and last character have to be a quotation mark.

GetTagListStatus()

Returns the current update status of the tag list storage.

"`inout_TagListDetails`" is `NONE`, if the list needs to be updated.

Table 8-60 GetTagListStatus() - Native C++

Syntax	`ERuntimeErrorCode GetTagListStatus(ETagListDetails* out_TagListDetails, bool* out_IsHMIVisibleOnly);`	
Parameters	• `ETagListDetails out_TagListDetails`: Status of the tag list details. `SRTLD_NONE` when an update of the list is required. • `bool out_IsHMIVisibleOnly`: If `true`, only tags marked with "HMI Visible" are available in the list.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.

Table 8-61 GetTagListStatus() - .NET (C#)

Syntax	`void GetTagListStatus(out ETagListDetails out_TagListDetails, out bool out_IsHMIVisibleOnly);`
Parameters	• `out ETagListDetails out_TagListDetails`: Status of the tag list details. `ETagListDetails.None` when an update of the list is required. • `out bool out_IsHMIVisibleOnly`: If `true`, only tags marked with "HMI Visible" are available in the list.
Return values	None
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`

Exceptions	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.

GetTagInfoCount()

Returns the number of entries in the tag list storage. If the function fails, the return value is 0.

Table 8-62 GetTagInfoCount() - Native C++

Syntax	`UINT32 GetTagInfoCount();`
Parameters	None
Return values	Number of entries in the tag list storage.

GetTagInfos() / TagInfos { get; }

Returns a list of all tags.

Table 8-63 GetTagInfos() - Native C++

Syntax	`ERuntimeErrorCode GetTagInfos(UINT32 in_BufferLength, STagInfo* inout_TagInfos, UINT32* out_TagCount);`	
Parameters	• `UINT32 in_BufferLength:` The number of elements that the storage can accommodate. • `STagInfo* inout_TagInfos:` The user-allocated storage that accommodates the tags. • `UINT32* out_TagCount:` Returns the number of tags that were written to the storage.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.
	`SREC_INDEX_OUT_OF_RANGE`	The elements do not fit in the storage.

Table 8-64 TagInfos { get; } - .NET (C#)

Syntax	`STagInfo[] TagInfos { get; }`	
Parameters	None	
Return values	An array that contains all available entries of the storage.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.

CreateConfigurationFile()

Writes all entries from the tag list to an XML file.

Table 8-65 CreateConfigurationFile() - Native C++

Syntax	`ERuntimeErrorCode CreateConfigurationFile(WCHAR* in_FullFileName);`	
Parameters	• `WCHAR* in_FullFileName:` Full file name of the XML file: <Path> + <File name> + <File extension>.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_WRONG_ARGUMENT`	The file name is invalid.

Table 8-66 CreateConfigurationFile() - .NET (C#)

Syntax	`void CreateConfigurationFile(string in_FullFileName);`	
Parameters	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.WrongArgument`	The file name is invalid.

8.6 I/O access

8.6.1 Synchronizing inputs and outputs

Description

In S7-PLCSIM Advanced the complete scope of the input and output area is used (see GetAreaSize/AreaSize (Page 107)). This is also possible when no I/O module is configured.

Inputs and outputs which are defined via configured I/O modules are synchronized to the defined update of the process image partition (PIP).

Inputs and outputs which are not assigned to an I/O module are synchronized in the cycle control point.

Note the following when synchronizing these inputs and outputs:

- Inputs can only be used as inputs.

  You can write values via the API. However, values written via STEP 7 or via communication to the input area will not be visible in the API. For example, if you go online in the TIA Portal and write data to the input area, those data values are not visible in the API. Methods such as ReadBytes() only read values written from the API.

- Outputs can be used as output and as input.

  You can write values via the API and via the CPU / STEP 7 user program. If API and user program write to the same area, the values from the API will overwrite the values from the user program.

NOTE

Conditions for synchronizing Fail-safe I/O

Fail-safe inputs and outputs are only synchronized if a Fail-safe module is plugged in and the Fail-safe program uses at least one bit of the module.

8.6.2 Simulate peripheral I/O

S7-PLCSIM Advanced writes to and reads from a memory area. This memory is synchronized with the internal process image of the simulated S7-1500 controller at the cycle control point and when calling cyclic and acyclic OBs (process image partitions, interrupts, events). The direct I/O accesses are made to this memory area. Only one process can access this memory at a given time.

The instance must be in RUN mode to apply changes made by the API.

NOTE

Precedence of write operations

If the STEP 7 user program is writing to a memory range at the same time an API application is writing to the same memory range, the writes from the API application take precedence. Writes from the API application override writes from the STEP 7 program to the simulated PLC instance.

8.6.3 I/O access via address - Reading

InputArea { get; }, MarkerArea { get; }, OutputArea { get; }

Returns an interface that you use to call the .NET functions in this section.

Table 8-67 InputArea { get; } MarkerArea { get; } OutputArea { get; } - .NET (C#)

Syntax	`IIOArea InputArea { get; }`
	`IIOArea MarkerArea { get; }`
	`IIOArea OutputArea { get; }`

Parameters	None
Return values	`IIOArea`: The interface is used to call the "I/O access via address" functions.

GetAreaSize() / AreaSize { get; }

Returns the size of the area in bytes.

Table 8-68 GetAreaSize() - Native C++

Syntax	`UINT32 GetAreaSize(EArea in_Area);`
Parameters	• `EArea in_Area:` The area whose size you want to receive. Permissible values: `{SRA_INPUT, SRA_MARKER, SRA_OUTPUT}`. See EArea (Page 259).
Return values	`UINT32:` Size of the area in bytes. If the function was successful, the value is not equal to 0.

Table 8-69 AreaSize { get; } - .NET (C#)

Syntax	`UInt32 InputArea.AreaSize { get; }`
	`UInt32 MarkerArea.AreaSize { get; }`
	`UInt32 OutputArea.AreaSize { get; }`

Parameters	None
Return values	`Uint32:` Size of the area in bytes. If the function was successful, the value is not equal to 0.

ReadBit()

Reads an individual bit from the area.

NOTE

The function allows access to the entire storage area of the simulated PLC instance. Therefore, use access via the tag name (Page 118) and not via the address areas.

Table 8-70 ReadBit() - Native C++

Syntax	`ERuntimeErrorCode ReadBit(EArea in_Area, UINT32 in_Offset, UINT8 in_Bit, bool* out_Value);`	
Parameters	• `EArea in_Area:` The area to be read from. Permissible values: `{SRA_INPUT, SRA_MARKER, SRA_OUTPUT}`. See EArea (Page 259). • `UINT32 in_Offset:` The byte offset within the area. The value must be between 0 and the value that `GetAreaSize()` returns. • `UINT8 in_Bit:` The bit offset within the byte. The value must be between 0 and 7. • `bool* out_Value:` Returns the bit value.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.
	`SREC_INDEX_OUT_OF_RANGE`	Offset or bits are invalid.
	`SREC_WRONG_ARGUMENT`	The area is invalid.

Table 8-71 ReadBit() - .NET (C#)

Syntax	`bool InputArea.ReadBit(UInt32 in_Offset, Byte in_Bit);`
	`bool MarkerArea.ReadBit(UInt32 in_Offset, Byte in_Bit);`
	`bool OutputArea.ReadBit(UInt32 in_Offset, Byte in_Bit);`	

Parameters	• `UInt32 in_Offset:` The byte offset within the area. The value must be between 0 and the value that `AreaSize` returns. • `Byte in_Bit:` The bit offset within the byte. The value must be between 0 and 7.	
Return values	`bool:` Bit value	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.
	`ERuntimeErrorCode.IndexOutOfRange`	Offset or bits are invalid.

ReadByte()

Reads an individual bit from the area.

NOTE

The function allows access to the entire storage area of the simulated PLC instance. Therefore, use access via the tag name and not via the address areas.

Table 8-72 ReadByte() - Native C++

Syntax	`ERuntimeErrorCode ReadByte(EArea in_Area, UINT32 in_Offset, BYTE* out_Value);`	
Parameters	• `EArea in_Area:` The area from which you want to read. Permissible values: `{SRA_INPUT, SRA_MARKER, SRA_OUTPUT}`. See EArea (Page 259). • `UINT32 in_Offset:` The byte offset within the area. The value must be between 0 and the value that `GetAreaSize()` returns. • `BYTE* out_Value:` Returns the byte value.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.
	`SREC_INDEX_OUT_OF_RANGE`	Offset is invalid.
	`SREC_WRONG_ARGUMENT`	The area is invalid.

Table 8-73 ReadByte() - .NET (C#)

Syntax	`Byte InputArea.ReadByte(UInt32 in_Offset);`
	`Byte MarkerArea.ReadByte(UInt32 in_Offset);`
	`Byte OutputArea.ReadByte(UInt32 in_Offset);`	

Parameters	• `UInt32 in_Offset:` The byte offset within the area. The value must be between 0 and the value that `AreaSize` returns.	
Return values	`Byte:` Byte value.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.
	`ERuntimeErrorCode.IndexOutOfRange`	Offset is invalid.

ReadBytes()

Reads a byte array from the area.

NOTE

The function allows access to the entire storage area of the simulated PLC instance. Therefore, use access via the tag name and not via the address areas.

Table 8-74 ReadByte() - Native C++

Syntax	`ERuntimeErrorCode ReadBytes(EArea in_Area, UINT32 in_Offset, UINT32 in_BytesToRead, UINT32* out_BytesRead, BYTE inout_Values[]);`	
Parameters	• `EArea in_Area:` The area from which you want to read. Permissible values: `{SRA_INPUT, SRA_MARKER, SRA_OUTPUT}`. See EArea (Page 259). • `UINT32 in_Offset:` The byte offset within the area. The value must be between 0 and the value that `GetAreaSize()` returns. • `UINT32 in_BytesToRead:` Contains the size of the value storage. • `UINT32* out_BytesRead:` Returns the number of bytes that were just written to the value storage. • `BYTE inout_Values[]:` The storage for the bytes that are read from the area.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.

Return values	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.
	`SREC_INDEX_OUT_OF_RANGE`	The offset lies outside the area range. No byte could be read.
	`SREC_WRONG_ARGUMENT`	The area is invalid.

Table 8-75 ReadBytes() - .NET (C#)

Syntax	`Byte[] InputArea.ReadBytes(UInt32 in_Offset, UInt32 in_BytesToRead);`
	`Byte[] MarkerArea.ReadBytes(UInt32 in_Offset,UInt32 in_BytesToRead);`
	`Byte[] OutputArea.ReadBytes(UInt32 in_Offset,UInt32 in_BytesToRead);`	

Parameters	• `UInt32 in_Offset:` The byte offset within the area. The value must be between 0 and the value that `AreaSize` returns. • `UInt32 in_BytesToRead:` The number of bytes to be read.	
Return values	`Byte[]:` The read bytes.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.
	`ERuntimeErrorCode.IndexOutOfRange`	The offset lies outside the area range. No byte could be read.

ReadSignals()

Structures and fields can be emulated through signal lists and be read by using the `ReadSignals()` function.

The function also takes into consideration the byte order (Endianness).

Only primitive data type signals are supported, but the function is not type-safe.

NOTE

The function allows access to the entire storage area of the simulated PLC instance.

Therefore, use access via the tag name (Page 118) and not via the address areas.

Table 8-76 ReadSignals() - Native C++

Syntax	`ERuntimeErrorCode ReadSignals(EArea in_Area, SDataValueByAddress* inout_Signals, UINT32 in_SignalCount, bool validateAddressTag = false);` `ERuntimeErrorCode ReadSignals(EArea in_Area, SDataValueByAddressWithCheck* inout_Signals, UINT32 in_SignalCount, bool* out_SignalsHaveChanged, bool validateAddressTag = false);`	
Parameters	• `EArea in_Area` The area from which you want to read. Permissible values: `{SRA_INPUT, SRA_MARKER, SRA_OUTPUT}`. See EArea (Page 259). • `SDataValueByAddress* inout_Signals` The signal list that is read. The result is stored in the structure. • `SDataValueByAddressWithCheck* inout_Signals` The signal list that is read. The result is stored in the structure. "`ValueHasChanged`" is set to `true` if the value of the signal has changed since the preceding call. • `UINT32 in_SignalCount` Number of signals in the list. • `bool* out_SignalsHaveChanged` Returns `true` if the value of at least one signal has changed since the preceding call. • `bool validateAddressTag` **B**oolean value to enable or disable the Memory Address to Tag association check.	
Signal error	Error code	Description
	`SREC_OK`	The signal operation is successful.
	`SREC_DOES_NOT_EXIST`	The entry does not exist in the stored tag list.
	`SREC_NOT_SUPPORTED`	Access to entire structures or arrays is not supported.
	`SREC_TYPE_MISMATCH`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.
	`SREC_WRONG_ARGUMENT`	The area is invalid.
	`SREC_SIGNAL_CONFIGURATION_ERROR`	At least one signal error is in the list.

Table 8-77 ReadSignals() - .NET (C#)

Syntax	`void ReadSignals(ref SDataValueByAddress[] inout_Signals, bool validateAddressTag);` `void ReadSignals(ref SDataValueByAddressWithCheck[] inout_Signals out bool out_SignalsHaveChanged, bool validateAddressTag);`	
Parameters	• `ref SDataValueByAddress[] inout_Signals` The signal list that is read. • `ref SDataValueByAddressWithCheck[] inout_Signals` The signal list that is read. The result is stored in the structure. "`ValueHasChanged`" is set to `true` if the value of the signal has changed since the preceding call. • `out bool out_SignalsHaveChanged` Returns `true` if the value of at least one signal has changed since the preceding call. • `bool validateAddressTag` Optional boolean value to enable or disable the Memory Address to Tag association check.	
Signal error	Runtime error code	Description
	`ERuntimeErrorCode.OK`	The signal operation is successful.
	`ERuntimeErrorCode.IndexOutOfRange`	Offset or bits are invalid.
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	

Exceptions	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.
	`ERuntimeErrorCode.SignalConfigura-tionError`	At least one signal error is in the list.

8.6.4 I/O access via address - Writing

WriteBit()

Writes an individual bit to the area.

NOTE
Data can be overwritten
The function allows access to the entire storage area of the simulated PLC instance. Therefore, use access via the tag name (Page 139) and not via the address areas.

Table 8-78 WriteBit() - Native C++

Syntax	`ERuntimeErrorCode WriteBit(EArea in_Area, UINT32 in_Offset, UINT8 in_Bit, bool in_Value);`	
Parameters	• `EArea in_Area:` The area that is to be written. Permissible values: `{SRA_INPUT, SRA_MARKER, SRA_OUTPUT}`. See EArea (Page 259). • `UINT32 in_Offset:` The byte offset within the area. The value must be between 0 and the value that `GetAreaSize()` returns. • `UINT8 in_Bit:` The bit offset within the byte. The value must be between 0 and 7. • `bool in_Value:` Bit value.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.
	`SREC_INDEX_OUT_OF_RANGE`	Offset or bits are invalid.
	`SREC_WRONG_ARGUMENT`	Area is invalid.

Table 8-79 WriteBit() - .NET (C#)

Syntax	`void InputArea WriteBit(UInt32 in_Offset, Byte in_Bit, bool in_Value);`
	`void MarkerArea WriteBit(UInt32 in_Offset, Byte in_Bit, bool in_Value);`
	`void OutputArea WriteBit(UInt32 in_Offset, Byte in_Bit, bool in_Value);`	

Parameters	• `UInt32 in_Offset:` The byte offset within the area. The value must be between 0 and the value that `AreaSize` returns. • `Byte in_Bit:` The bit offset within the byte. The value must be between 0 and 7. • `bool in_Value:` Bit value.	
Return values	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.
	`ERuntimeErrorCode.IndexOutOfRange`	Offset or bits are invalid.

WriteByte()

Writes an individual byte to the area.

NOTE

Data can be overwritten

The function allows access to the entire storage area of the simulated PLC instance. Therefore, use access via the tag name (Page 139) and not via the address areas.

Table 8-80 WriteByte() - Native C++

Syntax	`ERuntimeErrorCode WriteByte(EArea in_Area, UINT32 in_Offset, BYTE in_Value);`	
Parameters	• `EArea in_Area:` The area that is to be written. Permissible values: `{SRA_INPUT, SRA_MARKER, SRA_OUTPUT}`. See EArea (Page 259). • `UINT32 in_Offset:` The byte offset within the area. The value must be between 0 and the value that `GetAreaSize()` returns. • `BYTE in_Value:` Byte value.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.

Return values	SREC_INSTANCE_NOT_RUNNING	The process of the simulated PLC instance is not running.
	SREC_INDEX_OUT_OF_RANGE	Offset is invalid.
	SREC_WRONG_ARGUMENT	Area is invalid.

Table 8-81 WriteByte() - .NET (C#)

Syntax	void InputArea.WriteByte(UInt32 in_Offset, Byte in_Value);
	void MarkerArea.WriteByte(UInt32 in_Offset, Byte in_Value);
	void OutputArea.WriteByte(UInt32 in_Offset, Byte in_Value);	

Parameters	• UINT32 in_Offset: The byte offset within the area. The value must be between 0 and the value that AreaSize returns. • BYTE in_Value: Byte value.	
Return values	None	
Exceptions	Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException	
	Runtime error code	Description
	ERuntimeErrorCode.InterfaceRemoved	The instance is not registered in Runtime Manager.
	ERuntimeErrorCode.Timeout	The function does not return on time.
	ERuntimeErrorCode.InstanceNotRunning	The process of the simulated PLC instance is not running.
	ERuntimeErrorCode.IndexOutOfRange	Offset is invalid.

WriteBytes()

Writes a byte array to the area.

NOTE

Data can be overwritten

The function allows access to the entire storage area of the simulated PLC instance.

In particular, do not write to bytes that belong to other applications or contain internal data, for example, qualifier bits for fail-safe I/O modules.

Therefore, use access via the tag name (Page 139) and not via the address areas.

Table 8-82 WriteBytes() - Native C++

Syntax	ERuntimeErrorCode WriteBytes(EArea in_Area, UINT32 in_Offset, UINT32 in_BytesToWrite, UINT32* out_BytesWritten, BYTE in_Values[]);
Parameters	• EArea in_Area: The area that is to be written. Permissible values: {SRA_INPUT, SRA_MARKER, SRA_OUTPUT}. See EArea (Page 259). • UINT32 in_Offset: The byte offset within the area. The value must be between 0 and the value that GetAreaSize() returns. • UINT32 in_BytesToWrite: Contains the size of the array value to be written.

	• `UINT32* out_BytesWritten:` Contains the number of bytes that were just written. • `BYTE in_Values[]:` Byte array that is to be written to the area.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.
	`SREC_INDEX_OUT_OF_RANGE`	The offset lies outside the area range. No byte could be written.
	`SREC_WRONG_ARGUMENT`	The area is invalid.

Table 8-83 WriteBytes() - .NET (C#)

Syntax	`UInt32 InputArea.WriteBytes(UInt32 in_Offset, Byte[] in_Values);`
	`UInt32 InputArea.WriteBytes(UInt32 in_Offset, UInt32 in_BytesToWrite, Byte[] in_Values);`
	`UInt32 MarkerArea.WriteBytes(UInt32 in_Offset, Byte[] in_Values);`
	`UInt32 MarkerArea.WriteBytes(UInt32 in_Offset, UInt32 in_BytesToWrite, Byte[] in_Values);`
	`UInt32 OutputArea.WriteBytes(UInt32 in_Offset, Byte[] in_Values);`
	`UInt32 OutputArea.WriteBytes(UInt32 in_Offset, UInt32 in_BytesToWrite, Byte[] in_Values);`	

Parameters	• `UINT32 in_Offset:` The byte offset within the area. The value must be between 0 and the value that `AreaSize` returns. • `UInt32 in_BytesToWrite:` Contains the number of bytes to be written. The value must be between 1 and the size of the array value. • `BYTE in_Value:` Byte value.	
Return values	`Uint32:` Contains the number of bytes that were just written.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.
	`ERuntimeErrorCode.IndexOutOfRange`	The offset lies outside the area range. No byte could be written.

WriteSignals()

Writes multiple signals within an API call.

The function also takes into consideration the byte order (Endianness).

The function supports only primitive data type signals, but it is not typical.

NOTE

Data can be overwritten

The function allows access to the entire storage area of the simulated PLC instance. Therefore, use access via the tag name (Page 139) and not via the address areas.

Table 8-84 WriteSignals() - Native C++

Syntax	`ERuntimeErrorCode WriteSignals(EArea in_Area, SDataValueByAddress* in_Signals, UINT32 in_SignalCount, bool validateAddressTag = false);`	
Parameters	• `EArea in_Area` The area that is to be written. Permissible values: `{SRA_INPUT, SRA_MARKER, SRA_OUTPUT}`. See EArea (Page 259). • `SDataValueByAddress* inout_Signals` The signal list to be written. • `UINT32 in_SignalCount` Number of signals in the list. • `bool validateAddressTag` Boolean value to enable or disable the Memory Address to Tag association check.	
Signal error	Error code	Description
	`SREC_OK`	The signal operation is successful.
	`SREC_INDEX_OUT_OF_RANGE`	Offset or bits are invalid.
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.
	`SREC_SIGNAL_CONFIGURATION_ERROR`	At least one signal error is in the list.
	`SREC_WRONG_ARGUMENT`	The area is invalid.

Table 8-85 WriteSignals() - .NET (C#)

Syntax	`void WriteSignals( SDataValueByAddress[ ] in_Signals, bool validateAddressTag );`	
Parameters	• `SDataValueByAddress[] in_Signals` The signal list to be written. • `bool validateAddressTag` Optional boolean value to enable or disable the Memory Address to Tag association check.	
Signal error	Error code	Description
	`ERuntimeErrorCode.OK`	The signal operation is successful.
	`ERuntimeError-Code.IndexOutOfRange`	Offset or bits are invalid.
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.

Exceptions	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.
	`ERuntimeErrorCode.SignalConfigura-tionError`	At least one signal error is in the list.

8.6.5 I/O access via tag name - Reading

Individual access to I/O data is used for displaying and writing values that are not refreshed regularly in a graphical user interface (GUI).

NOTE

To simulate a regular exchange of signals, create a signal list for each set of signals. Use this signal list for all further accesses. Create a new list as soon as the set of signals changes.

For the signal lists use the functions `ReadSignals()` and `WriteSignals()`.

Read()

Reads the value of a PLC tag.

Table 8-86 Read() - Native C++

Syntax	`ERuntimeErrorCode Read(WCHAR* in_Tag, SDataValue* inout_Value);`	
Parameters	• `WCHAR* in_Tag`: The name of the PLC tag that is to be read. • `SDataValue* inout_Value`: Contains the value and the expected type of the PLC tag. If the expected type is `UNSPECIFIC`, it is set to the stored type when the function was successful. The `STRUCT` type is not supported. Structures and fields can be emulated through signal lists and be read by using the `ReadSignals()` function.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.
	`SREC_INDEX_OUT_OF_RANGE`	The offset lies outside the area range. No value could be read.
	`SREC_DOES_NOT_EXIST`	The entry does not exist in the stored tag list.
	`SREC_NOT_SUPPORTED`	Access to entire structures or arrays is not suppor-ted.
	`SREC_TYPE_MISMATCH`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.
	`SREC_NOT_ENOUGH_MEMORY`	Unexpected error occurred; turn on logging and contact support.
	`SREC_INTERNAL_ERROR`	Internal error occurred; turn on logging and contact support.

Table 8-87 Read() - .NET (C#)

Syntax	`SDataValue Read(string in_Tag)`	
Parameters	• `string in_Tag:` The name of the PLC tag that is to be read.	
Return values	`SDataValue:` Contains the value and the type of the PLC tag.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.
	`ERuntimeErrorCode.IndexOutOfRange`	The offset lies outside the area range. No value could be read.
	`ERuntimeErrorCode.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 266).
	`ERuntimeErrorCode.NotUpToDate`	The stored tag list must be updated.
	`ERuntimeErrorCode.NotEnoughMemory`	Unexpected error occurred; turn on logging and contact support.
	`ERuntimeErrorCode.InternalError`	Internal error occurred; turn on logging and contact support.

ReadBool()

Reads the value of a PLC tag.

Table 8-88 ReadBool() - Native C++

Syntax	`ERuntimeErrorCode ReadBool(WCHAR* in_Tag, bool* out_Value);`	
Parameters	• `WCHAR* in_Tag:` The name of the PLC tag that is to be read. • `bool* out_Value:` Contains the value of the PLC tag.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.
	`SREC_INDEX_OUT_OF_RANGE`	The offset lies outside the area range. No value could be read.
	`SREC_DOES_NOT_EXIST`	The entry does not exist in the stored tag list.
	`SREC_NOT_SUPPORTED`	Access to entire structures or arrays is not supported.

Return values	SREC_TYPE_MISMATCH	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	SREC_NOT_UP_TO_DATE	The stored tag list must be updated.
	SREC_NOT_ENOUGH_MEMORY	Unexpected error occurred; turn on logging and contact support.
	SREC_INTERNAL_ERROR	Internal error occurred; turn on logging and contact support.

Table 8-89 ReadBool() - .NET (C#)

Syntax	bool ReadBool(string in_Tag)	
Parameters	• string in_Tag: The name of the PLC tag that is to be read.	
Return values	bool: Contains the value of the PLC tag.	
Exceptions	Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException	
	Runtime error code	Description
	ERuntimeErrorCode.InterfaceRemoved	The instance is not registered in Runtime Manager.
	ERuntimeErrorCode.Timeout	The function does not return on time.
	ERuntimeErrorCode.InstanceNotRunning	The process of the simulated PLC instance is not running.
	ERuntimeErrorCode.IndexOutOfRange	The offset lies outside the area range. No value could be read.
	ERuntimeErrorCode.DoesNotExist	The entry does not exist in the stored tag list.
	ERuntimeErrorCode.NotSupported	Access to entire structures or arrays is not supported.
	ERuntimeErrorCode.TypeMismatch	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	ERuntimeErrorCode.NotUpToDate	The stored tag list must be updated.
	ERuntimeErrorCode.NotEnoughMemory	Unexpected error occurred; turn on logging and contact support.
	ERuntimeErrorCode.InternalError	Internal error occurred; turn on logging and contact support.

ReadInt8()

Reads the value of a PLC tag.

Table 8-90 ReadInt8() - Native C++

Syntax	ERuntimeErrorCode ReadInt8(WCHAR* in_Tag, INT8* out_Value);	
Parameters	• WCHAR* in_Tag: The name of the PLC tag that is to be read. • INT8* out_Value: Contains the value of the PLC tag.	
Return values	Runtime error code	Description
	SREC_OK	The function is successful.
	SREC_INTERFACE_REMOVED	The instance is not registered in Runtime Manager.
	SREC_TIMEOUT	The function does not return on time.

Return values	SREC_INSTANCE_NOT_RUNNING	The process of the simulated PLC instance is not running.
	SREC_INDEX_OUT_OF_RANGE	The offset lies outside the area range. No value could be read.
	SREC_DOES_NOT_EXIST	The entry does not exist in the stored tag list.
	SREC_NOT_SUPPORTED	Access to entire structures or arrays is not supported.
	SREC_TYPE_MISMATCH	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	SREC_NOT_UP_TO_DATE	The stored tag list must be updated.
	SREC_NOT_ENOUGH_MEMORY	Unexpected error occurred; turn on logging and contact support.
	SREC_INTERNAL_ERROR	Internal error occurred; turn on logging and contact support.

Table 8-91 ReadInt8() - .NET (C#)

Syntax	`Int8 ReadInt8(string in_Tag)`	
Parameters	• `string in_Tag:` The name of the PLC tag that is to be read.	
Return values	`Int8:` Contains the value and the type of the PLC tag.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.
	`ERuntimeErrorCode.IndexOutOfRange`	The offset lies outside the area range. No value could be read.
	`ERuntimeErrorCode.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	`ERuntimeErrorCode.NotUpToDate`	The stored tag list must be updated.
	`ERuntimeErrorCode.NotEnoughMemory`	Unexpected error occurred; turn on logging and contact support.
	`ERuntimeErrorCode.InternalError`	Internal error occurred; turn on logging and contact support.

ReadInt16()

Reads the value of a PLC tag.

Table 8-92 ReadInt16() - Native C++

Syntax	`ERuntimeErrorCode ReadInt16(WCHAR* in_Tag, INT16* out_Value);`	
Parameters	• `WCHAR* in_Tag:` The name of the PLC tag that is to be read. • `INT16* out_Value:` Contains the value of the PLC tag.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.
	`SREC_INDEX_OUT_OF_RANGE`	The offset lies outside the area range. No value could be read.
	`SREC_DOES_NOT_EXIST`	The entry does not exist in the stored tag list.
	`SREC_NOT_SUPPORTED`	Access to entire structures or arrays is not supported.
	`SREC_TYPE_MISMATCH`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.
	`SREC_NOT_ENOUGH_MEMORY`	Unexpected error occurred; turn on logging and contact support.
	`SREC_INTERNAL_ERROR`	Internal error occurred; turn on logging and contact support.

Table 8-93 ReadInt16() - .NET (C#)

Syntax	`Int16 ReadInt16(string in_Tag)`	
Parameters	• `string in_Tag:` The name of the PLC tag that is to be read.	
Return values	`Int16:` Contains the value and the type of the PLC tag.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.
	`ERuntimeErrorCode.IndexOutOfRange`	The offset lies outside the area range. No value could be read.
	`ERuntimeErrorCode.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).

Exceptions	`ERuntimeErrorCode.NotUpToDate`	The stored tag list must be updated.
	`ERuntimeErrorCode.NotEnoughMemory`	Unexpected error occurred; turn on logging and contact support.
	`ERuntimeErrorCode.InternalError`	Internal error occurred; turn on logging and contact support.

ReadInt32()

Reads the value of a PLC tag.

Table 8-94 ReadInt32() - Native C++

Syntax	`ERuntimeErrorCode ReadInt32(WCHAR* in_Tag, INT32* out_Value);`	
Parameters	• `WCHAR* in_Tag:` The name of the PLC tag that is to be read. • `INT32* out_Value:` Contains the value of the PLC tag.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.
	`SREC_INDEX_OUT_OF_RANGE`	The offset lies outside the area range. No value could be read.
	`SREC_DOES_NOT_EXIST`	The entry does not exist in the stored tag list.
	`SREC_NOT_SUPPORTED`	Access to entire structures or arrays is not supported.
	`SREC_TYPE_MISMATCH`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.
	`SREC_NOT_ENOUGH_MEMORY`	Unexpected error occurred; turn on logging and contact support.
	`SREC_INTERNAL_ERROR`	Internal error occurred; turn on logging and contact support.

Table 8-95 ReadInt32() - .NET (C#)

Syntax	`Int32 ReadInt32(string in_Tag)`	
Parameters	• `string in_Tag:` The name of the PLC tag that is to be read.	
Return values	`Int32:` Contains the value and the type of the PLC tag.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.

Exceptions	`ERuntimeErrorCode.IndexOutOfRange`	The offset lies outside the area range. No value could be read.
	`ERuntimeErrorCode.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	`ERuntimeErrorCode.NotUpToDate`	The stored tag list must be updated.
	`ERuntimeErrorCode.NotEnoughMemory`	Unexpected error occurred; turn on logging and contact support.
	`ERuntimeErrorCode.InternalError`	Internal error occurred; turn on logging and contact support.

ReadInt64()

Reads the value of a PLC tag.

Table 8-96 ReadInt64() - Native C++

Syntax	`ERuntimeErrorCode ReadInt64(WCHAR* in_Tag, INT64* out_Value);`	
Parameters	• `WCHAR* in_Tag:` The name of the PLC tag that is to be read. • `INT64* out_Value:` Contains the value of the PLC tag.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.
	`SREC_INDEX_OUT_OF_RANGE`	The offset lies outside the area range. No value could be read.
	`SREC_DOES_NOT_EXIST`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.
	`SREC_NOT_ENOUGH_MEMORY`	Unexpected error occurred; turn on logging and contact support.
	`SREC_INTERNAL_ERROR`	Internal error occurred; turn on logging and contact support.

Table 8-97 ReadInt64() - .NET (C#)

Syntax	`Int64 ReadInt64(string in_Tag)`	
Parameters	• `string in_Tag:` The name of the PLC tag that is to be read.	
Return values	`Int64:` Contains the value and the type of the PLC tag.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.
	`ERuntimeErrorCode.IndexOutOfRange`	The offset lies outside the area range. No value could be read.
	`ERuntimeErrorCode.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	`ERuntimeErrorCode.NotUpToDate`	The stored tag list must be updated.
	`ERuntimeErrorCode.NotEnoughMemory`	Unexpected error occurred; turn on logging and contact support.
	`ERuntimeErrorCode.InternalError`	Internal error occurred; turn on logging and contact support.

ReadUInt8()

Reads the value of a PLC tag.

Table 8-98 ReadUInt8() - Native C++

Syntax	`ERuntimeErrorCode ReadUInt8(WCHAR* in_Tag, UINT8* out_Value);`	
Parameters	• `WCHAR* in_Tag:` The name of the PLC tag that is to be read. • `UINT8* out_Value:` Contains the value of the PLC tag.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.
	`SREC_INDEX_OUT_OF_RANGE`	The offset lies outside the area range. No value could be read.
	`SREC_DOES_NOT_EXIST`	The entry does not exist in the stored tag list.
	`SREC_NOT_SUPPORTED`	Access to entire structures or arrays is not supported.

Return values	SREC_TYPE_MISMATCH	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	SREC_NOT_UP_TO_DATE	The stored tag list must be updated.
	SREC_NOT_ENOUGH_MEMORY	Unexpected error occurred; turn on logging and contact support.
	SREC_INTERNAL_ERROR	Internal error occurred; turn on logging and contact support.

Table 8-99 ReadUInt8() - .NET (C#)

Syntax	UInt8 ReadUInt8(string in_Tag)	
Parameters	• string in_Tag: The name of the PLC tag that is to be read.	
Return values	UInt8: Contains the value and the type of the PLC tag.	
Exceptions	Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException	
	Runtime error code	Description
	ERuntimeErrorCode.InterfaceRemoved	The instance is not registered in Runtime Manager.
	ERuntimeErrorCode.Timeout	The function does not return on time.
	ERuntimeErrorCode.InstanceNotRunning	The process of the simulated PLC instance is not running.
	ERuntimeErrorCode.IndexOutOfRange	The offset lies outside the area range. No value could be read.
	ERuntimeErrorCode.DoesNotExist	The entry does not exist in the stored tag list.
	ERuntimeErrorCode.NotSupported	Access to entire structures or arrays is not supported.
	ERuntimeErrorCode.TypeMismatch	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	ERuntimeErrorCode.NotUpToDate	The stored tag list must be updated.
	ERuntimeErrorCode.NotEnoughMemory	Unexpected error occurred; turn on logging and contact support.
	ERuntimeErrorCode.InternalError	Internal error occurred; turn on logging and contact support.

ReadUInt16()

Reads the value of a PLC tag.

Table 8-100 ReadUInt16() - Native C++

Syntax	ERuntimeErrorCode ReadUInt16(WCHAR* in_Tag, UINT16* out_Value);	
Parameters	• WCHAR* in_Tag: The name of the PLC tag that is to be read. • UINT16* out_Value: Contains the value of the PLC tag.	
Return values	Runtime error code	Description
	SREC_OK	The function is successful.
	SREC_INTERFACE_REMOVED	The instance is not registered in Runtime Manager.
	SREC_TIMEOUT	The function does not return on time.

Return values	SREC_INSTANCE_NOT_RUNNING	The process of the simulated PLC instance is not running.
	SREC_INDEX_OUT_OF_RANGE	The offset lies outside the area range. No value could be read.
	SREC_DOES_NOT_EXIST	The entry does not exist in the stored tag list.
	SREC_NOT_SUPPORTED	Access to entire structures or arrays is not supported.
	SREC_TYPE_MISMATCH	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	SREC_NOT_UP_TO_DATE	The stored tag list must be updated.
	SREC_NOT_ENOUGH_MEMORY	Unexpected error occurred; turn on logging and contact support.
	SREC_INTERNAL_ERROR	Internal error occurred; turn on logging and contact support.

Table 8-101 ReadUInt16() - .NET (C#)

Syntax	UInt16 ReadUInt16(string in_Tag)	
Parameters	• string in_Tag: The name of the PLC tag that is to be read.	
Return values	UInt16: Contains the value and the type of the PLC tag.	
Exceptions	Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException	
	Runtime error code	Description
	ERuntimeErrorCode.InterfaceRemoved	The instance is not registered in Runtime Manager.
	ERuntimeErrorCode.Timeout	The function does not return on time.
	ERuntimeErrorCode.InstanceNotRunning	The process of the simulated PLC instance is not running.
	ERuntimeErrorCode.IndexOutOfRange	The offset lies outside the area range. No value could be read.
	ERuntimeErrorCode.DoesNotExist	The entry does not exist in the stored tag list.
	ERuntimeErrorCode.NotSupported	Access to entire structures or arrays is not supported.
	ERuntimeErrorCode.TypeMismatch	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	ERuntimeErrorCode.NotUpToDate	The stored tag list must be updated.
	ERuntimeErrorCode.NotEnoughMemory	Unexpected error occurred; turn on logging and contact support.
	ERuntimeErrorCode.InternalError	Internal error occurred; turn on logging and contact support.

ReadUInt32()

Reads the value of a PLC tag.

Table 8-102 ReadUInt32() - Native C++

Syntax	`ERuntimeErrorCode ReadUInt32(WCHAR* in_Tag, UINT32* out_Value);`	
Parameters	• `WCHAR* in_Tag:` The name of the PLC tag that is to be read. • `UINT32* out_Value:` Contains the value of the PLC tag.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.
	`SREC_INDEX_OUT_OF_RANGE`	The offset lies outside the area range. No value could be read.
	`SREC_DOES_NOT_EXIST`	The entry does not exist in the stored tag list.
	`SREC_NOT_SUPPORTED`	Access to entire structures or arrays is not supported.
	`SREC_TYPE_MISMATCH`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.
	`SREC_NOT_ENOUGH_MEMORY`	Unexpected error occurred; turn on logging and contact support.
	`SREC_INTERNAL_ERROR`	Internal error occurred; turn on logging and contact support.

Table 8-103 ReadUInt32() - .NET (C#)

Syntax	`UInt32 ReadUInt32(string in_Tag)`	
Parameters	• `string in_Tag:` The name of the PLC tag that is to be read.	
Return values	`Uint32:` Contains the value and the type of the PLC tag.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.
	`ERuntimeErrorCode.IndexOutOfRange`	The offset lies outside the area range. No value could be read.
	`ERuntimeErrorCode.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).

Exceptions	`ERuntimeErrorCode.NotUpToDate`	The stored tag list must be updated.
	`ERuntimeErrorCode.NotEnoughMemory`	Unexpected error occurred; turn on logging and contact support.
	`ERuntimeErrorCode.InternalError`	Internal error occurred; turn on logging and contact support.

ReadUInt64()

Reads the value of a PLC tag.

Table 8-104 ReadInt64() - Native C++

Syntax	`ERuntimeErrorCode ReadUInt64(WCHAR* in_Tag, UINT64* out_Value);`	
Parameters	• `WCHAR* in_Tag:` The name of the PLC tag that is to be read. • `UINT64* out_Value:` Contains the value of the PLC tag.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.
	`SREC_INDEX_OUT_OF_RANGE`	The offset lies outside the area range. No value could be read.
	`SREC_DOES_NOT_EXIST`	The entry does not exist in the stored tag list.
	`SREC_NOT_SUPPORTED`	Access to entire structures or arrays is not supported.
	`SREC_TYPE_MISMATCH`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.
	`SREC_NOT_ENOUGH_MEMORY`	Unexpected error occurred; turn on logging and contact support.
	`SREC_INTERNAL_ERROR`	Internal error occurred; turn on logging and contact support.

Table 8-105 ReadUInt64() - .NET (C#)

Syntax	`UInt64 ReadUInt64(string in_Tag)`	
Parameters	• `string in_Tag:` The name of the PLC tag that is to be read.	
Return values	`UInt64:` Contains the value and the type of the PLC tag.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.

Exceptions	`ERuntimeErrorCode.IndexOutOfRange`	The offset lies outside the area range. No value could be read.
	`ERuntimeErrorCode.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	`ERuntimeErrorCode.NotUpToDate`	The stored tag list must be updated.
	`ERuntimeErrorCode.NotEnoughMemory`	Unexpected error occurred; turn on logging and contact support.
	`ERuntimeErrorCode.InternalError`	Internal error occurred; turn on logging and contact support.

ReadFloat()

Reads the value of a PLC tag.

Table 8-106 ReadFloat() - Native C++

Syntax	`ERuntimeErrorCode ReadFloat(WCHAR* in_Tag, float* out_Value);`	
Parameters	• `WCHAR* in_Tag:` The name of the PLC tag that is to be read. • `float* out_Value:` Contains the value of the PLC tag.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.
	`ERuntimeErrorCode.IndexOutOfRange`	The offset lies outside the area range. No value could be read.
	`SREC_DOES_NOT_EXIST`	The entry does not exist in the stored tag list.
	`SREC_NOT_SUPPORTED`	Access to entire structures or arrays is not supported.
	`SREC_TYPE_MISMATCH`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.
	`SREC_NOT_ENOUGH_MEMORY`	Unexpected error occurred; turn on logging and contact support.
	`SREC_INTERNAL_ERROR`	Internal error occurred; turn on logging and contact support.

Table 8-107 ReadFloat() - .NET (C#)

Syntax	`float ReadFloat(string in_Tag)`	
Parameters	• `string in_Tag:` The name of the PLC tag that is to be read.	
Return values	`float:` Contains the value and the type of the PLC tag.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.
	`ERuntimeErrorCode.IndexOutOfRange`	The offset lies outside the area range. No value could be read.
	`ERuntimeErrorCode.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	`ERuntimeErrorCode.NotUpToDate`	The stored tag list must be updated.
	`ERuntimeErrorCode.NotEnoughMemory`	Unexpected error occurred; turn on logging and contact support.
	`ERuntimeErrorCode.InternalError`	Internal error occurred; turn on logging and contact support.

ReadDouble()

Reads the value of a PLC tag.

Table 8-108 ReadDouble() - Native C++

Syntax	`ERuntimeErrorCode ReadDouble(WCHAR* in_Tag, double* out_Value);`	
Parameters	• `WCHAR* in_Tag:` The name of the PLC tag that is to be read. • `double* out_Value:` Contains the value of the PLC tag.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.
	`ERuntimeErrorCode.IndexOutOfRange`	The offset lies outside the area range. No value could be read.
	`SREC_DOES_NOT_EXIST`	The entry does not exist in the stored tag list.
	`SREC_NOT_SUPPORTED`	Access to entire structures or arrays is not supported.

Return values	SREC_TYPE_MISMATCH	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	SREC_NOT_UP_TO_DATE	The stored tag list must be updated.
	SREC_NOT_ENOUGH_MEMORY	Unexpected error occurred; turn on logging and contact support.
	SREC_INTERNAL_ERROR	Internal error occurred; turn on logging and contact support.

Table 8-109 ReadDouble() - .NET (C#)

Syntax	double ReadDouble(string in_Tag)	
Parameters	• string in_Tag: The name of the PLC tag that is to be read.	
Return values	double: Contains the value and the type of the PLC tag.	
Exceptions	Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException	
	Runtime error code	Description
	ERuntimeErrorCode.InterfaceRemoved	The instance is not registered in Runtime Manager.
	ERuntimeErrorCode.Timeout	The function does not return on time.
	ERuntimeErrorCode.InstanceNotRunning	The process of the simulated PLC instance is not running.
	ERuntimeErrorCode.IndexOutOfRange	The offset lies outside the area range. No value could be read.
	ERuntimeErrorCode.DoesNotExist	The entry does not exist in the stored tag list.
	ERuntimeErrorCode.NotSupported	Access to entire structures or arrays is not supported.
	ERuntimeErrorCode.TypeMismatch	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	ERuntimeErrorCode.NotUpToDate	The stored tag list must be updated.
	ERuntimeErrorCode.NotEnoughMemory	Unexpected error occurred; turn on logging and contact support.
	ERuntimeErrorCode.InternalError	Internal error occurred; turn on logging and contact support.

ReadChar()

Reads the value of a PLC tag.

Table 8-110 ReadChar() - Native C++

Syntax	ERuntimeErrorCode ReadChar(WCHAR* in_Tag, char* out_Value);	
Parameters	• WCHAR* in_Tag: The name of the PLC tag that is to be read. • char* out_Value: Contains the value of the PLC tag.	
Return values	Runtime error code	Description
	SREC_OK	The function is successful.
	SREC_INTERFACE_REMOVED	The instance is not registered in Runtime Manager.
	SREC_TIMEOUT	The function does not return on time.

Return values	SREC_INSTANCE_NOT_RUNNING	The process of the simulated PLC instance is not running.
	ERuntimeErrorCode.IndexOutOfRange	The offset lies outside the area range. No value could be read.
	SREC_DOES_NOT_EXIST	The entry does not exist in the stored tag list.
	SREC_NOT_SUPPORTED	Access to entire structures or arrays is not supported.
	SREC_TYPE_MISMATCH	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	SREC_NOT_UP_TO_DATE	The stored tag list must be updated.
	SREC_NOT_ENOUGH_MEMORY	Unexpected error occurred; turn on logging and contact support.
	SREC_INTERNAL_ERROR	Internal error occurred; turn on logging and contact support.

Table 8-111 ReadChar() - .NET (C#)

Syntax	sbyte ReadChar(string in_Tag)	
Parameters	• string in_Tag: The name of the PLC tag that is to be read.	
Return values	sbyte: Contains the value and the type of the PLC tag.	
Exceptions	Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException	
	Runtime error code	Description
	ERuntimeErrorCode.InterfaceRemoved	The instance is not registered in Runtime Manager.
	ERuntimeErrorCode.Timeout	The function does not return on time.
	ERuntimeErrorCode.InstanceNotRunning	The process of the simulated PLC instance is not running.
	ERuntimeErrorCode.IndexOutOfRange	The offset lies outside the area range. No value could be read.
	ERuntimeErrorCode.DoesNotExist	The entry does not exist in the stored tag list.
	ERuntimeErrorCode.NotSupported	Access to entire structures or arrays is not supported.
	ERuntimeErrorCode.TypeMismatch	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	ERuntimeErrorCode.NotUpToDate	The stored tag list must be updated.
	ERuntimeErrorCode.NotEnoughMemory	Unexpected error occurred; turn on logging and contact support.
	ERuntimeErrorCode.InternalError	Internal error occurred; turn on logging and contact support.

ReadWChar()

Reads the value of a PLC tag.

Table 8-112 ReadWChar() - Native C++

Syntax	`ERuntimeErrorCode ReadWChar(WCHAR* in_Tag, WCHAR* out_Value);`	
Parameters	• `WCHAR* in_Tag:` The name of the PLC tag that is to be read. • `WCHAR* out_Value:` Contains the value of the PLC tag.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.
	`ERuntimeErrorCode.IndexOutOfRange`	The offset lies outside the area range. No value could be read.
	`SREC_DOES_NOT_EXIST`	The entry does not exist in the stored tag list.
	`SREC_NOT_SUPPORTED`	Access to entire structures or arrays is not supported.
	`SREC_TYPE_MISMATCH`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.
	`SREC_NOT_ENOUGH_MEMORY`	Unexpected error occurred; turn on logging and contact support.
	`SREC_INTERNAL_ERROR`	Internal error occurred; turn on logging and contact support.

Table 8-113 ReadWChar() - .NET (C#)

Syntax	`char ReadWChar(string in_Tag)`	
Parameters	• `string in_Tag:` The name of the PLC tag that is to be read.	
Return values	`char:` Contains the value and the type of the PLC tag.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.
	`ERuntimeErrorCode.IndexOutOfRange`	The offset lies outside the area range. No value could be read.
	`ERuntimeErrorCode.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).

Exceptions	`ERuntimeErrorCode.NotUpToDate`	The stored tag list must be updated.
	`ERuntimeErrorCode.NotEnoughMemory`	Unexpected error occurred; turn on logging and contact support.
	`ERuntimeErrorCode.InternalError`	Internal error occurred; turn on logging and contact support.

ReadSignals()

Reads multiple signals within an API call.

When the function is called for the first time, it stores internal information in the structures `SDataValueByName*` to improve the performance of the subsequent calls.

NOTE

To simulate a regular exchange of signals, create a signal list for each set of signals. Use this signal list for all further accesses. Create a new list as soon as the set of signals changes.

Table 8-114 ReadSignals() - Native C++

Syntax	`ERuntimeErrorCode ReadSignals(SDataValueByName* inout_Signals, UINT32 in_SignalCount);` `ERuntimeErrorCode ReadSignals(SDataValueByNameWithCheck* inout_Signals, UINT32 in_SignalCount bool* out_SignalsHaveChanged);`	
Parameters	• `SDataValueByName* inout_Signals:` Contains the name, the value and the expected type of the PLC tag. If the expected type is `UNSPECIFIC`, it is set to the stored type when the function was successful. The `STRUCT` type is not supported. • `SDataValueByNameWithCheck* inout_Signals:` Contains the name, the value and the expected type of the PLC tag. If the expected type is `UNSPECIFIC`, it is set to the stored type when the function was successful. The `STRUCT` type is not supported. "`ValueHasChanged`" is set to `true` if the value of the signal has changed since the preceding call. • `UINT32 in_SignalCount:` The number of signals to be read. • `bool* out_SignalsHaveChanged:` Returns `true` if the value of at least one signal has changed since the preceding call.	
Signal error	Runtime error code	Description
	`SREC_OK`	The signal operation is successful.
	`SREC_DOES_NOT_EXIST`	The entry does not exist in the stored tag list.
	`SREC_NOT_SUPPORTED`	Access to entire structures or arrays is not supported.
	`SREC_TYPE_MISMATCH`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	`SREC_INDEX_OUT_OF_RANGE`	Offset or bits are invalid.
Return values	Runtime error code	Condition
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.

Return values	SREC_INSTANCE_NOT_RUNNING	The process of the simulated PLC instance is not running.
	SREC_NOT_UP_TO_DATE	The stored tag list must be updated.
	SREC_SIGNAL_CONFIGURATION_ERROR	At least one signal error is in the list.

Table 8-115 ReadSignals() - .NET (C#)

Syntax	`void ReadSignals(ref SDataValueByName[] inout_Signals)` `void ReadSignals(ref SDataValueByNameWithCheck[] inout_Signals out bool out_SignalsHaveChanged);`	
Parameters	• `ref SDataValueByName[] inout_Signals:` Contains the name, the value and the expected type of the PLC tag. If the expected type is `UNSPECIFIC`, it is set to the stored type when the function was successful. The `STRUCT` type is not supported. • `ref SDataValueByNameWithCheck[] inout_Signals:` Contains the name, the value and the expected type of the PLC tag. If the expected type is `UNSPECIFIC`, it is set to the stored type when the function was successful. The `STRUCT` type is not supported. "`ValueHasChanged`" is set to `true` if the value of the signal has changed since the preceding call. • `out bool out_SignalsHaveChanged:` Returns `true` if the value of at least one signal has changed since the preceding call.	
Return values	`SDataValue`: Contains the value and the type of the PLC tag.	
Signal error	Runtime error code	Description
	`ERuntimeErrorCode.OK`	The signal operation is successful.
	`ERuntimeErrorCode.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	`ERuntimeErrorCode.IndexOutOfRange`	Offset or bits are invalid.
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.
	`ERuntimeErrorCode.NotUpToDate`	The stored tag list must be updated.

ReadString()

Reads the value of a PLC tag.

Table 8-116 ReadString() - Native C++

Syntax	`ERuntimeErrorCode IInstance::ReadString(`
	`const WCHAR* in_Tag,`
	`CHAR* out_Value,`
	`const UINT32 in_BufferSize,`
	`UINT32* out_CharsRead`
	`);`	

Parameters	• `const WCHAR* in_Tag:` The name of the PLC tag that is to be read. • `CHAR* out_Value:` Contains the value of the PLC tag • `const UINT32 in_BufferSize:` Size of the buffer provided • `UINT32* out_CharsRead:` Number of characters read	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.
	`SREC_INDEX_OUT_OF_RANGE`	The offset lies outside the area range. No value could be read.
	`SREC_DOES_NOT_EXIST`	The entry does not exist in the stored tag list.
	`SREC_NOT_SUPPORTED`	Access to entire structures or arrays is not supported.
	`SREC_TYPE_MISMATCH`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.
	`SREC_BUFFER_OVERFLOW`	The buffer provided is too small.
	`SREC_NOT_ENOUGH_MEMORY`	Unexpected error occurred; turn on logging and contact support.
	`SREC_INTERNAL_ERROR`	Internal error occurred; turn on logging and contact support.

Table 8-117 ReadString() - .NET (C#)

Syntax	`string ReadString(`
	`string in_Tag`
	`);`	

Parameters	• `string in_Tag:` The name of the PLC tag that is to be read.	
Return values	`string`: Contains the value of the PLC tag.	
Exceptions	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.

Exceptions	`ERuntimeErrorCode.IndexOutOfRange`	The offset lies outside the area range. No value could be read.
	`ERuntimeErrorCode.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	`ERuntimeErrorCode.NotUpToDate`	The stored tag list must be updated.
	`ERuntimeErrorCode.BufferOverflow`	The buffer provided is too small.
	`ERuntimeErrorCode.NotEnoughMemory`	Unexpected error occurred; turn on logging and contact support.
	`ERuntimeErrorCode.InternalError`	Internal error occurred; turn on logging and contact support.

ReadWString()

Reads the value of a PLC tag.

Table 8-118 ReadWString() - Native C++

Syntax	`ERuntimeErrorCode IInstance::ReadWString(`
	`const WCHAR* in_Tag,`
	`WCHAR* out_Value,`
	`const UINT32 in_BufferSize,`
	`UINT32* out_CharsRead`
	`);`	

Parameters	• `const WCHAR* in_Tag:` The name of the PLC tag that is to be read. • `WCHAR* out_Value:` Contains the value of the PLC tag • `const UINT32 in_BufferSize:` Size of the buffer provided • `UINT32* out_CharsRead:` Number of characters read	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.
	`SREC_INDEX_OUT_OF_RANGE`	The offset lies outside the area range. No value could be read.
	`SREC_DOES_NOT_EXIST`	The entry does not exist in the stored tag list.
	`SREC_NOT_SUPPORTED`	Access to entire structures or arrays is not supported.
	`SREC_TYPE_MISMATCH`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Return values	SREC_BUFFER_OVERFLOW	The buffer provided is too small.
	SREC_NOT_ENOUGH_MEMORY	Unexpected error occurred; turn on logging and contact support.
	SREC_INTERNAL_ERROR	Internal error occurred; turn on logging and contact support.

Table 8-119 ReadWString() - .NET (C#)

Syntax	`wstring ReadWString(`
	`wstring in_Tag`
	`);`	

Parameters	• `wstring in_Tag:` The name of the PLC tag that is to be read.	
Return values	`wstring`: Contains the value of the PLC tag.	
Exceptions	Runtime error code	Description
	ERuntimeErrorCode.InterfaceRemoved	The instance is not registered in Runtime Manager.
	ERuntimeErrorCode.Timeout	The function does not return on time.
	ERuntimeErrorCode.InstanceNotRunning	The process of the simulated PLC instance is not running.
	ERuntimeErrorCode.IndexOutOfRange	The offset lies outside the area range. No value could be read.
	ERuntimeErrorCode.DoesNotExist	The entry does not exist in the stored tag list.
	ERuntimeErrorCode.NotSupported	Access to entire structures or arrays is not supported.
	ERuntimeErrorCode.TypeMismatch	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	ERuntimeErrorCode.NotUpToDate	The stored tag list must be updated.
	ERuntimeErrorCode.BufferOverflow	The buffer provided is too small.
	ERuntimeErrorCode.NotEnoughMemory	Unexpected error occurred; turn on logging and contact support.
	ERuntimeErrorCode.InternalError	Internal error occurred; turn on logging and contact support.

8.6.6 I/O access via tag name - Writing

Individual access to I/O data is used for displaying and writing values that are not refreshed regularly in a graphical user interface (GUI).

NOTE

To simulate a regular exchange of signals, create a signal list for each set of signals. Use this signal list for all further accesses. Create a new list as soon as the set of signals changes.

For the signal lists use the functions `ReadSignals()` and `WriteSignals()`.

Write()

Writes the value of a PLC tag.

Table 8-120 Write() - Native C++

Syntax	`ERuntimeErrorCode Write(WCHAR* in_Tag, SDataValue* in_Value);`	
Parameters	• `WCHAR* in_Tag:` The name of the PLC tag that is to be written. • `SDataValue* in_Value:` Contains the value and the expected type of the PLC tag. The `UNSPECIFIC` and `STRUCT` types are not supported. Structures and fields can be emulated through signal lists and then be read by using the `ReadSignals()` function and written by using the `WriteSignals()` function.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.
	`SREC_INDEX_OUT_OF_RANGE`	The offset lies outside the area range. No value could be written.
	`SREC_DOES_NOT_EXIST`	The entry does not exist in the stored tag list.
	`SREC_NOT_SUPPORTED`	Access to entire structures or arrays is not supported.
	`SREC_TYPE_MISMATCH`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.
	`SREC_WRONG_ARGUMENT`	The expected type is `UNSPECIFIC`.
	`SREC_NOT_ENOUGH_MEMORY`	Unexpected error occurred; turn on logging and contact support.
	`SREC_INTERNAL_ERROR`	Internal error occurred; turn on logging and contact support.

Table 8-121 Write() - .NET (C#)

Syntax	`void Write(string in_Tag SDataValue in_Value)`	
Parameters	• `string in_Tag:` The name of the PLC tag that is to be written. • `SDataValue in_Value:` Contains the value and the expected type of the PLC tag. The `UNSPECIFIC` and `STRUCT` types are not supported. Structures and fields can be emulated through signal lists and then be written by using the `WriteSignals()` function.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.

Exceptions	ERuntimeErrorCode.IndexOutOfRange	The offset lies outside the area range. No value could be written.
	ERuntimeErrorCode.DoesNotExist	The entry does not exist in the stored tag list.
	ERuntimeErrorCode.NotSupported	Access to entire structures or arrays is not supported.
	ERuntimeErrorCode.TypeMismatch	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	ERuntimeErrorCode.NotUpToDate	The stored tag list must be updated.
	ERuntimeErrorCode.WrongArgument	The expected type is UNSPECIFIC.
	ERuntimeErrorCode.NotEnoughMemory	Unexpected error occurred; turn on logging and contact support.
	ERuntimeErrorCode.InternalError	Internal error occurred; turn on logging and contact support.

WriteBool()

Writes the value of a PLC tag.

Table 8-122 WriteBool() - Native C++

Syntax	ERuntimeErrorCode WriteBool(WCHAR* in_Tag, bool in_Value);	
Parameters	• WCHAR* in_Tag: The name of the PLC tag that is to be written. • bool in_Value: Contains the value of the PLC tag.	
Return values	Runtime error code	Description
	SREC_OK	The function is successful.
	SREC_INTERFACE_REMOVED	The instance is not registered in Runtime Manager.
	SREC_TIMEOUT	The function does not return on time.
	SREC_INSTANCE_NOT_RUNNING	The process of the simulated PLC instance is not running.
	SREC_INDEX_OUT_OF_RANGE	The offset lies outside the area range. No value could be written.
	SREC_DOES_NOT_EXIST	The entry does not exist in the stored tag list.
	SREC_NOT_SUPPORTED	Access to entire structures or arrays is not supported.
	SREC_TYPE_MISMATCH	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	SREC_NOT_UP_TO_DATE	The stored tag list must be updated.
	SREC_NOT_ENOUGH_MEMORY	Unexpected error occurred; turn on logging and contact support.
	SREC_INTERNAL_ERROR	Internal error occurred; turn on logging and contact support.

Table 8-123 WriteBool() - .NET (C#)

Syntax	`void WriteBool(string in_Tag bool in_Value)`	
Parameters	• `string in_Tag`: The name of the PLC tag that is to be written. • `bool in_Value`: Contains the value of the PLC tag.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.
	`ERuntimeErrorCode.IndexOutOfRange`	The offset lies outside the area range. No value could be written.
	`ERuntimeErrorCode.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 266-267).
	`ERuntimeErrorCode.NotUpToDate`	The stored tag list must be updated.
	`ERuntimeErrorCode.NotEnoughMemory`	Unexpected error occurred; turn on logging and contact support.
	`ERuntimeErrorCode.InternalError`	Internal error occurred; turn on logging and contact support.

WriteInt8()

Writes the value of a PLC tag.

Table 8-124 WriteInt8() - Native C++

Syntax	`ERuntimeErrorCode WriteInt8(WCHAR* in_Tag, INT8 in_Value);`	
Parameters	• `WCHAR* in_Tag`: The name of the PLC tag that is to be written. • `INT8 in_Value`: Contains the value of the PLC tag.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.
	`SREC_INDEX_OUT_OF_RANGE`	The offset lies outside the area range. No value could be written.
	`SREC_DOES_NOT_EXIST`	The entry does not exist in the stored tag list.

Return values	SREC_NOT_SUPPORTED	Access to entire structures or arrays is not supported.
	SREC_TYPE_MISMATCH	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	SREC_NOT_UP_TO_DATE	The stored tag list must be updated.
	SREC_NOT_ENOUGH_MEMORY	Unexpected error occurred; turn on logging and contact support.
	SREC_INTERNAL_ERROR	Internal error occurred; turn on logging and contact support.

Table 8-125 WriteInt8() - .NET (C#)

Syntax	`void WriteInt8(string in_Tag Int8 in_Value)`	
Parameters	• `string in_Tag:` The name of the PLC tag that is to be written. • `Int8 in_Value:` Contains the value of the PLC tag.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.
	`ERuntimeErrorCode.IndexOutOfRange`	The offset lies outside the area range. No value could be written.
	`ERuntimeErrorCode.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	`ERuntimeErrorCode.NotUpToDate`	The stored tag list must be updated.
	`ERuntimeErrorCode.NotEnoughMemory`	Unexpected error occurred; turn on logging and contact support.
	`ERuntimeErrorCode.InternalError`	Internal error occurred; turn on logging and contact support.

WriteInt16()

Writes the value of a PLC tag.

Table 8-126 WriteInt16() - Native C++

Syntax	`ERuntimeErrorCode WriteInt16(WCHAR* in_Tag, INT16 in_Value);`	
Parameters	• `WCHAR* in_Tag:` The name of the PLC tag that is to be written. • `INT16 in_Value:` Contains the value of the PLC tag.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.
	`SREC_INDEX_OUT_OF_RANGE`	The offset lies outside the area range. No value could be written.
	`SREC_DOES_NOT_EXIST`	The entry does not exist in the stored tag list.
	`SREC_NOT_SUPPORTED`	Access to entire structures or arrays is not supported.
	`SREC_TYPE_MISMATCH`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.
	`SREC_NOT_ENOUGH_MEMORY`	Unexpected error occurred; turn on logging and contact support.
	`SREC_INTERNAL_ERROR`	Internal error occurred; turn on logging and contact support.

Table 8-127 WriteInt16() - .NET (C#)

Syntax	`void WriteInt16(string in_Tag Int16 in_Value)`	
Parameters	• `string in_Tag:` The name of the PLC tag that is to be written. • `Int16 in_Value:` Contains the value of the PLC tag.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.
	`ERuntimeErrorCode.IndexOutOfRange`	The offset lies outside the area range. No value could be written.
	`ERuntimeErrorCode.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).

Exceptions	`ERuntimeErrorCode.NotUpToDate`	The stored tag list must be updated.
	`ERuntimeErrorCode.NotEnoughMemory`	Unexpected error occurred; turn on logging and contact support.
	`ERuntimeErrorCode.InternalError`	Internal error occurred; turn on logging and contact support.

WriteInt32()

Writes the value of a PLC tag.

Table 8-128 WriteInt32() - Native C++

Syntax	`ERuntimeErrorCode WriteInt32(WCHAR* in_Tag, INT32 in_Value);`	
Parameters	• `WCHAR* in_Tag:` The name of the PLC tag that is to be written. • `INT32 in_Value:` Contains the value of the PLC tag.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.
	`SREC_INDEX_OUT_OF_RANGE`	The offset lies outside the area range. No value could be written.
	`SREC_DOES_NOT_EXIST`	The entry does not exist in the stored tag list.
	`SREC_NOT_SUPPORTED`	Access to entire structures or arrays is not supported.
	`SREC_TYPE_MISMATCH`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.
	`SREC_NOT_ENOUGH_MEMORY`	Unexpected error occurred; turn on logging and contact support.
	`SREC_INTERNAL_ERROR`	Internal error occurred; turn on logging and contact support.

Table 8-129 WriteInt32() - .NET (C#)

Syntax	`void WriteInt32(string in_Tag Int32 in_Value)`	
Parameters	• `string in_Tag:` The name of the PLC tag that is to be written. • `Int32 in_Value:` Contains the value of the PLC tag.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.

Exceptions	ERuntimeErrorCode.InstanceNotRunning	The process of the simulated PLC instance is not running.
	ERuntimeErrorCode.IndexOutOfRange	The offset lies outside the area range. No value could be written.
	ERuntimeErrorCode.DoesNotExist	The entry does not exist in the stored tag list.
	ERuntimeErrorCode.NotSupported	Access to entire structures or arrays is not supported.
	ERuntimeErrorCode.TypeMismatch	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	ERuntimeErrorCode.NotUpToDate	The stored tag list must be updated.
	ERuntimeErrorCode.NotEnoughMemory	Unexpected error occurred; turn on logging and contact support.
	ERuntimeErrorCode.InternalError	Internal error occurred; turn on logging and contact support.

WriteInt64()

Writes the value of a PLC tag.

Table 8-130 WriteInt64() - Native C++

Syntax	`ERuntimeErrorCode WriteInt64(WCHAR* in_Tag, INT64 in_Value);`	
Parameters	• `WCHAR* in_Tag:` The name of the PLC tag that is to be written. • `INT64 in_Value:` Contains the value of the PLC tag.	
Return values	Runtime error code	Description
	SREC_OK	The function is successful.
	SREC_INTERFACE_REMOVED	The instance is not registered in Runtime Manager.
	SREC_TIMEOUT	The function does not return on time.
	SREC_INSTANCE_NOT_RUNNING	The process of the simulated PLC instance is not running.
	SREC_INDEX_OUT_OF_RANGE	The offset lies outside the area range. No value could be written.
	SREC_DOES_NOT_EXIST	The entry does not exist in the stored tag list.
	SREC_NOT_SUPPORTED	Access to entire structures or arrays is not supported.
	SREC_TYPE_MISMATCH	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	SREC_NOT_UP_TO_DATE	The stored tag list must be updated.
	SREC_NOT_ENOUGH_MEMORY	Unexpected error occurred; turn on logging and contact support.
	SREC_INTERNAL_ERROR	Internal error occurred; turn on logging and contact support.

Table 8-131 WriteInt64() - .NET (C#)

Syntax	`void WriteInt64(string in_Tag Int64 in_Value)`	
Parameters	• `string in_Tag:` The name of the PLC tag that is to be written. • `Int64 in_Value:` Contains the value of the PLC tag.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.
	`ERuntimeErrorCode.IndexOutOfRange`	The offset lies outside the area range. No value could be written.
	`ERuntimeErrorCode.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	`ERuntimeErrorCode.NotUpToDate`	The stored tag list must be updated.
	`ERuntimeErrorCode.NotEnoughMemory`	Unexpected error occurred; turn on logging and contact support.
	`ERuntimeErrorCode.InternalError`	Internal error occurred; turn on logging and contact support.

WriteUInt8()

Writes the value of a PLC tag.

Table 8-132 WriteUInt8() - Native C++

Syntax	`ERuntimeErrorCode WriteUInt8(WCHAR* in_Tag, UINT8 in_Value);`	
Parameters	• `WCHAR* in_Tag:` The name of the PLC tag that is to be written. • `UINT8 in_Value:` Contains the value of the PLC tag.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.
	`SREC_INDEX_OUT_OF_RANGE`	The offset lies outside the area range. No value could be written.
	`SREC_DOES_NOT_EXIST`	The entry does not exist in the stored tag list.
	`SREC_NOT_SUPPORTED`	Access to entire structures or arrays is not supported.

Return values	SREC_TYPE_MISMATCH	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	SREC_NOT_UP_TO_DATE	The stored tag list must be updated.
	SREC_NOT_ENOUGH_MEMORY	Unexpected error occurred; turn on logging and contact support.
	SREC_INTERNAL_ERROR	Internal error occurred; turn on logging and contact support.

Table 8-133 WriteUInt8() - .NET (C#)

Syntax	void WriteUInt8(string in_Tag UInt8 in_Value)	
Parameters	• string in_Tag: The name of the PLC tag that is to be written. • UInt8 in_Value: Contains the value of the PLC tag.	
Exceptions	Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException	
	Runtime error code	Description
	ERuntimeErrorCode.InterfaceRemoved	The instance is not registered in Runtime Manager.
	ERuntimeErrorCode.Timeout	The function does not return on time.
	ERuntimeErrorCode.InstanceNotRunning	The process of the simulated PLC instance is not running.
	ERuntimeErrorCode.IndexOutOfRange	The offset lies outside the area range. No value could be written.
	ERuntimeErrorCode.DoesNotExist	The entry does not exist in the stored tag list.
	ERuntimeErrorCode.NotSupported	Access to entire structures or arrays is not supported.
	ERuntimeErrorCode.TypeMismatch	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	ERuntimeErrorCode.NotUpToDate	The stored tag list must be updated.
	ERuntimeErrorCode.NotEnoughMemory	Unexpected error occurred; turn on logging and contact support.
	ERuntimeErrorCode.InternalError	Internal error occurred; turn on logging and contact support.

WriteUInt16()

Reads the value of a PLC tag.

Table 8-134 WriteUInt16() - Native C++

Syntax	ERuntimeErrorCode WriteUInt16(WCHAR* in_Tag, UINT16 in_Value);	
Parameters	• WCHAR* in_Tag: The name of the PLC tag that is to be written. • UINT16 in_Value: Contains the value of the PLC tag.	
Return values	Runtime error code	Description
	SREC_OK	The function is successful.
	SREC_INTERFACE_REMOVED	The instance is not registered in Runtime Manager.

Return values	SREC_TIMEOUT	The function does not return on time.
	SREC_INSTANCE_NOT_RUNNING	The process of the simulated PLC instance is not running.
	SREC_INDEX_OUT_OF_RANGE	The offset lies outside the area range. No value could be written.
	SREC_DOES_NOT_EXIST	The entry does not exist in the stored tag list.
	SREC_NOT_SUPPORTED	Access to entire structures or arrays is not supported.
	SREC_TYPE_MISMATCH	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	SREC_NOT_UP_TO_DATE	The stored tag list must be updated.
	SREC_NOT_ENOUGH_MEMORY	Unexpected error occurred; turn on logging and contact support.
	SREC_INTERNAL_ERROR	Internal error occurred; turn on logging and contact support.

Table 8-135 WriteUInt16() - .NET (C#)

Syntax	`void WriteUInt16(string in_Tag UInt16 in_Value)`	
Parameters	• `string in_Tag:` The name of the PLC tag that is to be written. • `UInt16 in_Value:` Contains the value of the PLC tag.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.
	`ERuntimeErrorCode.IndexOutOfRange`	The offset lies outside the area range. No value could be written.
	`ERuntimeErrorCode.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	`ERuntimeErrorCode.NotUpToDate`	The stored tag list must be updated.
	`ERuntimeErrorCode.NotEnoughMemory`	Unexpected error occurred; turn on logging and contact support.
	`ERuntimeErrorCode.InternalError`	Internal error occurred; turn on logging and contact support.

WriteUInt32()

Writes the value of a PLC tag.

Table 8-136 WriteUInt32() - Native C++

Syntax	`ERuntimeErrorCode WriteUInt32(WCHAR* in_Tag, UINT32 in_Value);`	
Parameters	• `WCHAR* in_Tag:` The name of the PLC tag that is to be written. • `UINT32 in_Value:` Contains the value of the PLC tag.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.
	`SREC_INDEX_OUT_OF_RANGE`	The offset lies outside the area range. No value could be written.
	`SREC_DOES_NOT_EXIST`	The entry does not exist in the stored tag list.
	`SREC_NOT_SUPPORTED`	Access to entire structures or arrays is not supported.
	`SREC_TYPE_MISMATCH`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.
	`SREC_NOT_ENOUGH_MEMORY`	Unexpected error occurred; turn on logging and contact support.
	`SREC_INTERNAL_ERROR`	Internal error occurred; turn on logging and contact support.

Table 8-137 WriteUInt32() - .NET (C#)

Syntax	`void WriteUInt32(string in_Tag UInt32 in_Value)`	
Parameters	• `string in_Tag:` The name of the PLC tag that is to be written. • `UInt32 in_Value:` Contains the value of the PLC tag.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.
	`ERuntimeErrorCode.IndexOutOfRange`	The offset lies outside the area range. No value could be written.
	`ERuntimeErrorCode.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).

Exceptions	ERuntimeErrorCode.NotUpToDate	The stored tag list must be updated.
	ERuntimeErrorCode.NotEnoughMemory	Unexpected error occurred; turn on logging and contact support.
	ERuntimeErrorCode.InternalError	Internal error occurred; turn on logging and contact support.

WriteUInt64()

Writes the value of a PLC tag.

Table 8-138 WriteUInt64() - Native C++

Syntax	ERuntimeErrorCode WriteUInt64(WCHAR* in_Tag, UINT64 in_Value);	
Parameters	• WCHAR* in_Tag: The name of the PLC tag that is to be written. • UINT64 in_Value: Contains the value of the PLC tag.	
Return values	Runtime error code	Description
	SREC_OK	The function is successful.
	SREC_INTERFACE_REMOVED	The instance is not registered in Runtime Manager.
	SREC_TIMEOUT	The function does not return on time.
	SREC_INSTANCE_NOT_RUNNING	The process of the simulated PLC instance is not running.
	SREC_INDEX_OUT_OF_RANGE	The offset lies outside the area range. No value could be written.
	SREC_DOES_NOT_EXIST	The entry does not exist in the stored tag list.
	SREC_NOT_SUPPORTED	Access to entire structures or arrays is not supported.
	SREC_TYPE_MISMATCH	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	SREC_NOT_UP_TO_DATE	The stored tag list must be updated.
	SREC_NOT_ENOUGH_MEMORY	Unexpected error occurred; turn on logging and contact support.
	SREC_INTERNAL_ERROR	Internal error occurred; turn on logging and contact support.

Table 8-139 WriteUInt64() - .NET (C#)

Syntax	void WriteUInt64(string in_Tag UInt64 in_Value)	
Parameters	• string in_Tag: The name of the PLC tag that is to be written. • UInt64 in_Value: Contains the value of the PLC tag.	
Exceptions	Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException	
	Runtime error code	Description
	ERuntimeErrorCode.InterfaceRemoved	The instance is not registered in Runtime Manager.
	ERuntimeErrorCode.Timeout	The function does not return on time.

Exceptions	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.
	`ERuntimeErrorCode.IndexOutOfRange`	The offset lies outside the area range. No value could be written.
	`ERuntimeErrorCode.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	`ERuntimeErrorCode.NotUpToDate`	The stored tag list must be updated.
	`ERuntimeErrorCode.NotEnoughMemory`	Unexpected error occurred; turn on logging and contact support.
	`ERuntimeErrorCode.InternalError`	Internal error occurred; turn on logging and contact support.

WriteFloat()

Writes the value of a PLC tag.

Table 8-140 WriteFloat() - Native C++

Syntax	`ERuntimeErrorCode WriteFloat(WCHAR* in_Tag, float in_Value);`	
Parameters	• `WCHAR* in_Tag:` The name of the PLC tag that is to be written. • `float in_Value:` Contains the value of the PLC tag.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.
	`SREC_INDEX_OUT_OF_RANGE`	The offset lies outside the area range. No value could be written.
	`SREC_DOES_NOT_EXIST`	The entry does not exist in the stored tag list.
	`SREC_NOT_SUPPORTED`	Access to entire structures or arrays is not supported.
	`SREC_TYPE_MISMATCH`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.
	`SREC_NOT_ENOUGH_MEMORY`	Unexpected error occurred; turn on logging and contact support.
	`SREC_INTERNAL_ERROR`	Internal error occurred; turn on logging and contact support.

Table 8-141 WriteFloat() - .NET (C#)

Syntax	`void WriteFloat(string in_Tag float in_Value)`	
Parameters	• `string in_Tag:` The name of the PLC tag that is to be written. • `float in_Value:` Contains the value of the PLC tag.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.
	`ERuntimeErrorCode.IndexOutOfRange`	The offset lies outside the area range. No value could be written.
	`ERuntimeErrorCode.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	`ERuntimeErrorCode.NotUpToDate`	The stored tag list must be updated.
	`ERuntimeErrorCode.NotEnoughMemory`	Unexpected error occurred; turn on logging and contact support.
	`ERuntimeErrorCode.InternalError`	Internal error occurred; turn on logging and contact support.

WriteDouble()

Writes the value of a PLC tag.

Table 8-142 WriteDouble() - Native C++

Syntax	`ERuntimeErrorCode WriteDouble(WCHAR* in_Tag, double in_Value);`	
Parameters	• `WCHAR* in_Tag:` The name of the PLC tag that is to be written. • `double in_Value:` Contains the value of the PLC tag.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.
	`SREC_INDEX_OUT_OF_RANGE`	The offset lies outside the area range. No value could be written.
	`SREC_DOES_NOT_EXIST`	The entry does not exist in the stored tag list.
	`SREC_NOT_SUPPORTED`	Access to entire structures or arrays is not supported.

Return values	SREC_TYPE_MISMATCH	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	SREC_NOT_UP_TO_DATE	The stored tag list must be updated.
	SREC_NOT_ENOUGH_MEMORY	Unexpected error occurred; turn on logging and contact support.
	SREC_INTERNAL_ERROR	Internal error occurred; turn on logging and contact support.

Table 8-143 WriteDouble() - .NET (C#)

Syntax	void WriteDouble(string in_Tag double in_Value)	
Parameters	• string in_Tag: The name of the PLC tag that is to be written. • double in_Value: Contains the value of the PLC tag.	
Exceptions	Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException	
	Runtime error code	Description
	ERuntimeErrorCode.InterfaceRemoved	The instance is not registered in Runtime Manager.
	ERuntimeErrorCode.Timeout	The function does not return on time.
	ERuntimeErrorCode.InstanceNotRunning	The process of the simulated PLC instance is not running.
	ERuntimeErrorCode.IndexOutOfRange	The offset lies outside the area range. No value could be written.
	ERuntimeErrorCode.DoesNotExist	The entry does not exist in the stored tag list.
	ERuntimeErrorCode.NotSupported	Access to entire structures or arrays is not supported.
	ERuntimeErrorCode.TypeMismatch	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	ERuntimeErrorCode.NotUpToDate	The stored tag list must be updated.
	ERuntimeErrorCode.NotEnoughMemory	Unexpected error occurred; turn on logging and contact support.
	ERuntimeErrorCode.InternalError	Internal error occurred; turn on logging and contact support.

WriteChar()

Writes the value of a PLC tag.

Table 8-144 WriteChar() - Native C++

Syntax	ERuntimeErrorCode WriteChar(WCHAR* in_Tag, char in_Value);	
Parameters	• WCHAR* in_Tag: The name of the PLC tag that is to be written. • char in_Value: Contains the value of the PLC tag.	
Return values	Runtime error code	Description
	SREC_OK	The function is successful.
	SREC_INTERFACE_REMOVED	The instance is not registered in Runtime Manager.

Return values	SREC_TIMEOUT	The function does not return on time.
	SREC_INSTANCE_NOT_RUNNING	The process of the simulated PLC instance is not running.
	SREC_INDEX_OUT_OF_RANGE	The offset lies outside the area range. No value could be written.
	SREC_DOES_NOT_EXIST	The entry does not exist in the stored tag list.
	SREC_NOT_SUPPORTED	Access to entire structures or arrays is not supported.
	SREC_TYPE_MISMATCH	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	SREC_NOT_UP_TO_DATE	The stored tag list must be updated.
	SREC_NOT_ENOUGH_MEMORY	Unexpected error occurred; turn on logging and contact support.
	SREC_INTERNAL_ERROR	Internal error occurred; turn on logging and contact support.

Table 8-145 WriteChar() - .NET (C#)

Syntax	`void WriteChar(string in_Tag sbyte in_Value)`	
Parameters	• `string in_Tag:` The name of the PLC tag that is to be written. • `sbyte in_Value:` Contains the value of the PLC tag.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.
	`ERuntimeErrorCode.IndexOutOfRange`	The offset lies outside the area range. No value could be written.
	`ERuntimeErrorCode.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	`ERuntimeErrorCode.NotUpToDate`	The stored tag list must be updated.
	`ERuntimeErrorCode.NotEnoughMemory`	Unexpected error occurred; turn on logging and contact support.
	`ERuntimeErrorCode.InternalError`	Internal error occurred; turn on logging and contact support.

WriteWChar()

Writes the value of a PLC tag.

Table 8-146 WriteWChar() - Native C++

Syntax	`ERuntimeErrorCode WriteWChar(WCHAR* in_Tag, WCHAR in_Value);`	
Parameters	• `WCHAR* in_Tag:` The name of the PLC tag that is to be written. • `WCHAR in_Value:` Contains the value of the PLC tag.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.
	`SREC_INDEX_OUT_OF_RANGE`	The offset lies outside the area range. No value could be written.
	`SREC_DOES_NOT_EXIST`	The entry does not exist in the stored tag list.
	`SREC_NOT_SUPPORTED`	Access to entire structures or arrays is not supported.
	`SREC_TYPE_MISMATCH`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.
	`SREC_NOT_ENOUGH_MEMORY`	Unexpected error occurred; turn on logging and contact support.
	`SREC_INTERNAL_ERROR`	Internal error occurred; turn on logging and contact support.

Table 8-147 WriteWChar() - .NET (C#)

Syntax	`void WriteWChar(string in_Tag char in_Value)`	
Parameters	• `string in_Tag:` The name of the PLC tag that is to be written. • `char in_Value:` Contains the value of the PLC tag.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.
	`SREC_INDEX_OUT_OF_RANGE`	The offset lies outside the area range. No value could be written.
	`ERuntimeErrorCode.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).

Exceptions	`ERuntimeErrorCode.NotUpToDate`	The stored tag list must be updated.
	`ERuntimeErrorCode.NotEnoughMemory`	Unexpected error occurred; turn on logging and contact support.
	`ERuntimeErrorCode.InternalError`	Internal error occurred; turn on logging and contact support.

WriteSignals()

Writes multiple signals within an API call.

When the function is called for the first time, it stores internal information in the structures `SDataValueByName*` to improve the performance of the subsequent calls.

NOTE

To simulate a regular exchange of signals, create a signal list for each set of signals. Use this signal list for all further accesses. Create a new list as soon as the set of signals changes.

Table 8-148 WriteSignals() - Native C++

Syntax	`ERuntimeErrorCode WriteSignals(SDataValueByName* inout_Signals, UINT32 in_SignalCount);`	
Parameters	• `SDataValueByName* inout_Signals:` Contains the name, the value and the expected type of the PLC tag. The `UNSPECIFIC` and `STRUCT` types are not supported. • `UINT32 in_SignalCount:` Number of signals.	
Signal error	Error code	Description
	`SREC_INDEX_OUT_OF_RANGE`	Offset or bits are invalid.
Return values	Runtime error code	Condition
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.
	`SREC_DOES_NOT_EXIST`	The entry does not exist in the stored tag list.
	`SREC_NOT_SUPPORTED`	Access to entire structures or arrays is not supported.
	`SREC_TYPE_MISMATCH`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.
	`SREC_WRONG_ARGUMENT`	The expected type is `UNSPECIFIC`.

Table 8-149 WriteSignals() - .NET (C#)

Syntax	`void WriteSignals(SDataValueByName[] in_Signals)`	
Parameters	• `SDataValueByName` Contains the name, the value and the expected type of the PLC tag. The `UNSPECIFIC` and `STRUCT` types are not supported.	
Signal error	Error code	Description
	`ERuntimeErrorCode.IndexOutOfRange`	Offset or bits are invalid.
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.
	`ERuntimeErrorCode.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 266-267).
	`ERuntimeErrorCode.NotUpToDate`	The stored tag list must be updated.
	`ERuntimeErrorCode.WrongArgument`	The expected type is `UNSPECIFIC`.

WriteString()

Writes the value of a PLC tag.

Table 8-150 WriteString() - Native C++

Syntax	`ERuntimeErrorCode IInstance::WriteString(`
	`const WCHAR* in_Tag,`
	`const CHAR* in_Value,`
	`const UINT32 in_CharsToWrite`
	`);`	

Parameters	• `const WCHAR* in_Tag:` The name of the PLC tag that is to be written. • `CHAR* in_Value:` Contains the value of the PLC tag • `const UINT32 in_CharsToWrite:` Number of characters to write	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.
	`SREC_INDEX_OUT_OF_RANGE`	The offset lies outside the area range. No value could be written.
	`SREC_DOES_NOT_EXIST`	The entry does not exist in the stored tag list.

Return values	SREC_NOT_SUPPORTED	Access to entire structures or arrays is not supported.
	SREC_TYPE_MISMATCH	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	SREC_NOT_UP_TO_DATE	The stored tag list must be updated.
	SREC_BUFFER_OVERFLOW	The characters to be written are greater than the maximum value of the tag.
	SREC_NOT_ENOUGH_MEMORY	Unexpected error occurred; turn on logging and contact support.
	SREC_INTERNAL_ERROR	Internal error occurred; turn on logging and contact support.

Table 8-151 WriteString() - .NET (C#)

Syntax	`void WriteString(`
	`string in_Tag,`
	`string in_Value`
	`);`	

Parameters	• `string in_Tag:` The name of the PLC tag that is to be written. • `string in_Value:` The value to be written to the PLC tag.	
Exceptions	Runtime error code	Description
	ERuntimeErrorCode.InterfaceRemoved	The instance is not registered in Runtime Manager.
	ERuntimeErrorCode.Timeout	The function does not return on time.
	ERuntimeErrorCode.InstanceNotRunning	The process of the simulated PLC instance is not running.
	ERuntimeErrorCode.IndexOutOfRange	The offset lies outside the area range. No value could be written.
	ERuntimeErrorCode.DoesNotExist	The entry does not exist in the stored tag list.
	ERuntimeErrorCode.NotSupported	Access to entire structures or arrays is not supported.
	ERuntimeErrorCode.TypeMismatch	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	ERuntimeErrorCode.NotUpToDate	The stored tag list must be updated.
	ERuntimeErrorCode.BufferOverflow	The characters to be written are greater than the maximum value of the tag.
	ERuntimeErrorCode.NotEnoughMemory	Unexpected error occurred; turn on logging and contact support.
	ERuntimeErrorCode.InternalError	Internal error occurred; turn on logging and contact support.

WriteWString()

Writes the value of a PLC tag.

Table 8-152 WriteWString() - Native C++

Syntax	`ERuntimeErrorCode IInstance::WriteWString(`
	`const WCHAR* in_Tag,`
	`const WCHAR* in_Value,`
	`const UINT32 in_CharsToWrite,`
	`);`	

Parameters	• `const WCHAR* in_Tag:` The name of the PLC tag that is to be written. • `const WCHAR* in_Value:` Contains the value of the PLC tag • `const UINT32 in_CharsToWrite:` Number of characters that are to be written.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.
	`SREC_INDEX_OUT_OF_RANGE`	The offset lies outside the area range. No value could be written.
	`SREC_DOES_NOT_EXIST`	The entry does not exist in the stored tag list.
	`SREC_NOT_SUPPORTED`	Access to entire structures or arrays is not supported.
	`SREC_TYPE_MISMATCH`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.
	`SREC_BUFFER_OVERFLOW`	The characters to be written are greater than the maximum value of the tag.
	`SREC_NOT_ENOUGH_MEMORY`	Unexpected error occurred; turn on logging and contact support.
	`SREC_INTERNAL_ERROR`	Internal error occurred; turn on logging and contact support.

Table 8-153 WriteWString() - .NET (C#)

Syntax	`void WriteWString(`
	`wstring in_Tag,`
	`wstring in_Value`
	`);`	

Parameters	• `wstring in_Tag:` The name of the PLC tag that is to be written. • `wstring in_Value:` The value to be written to the PLC tag.	
Exceptions	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.

Exceptions	`ERuntimeErrorCode.IndexOutOfRange`	The offset lies outside the area range. No value could be written.
	`ERuntimeErrorCode.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 265).
	`ERuntimeErrorCode.NotUpToDate`	The stored tag list must be updated.
	`ERuntimeErrorCode.BufferOverflow`	The characters to be written are greater than the maximum value of the tag.
	`ERuntimeErrorCode.NotEnoughMemory`	Unexpected error occurred; turn on logging and contact support.
	`ERuntimeErrorCode.InternalError`	Internal error occurred; turn on logging and contact support.

GetSystemTime()

Returns the virtual system time of the simulated PLC instance. Returns an empty structure when the function fails.

Table 8-154 GetSystemTime() - Native C++

Syntax	`SYSTEMTIME GetSystemTime();`
Parameters	None
Return values	`SYSTEMTIME`: System time of the simulated PLC instance.

SetSystemTime()

Sets the virtual system time of the simulated PLC instance. A system time between "Jan 1 1970 00:00:00:000" and "Dec 31 2200 23:59:59:999" is valid.

Table 8-155 SetSystemTime() - Native C++

Syntax	`ERuntimeErrorCode SetSystemTime(SYSTEMTIME in_SystemTime);`	
Parameters	• `SYSTEMTIME in_SystemTime`: System time that is to be set for the simulated PLC instance.	
Return values	Runtime error code	Condition
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_WRONG_ARGUMENT`	The value is outside the limits.

SystemTime { get; set; }

Sets or returns the virtual system time of the simulated PLC instance. A system time between "Jan 1 1970 00:00:00:000" and "Dec 31 2200 23:59:59:999" is valid.

Table 8-156 SystemTime { get; set; } - .NET (C#)

Syntax	`DateTime SystemTime { get; set; }`	
Parameters	None	
Return values	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.WrongArgument`	The value is outside the limits.

GetScaleFactor()

Returns the scaling factor with which the virtual time advances.

Table 8-157 GetScaleFactor() - Native C++

Syntax	`double GetScaleFactor();`
Parameters	None
Return values	`double:` Scaling factor of the virtual time.

SetScaleFactor()

Sets the scaling factor with which the virtual time advances.

Siemens recommends that you start with a small scaling factor and gradually work your way up to a scaling factor where the simulated PLC instance remains in RUN.

A value between 0.01 and 100 is valid. The default setting is 1.

- If the value is less than 1, the virtual time of the simulated PLC instance runs X-times slower than the real time.
- If the value is greater than 1, the virtual time of the simulated PLC instance runs X-times faster than the real time.

A change in the value during runtime only takes effect at the cycle control point.

Table 8-158 SetScaleFactor() - Native C++

Syntax	`ERuntimeErrorCode SetScaleFactor (double in_Value);`	
Parameters	• `double in_Value:` Scaling factor of the virtual time.	
Return values	Runtime error code	Condition
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_WRONG_ARGUMENT`	The value is outside the limits.

ScaleFactor { get; set; }

Sets or returns the scaling factor with which the virtual time advances.

Start with a small scaling factor and incrementally approach a scaling factor at which the simulated PLC instance remains in RUN.

A value between 0.01 and 100 is valid. The default setting is 1.

- If the value is less than 1, the virtual time of the simulated PLC instance runs X-times slower than the real time.
- If the value is greater than 1, the virtual time of the simulated PLC instance runs X-times faster than the real time.

A change in the value during runtime only takes effect at the cycle control point.

Table 8-159 ScaleFactor { get; set; } - .NET (C#)

Syntax	`double ScaleFactor { get; set; }`	
Parameters	None	
Return values	`double`: Scaling factor of the virtual time.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.WrongArgument`	The value is outside the limits.

8.7 Cycle control

GetOperatingMode()

Returns the operating mode (Page 260) of the simulated PLC instance.

Table 8-160 GetOperatingMode() - Native C++

Syntax	`EOperatingMode GetOperatingMode();`
Parameters	None
Return values	`EOperatingMode`: Operating mode of the simulated PLC instance

SetOperatingMode()

Sets the operating mode of the simulated PLC instance.

A change in the value during runtime only takes effect at the synchronization point.

Table 8-161 SetOperatingMode() - Native C++

Syntax	`void SetOperatingMode(EOperatingMode in_OperatingMode);`
Parameters	• `EOperatingMode in_OperatingMode`: Operating mode of the simulated PLC instance

OperatingMode { get; set; }

Returns or sets the operating mode of the simulated PLC instance.

A change in the value during runtime only takes effect at the synchronization point.

Table 8-162 OperatingMode { get; set; } - .NET (C#)

Syntax	`EOperatingMode OperatingMode { get; set; }`	
Parameters	None	
Return values	`EOperatingMode`: Operating mode of the simulated PLC instance	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.

SetSendSyncEventInDefaultModeEnabled()

Sets the `SendSyncEventInDefault` mode. In this mode the `OnSyncPointReached` event is triggered after each cycle end in the Default operating mode. See OnSyncPointReached (Page 201).

Table 8-163 SetSendSyncEventInDefaultModeEnabled() - Native C++

Syntax	`ERuntimeErrorCode SetSendSyncEventInDefaultModeEnabled(bool in_Enable);`	
Parameters	• `bool in_Enable`: If `true`, the `OnSyncPointReached` event is triggered after each cycle in the Default operating mode.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.

IsSendSyncEventInDefaultModeEnabled()

Returns the `SendSyncEventInDefaultMode` mode. When the function fails, the return value is `false`.

Table 8-164 IsSendSyncEventInDefaultModeEnabled() - Native C++

Syntax	`bool IsSendSyncEventInDefaultModeEnabled();`
Parameters	None
Return values	• `false`: The event is not triggered (unless the Sync-Freeze mode is active). • `true`: The event is triggered after every cycle.

IsSendSyncEventInDefaultModeEnabled { get; set; }

Returns or sets the `SendSyncEventInDefaultMode` mode. In this mode the `OnSyncPointReached` event is triggered after each cycle end for every operating mode. If

the event is also to be received in the Default operating mode, set the return value to `true`. See OnSyncPointReached (Page 201).

Table 8-165 IsSendSyncEventInDefaultModeEnabled { get; set; } - .NET (C#)

Syntax	`bool IsSendSyncEventInDefaultModeEnabled { get; set;}`	
Parameters	None	
Return values	• `false`: The event is not triggered (unless the Sync-Freeze mode is active). • `true`: The event is triggered after every cycle.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.

GetOverwrittenMinimalCycleTime_ns()

Returns the overwritten minimum cycle time (in nanoseconds) that is used in the `SingleStep_CT` and `SingleStep_CPT` operating modes.

Table 8-166 GetOverwrittenMinimalCycleTime_ns() - Native C++

Syntax	`INT64 GetOverwrittenMinimalCycleTime_ns();`
Parameters	None
Return values	`INT64`: The overwritten minimum cycle time in nanoseconds.

SetOverwrittenMinimalCycleTime_ns()

Sets the overwritten minimum cycle time (in nanoseconds) that is used in the `SingleStep_CT` and `SingleStep_CPT` operating modes.

A value between 0 and 6000000000 is valid. The default setting is 100 ms.

A change in the value during runtime only takes effect at the cycle control point.

Table 8-167 SetOverwrittenMinimalCycleTime_ns() - Native C++

Syntax	`ERuntimeErrorCode SetOverwrittenMinimalCycleTime_ns(INT64 in_CycleTime_ns);`	
Parameters	• `INT64 in_CycleTime_ns:` The overwritten minimum cycle time in nanoseconds.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_WRONG_ARGUMENT`	The value is outside the limits.

OverwrittenMinimalCycleTime_ns { get; set; }

Returns or sets the overwritten minimum cycle time in nanoseconds that is used in the `SingleStep_CT` and `SingleStep_CPT` operating modes.

A value between 0 and 6000000000 is valid. The default setting is 100 ms.

A change in the value during runtime only takes effect at the cycle control point.

Table 8-168 OverwrittenMinimalCycleTime_ns { get; set; } - .NET (C#)

Syntax	`Int64 OverwrittenMinimalCycleTime_ns { get; set; }`	
Parameters	None	
Return values	`Int64`: The overwritten minimum cycle time in nanoseconds.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.WrongArgument`	The value is outside the limits.

RunToNextSyncPoint()

If the simulated PLC instance is running in a SingleStep operating mode, it is stopped at the synchronization point (Freeze state). The `RunToNextSyncPoint()` function cancels the freeze state. The simulated PLC instance continues to run until the next synchronization point.

Table 8-169 RunToNextSyncPoint() - Native C++

Syntax	`ERuntimeErrorCode RunToNextSyncPoint();` `ERuntimeErrorCode RunToNextSyncPoint(UINT32 in_IoSystemId, UINT64 in_BusTimeStamp_ns);`	
Parameters	The call with parameters is only valid for the `SingleStep_Bus` operating mode. • `UINT32 in_IoSystemId:` Number of the I/O system (PROFIBUS 1..32, PROFINET 100..115) that is to be used for the cycle-controlled synchronization of the simulated PLC instance. • `UINT64 in_BusTimeStamp_ns:` Time stamp in nanoseconds of the respective I/O system at the start of each cycle.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INVALID_OPERATING_STATE`	The instance is in an invalid operating state for processing the call (BOOTING, SHUTTING_DOWN, OFF)
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.
	`SREC_WRONG_ARGUMENT`	For the `SingleStep_Bus` operating mode: The ID of the I/O system is invalid. For all other operating modes: The function call with parameters is invalid.

Table 8-170 RunToNextSyncPoint() - .NET (C#)

Syntax	`void RunToNextSyncPoint();` `void RunToNextSyncPoint(UINT32 in_IoSystemId, UINT64 in_BusTimeStamp_ns);`	
Parameters	The call with parameters is only valid for the `SingleStep_Bus` operating mode. • `UINT32 in_IoSystemId:` Number of the I/O system (PROFIBUS 1..32, PROFINET 100..115) that is to be used for the cycle-controlled synchronization of the simulated PLC instance. • `UINT64 in_BusTimeStamp_ns:` Time stamp in nanoseconds of the respective I/O system at the start of each cycle.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InvalidOperatingState`	The instance is in an invalid operating state for processing the call (BOOTING, SHUTTING_DOWN, OFF)
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.
	`ERuntimeErrorCode.WrongArgument`	For the `SingleStep_Bus` operating mode: The ID of the I/O system is invalid. For all other operating modes: The function call with parameters is invalid.

StartProcessing()

If the simulated PLC instance is running in a TimespanSynchronized operating mode, it is stopped at the synchronization point (Freeze state). The `StartProcessing()` function cancels the freeze state. The simulated PLC instance will now run for at least the requested time before it changes to Freeze state at the next synchronization point.

Table 8-171 StartProcessing() - Native C++

Syntax	`ERuntimeErrorCode StartProcessing(INT64 in_MinimalTimeToRun_ns);`	
Parameters	• `INT64 in_MinimalTimeToRun_ns:` The minimum virtual time (in nanoseconds) that the simulated PLC instance runs before it changes to Freeze state.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the simulated PLC instance is not running.
	`SREC_WRONG_ARGUMENT`	The value is less than 0.

Table 8-172 StartProcessing() - .NET (C#)

Syntax	`void StartProcessing(Int64 in_MinimalTimeToRun_ns);`	
Parameters	• `Int64 in_MinimalTimeToRun_ns:` The minimum virtual time (in nanoseconds) that the simulated PLC instance runs before it changes to Freeze state.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the simulated PLC instance is not running.
	`ERuntimeErrorCode.WrongArgument`	The value is less than 0.

More information

For more information, see sections Virtual and real time (Page 168), Stop simulation (Page 100).

8.8 Virtual and real time

The simulated PLC instance uses two types of internal clocks for the simulation: A virtual clock and a real clock. The virtual clock is always the basis for the user program. It is used by components that are relevant for running the STEP 7 user program, such as cyclic OBs, cycle time monitoring, minimum cycle time, virtual system time and time calculations. Also, the time between two cycle control or single step control points is measured in virtual time.

The virtual time can be accelerated or slowed for simulation and optimization purposes.

The real clock always runs unchanged. It is used by components that are not subject to control processes, for example, communication with STEP 7.

Interruption of the process

Since S7-PLCSIM Advanced runs in a Windows environment, Windows might temporarily suspend the simulated PLC instance process. In such a case, both the virtual and the real clock stop in the simulated PLC instance. They only continue to run when Windows resumes processing.

Virtual system time

When you start S7-PLCSIM Advanced, the virtual system time of the instance starts with the system time of Windows.

The virtual system time is based on the virtual clock. If a scaling factor is used, the system time runs correspondingly faster or slower.

All events from the simulated PLC instance provide a time stamp based on the system time.

NOTE

Difference between system time and local time

- System time**:** UTC ± 0 with daylight saving / standard time
- Local time: UTC ± time zone with daylight saving time / standard time

Time offset

NOTE

Keep in mind that the time information of virtual system time and real local time differs by the time offset that is formed in addition to the selected scaling factor from the time zone offset and the daylight saving time/standard time offset.

Scaling factor

Using a scaling factor, you can speed up or slow down the virtual clock of the simulated PLC instance for simulations.

To set the required scaling factor, enable the grayed out icon on the control panel to the right of your S7-PLCSIM Advanced instance. The symbol then becomes active and you can use the scaling function.

- The default setting is 1, i.e. the virtual time passes the same as the real time.
- **Fast forward:** A scaling factor greater than 1 accelerates the virtual clock.

  Example: Scaling factor 2.0 → The virtual time is running twice as fast.
- **Slow motion**: A scaling factor less than 1 decelerates the virtual clock.

  Example: Scaling factor 0.5 → The progress of the virtual time slows down to 50%.

SetCycleTimeMonitoringMode()

With this function you can chanage the source of the timer for the maximum cycle time monitoring.

Table 8-173 SetCycleTimeMonitoringMode() - Native C++

Syntax	`ERuntimeErrorCode SetCycleTimeMonitoringMode(ECycleTimeMonitoringMode in_CycleTimeMonitoringMode)` `ERuntimeErrorCode SetCycleTimeMonitoringMode(ECycleTimeMonitoringMode in_CycleTimeMonitoringMode, INT64 in_MaxCycleTime_ns)`
Parameters	• `ECycleTimeMonitoringMode in_CycleTimeMonitoringMode:`

	Select one of the following options for the maximum cycle time monitoring: – SRCTMM_DOWNLOADED: The maximum cycle time from the project that was downloaded from STEP 7 is used as maximum cycle time monitoring. – SRCTMM_IGNORED **(default)**: A timer value of one minute is used as maximum cycle time monitoring to prevent a potential error in case of an overflow of cyclic events. See Monitoring overflow (Page 171). – SRCTMM_SPECIFIED: A value that is specified with the in_MaxCycleTime_ns parameter is used as maximum cycle time monitoring. Default: 150 ms. • INT64 in_MaxCycleTime_ns: The user-specific value for the maximum cycle time monitoring. A value between 1000000 and 60000000000 ns (1 millisecond to 1 minute) is valid. If no value is specified in the API, the default value of 150 ms applies.	
Return values	Runtime error code	Description
	SREC_OK	The function is successful.
	SREC_INTERFACE_REMOVED	The instance is not registered in Runtime Manager.
	SREC_TIMEOUT	The function does not return on time.
	SREC_WRONG_ARGUMENT	The cycle time monitoring mode is invalid.
	SREC_INDEX_OUT_OF_RANGE	The user-specific value for the maximum cycle time monitoring is outside the limits.

Table 8-174 SetCycleTimeMonitoringMode() - .NET (C#)

Syntax	void SetCycleTimeMonitoringMode(ECycleTimeMonitoringMode in_CycleTimeMonitoringMode) void SetCycleTimeMonitoringMode(ECycleTimeMonitoringMode in_CycleTimeMonitoringMode, Int64 in_MaxCycleTime_ns)	
Parameters	• ECycleTimeMonitoringMode in_CycleTimeMonitoringMode: Select one of the following options for the maximum cycle time monitoring: – ECycleTimeMonitoringMode.Downloaded: The maximum cycle time from the project that was downloaded from STEP 7 is used as maximum cycle time monitoring. – ECycleTimeMonitoringMode.Ignored **(default)**: A timer value of one minute is used as maximum cycle time monitoring to prevent a potential error in case of an overflow of cyclic events. See Monitoring overflow (Page 171). – ECycleTimeMonitoringMode.Specified: A value that is specified with the in_MaxCycleTime_ns parameter is used as maximum cycle time monitoring. Default: 150 ms. • Int64 in_MaxCycleTime_ns: The user-specific value for the maximum cycle time monitoring. A value between 1000000 and 60000000000 ns (1 millisecond to 1 minute) is valid. If no value is specified in the API, the default value of 150 ms applies.	
Exceptions	Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException	
	Runtime error code	Description
	ERuntimeErrorCode.InterfaceRemoved	The instance is not registered in Runtime Manager.
	ERuntimeErrorCode.Timeout	The function does not return on time.
	ERuntimeErrorCode.WrongArgument	The cycle time monitoring mode is invalid.
	ERuntimeErrorCode.IndexOutOfRange	The user-specific value for the maximum cycle time monitoring is outside the limits.

GetCycleTimeMonitoringMode()

This function returns information on the source of the timer for the maximum cycle time monitoring.

Table 8-175 GetCycleTimeMonitoringMode() - Native C++

Syntax	`ERuntimeErrorCode GetCycleTimeMonitoringMode(ECycleTimeMonitoringMode* out_CycleTimeMonitoringMode, INT64* out_MaxCycleTime_ns)`	
Parameters	• `ECycleTimeMonitoringMode* out_CycleTimeMonitoringMode:` The configured mode for cycle time monitoring. The default setting is `SRCTM_IGNORED`. • `INT64 in_MaxCycleTime_ns:` The user-specific value for the maximum cycle time monitoring. If no value is specified in the API, the default value of 150 ms is returned.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.

Table 8-176 GetCycleTimeMonitoringMode() - .NET (C#)

Syntax	`void GetCycleTimeMonitoringMode(out ECycleTimeMonitoringMode out_CycleTimeMonitoringMode, out Int64 out_MaxCycleTime_ns)`	
Parameters	• `ECycleTimeMonitoringMode out_CycleTimeMonitoringMode:` The configured mode for cycle time monitoring. The default setting is `ECycleTimeMonitoringMode.Ignored`. • `Int64 in_MaxCycleTime_ns:` The user-specific value for the maximum cycle time monitoring. If no value is specified in the API, the default value of 150 ms is returned.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.

8.9 Monitoring overflow

Monitoring of main cycle

The maximum cycle time monitoring for S7-PLCSIM Advanced is one minute.

Monitoring of cyclical events

If your simulation contains cyclic interrupts, the queue of S7-PLCSIM Advanced can overflow for cyclic events. Due to the runtime speed of S7-PLCSIM Advanced compared to real hardware, the time needed to create the diagnostic buffer entry might be longer than the time until the next cyclic interrupt.

In this case, an additional entry is placed in the queue, causing another overflow. In the event of an overflow, S7-PLCSIM Advanced provides diagnostics buffer messages and a red error icon in the project tree in STEP 7.

8.10 Acyclic services

8.10.1 Overview

The acyclic services of S7-PLCSIM Advanced include:

- Read and write processes of parameter and status data from the user program of the PLC to the I/O modules
- Interrupt and event information which the I/O modules send to the CPU.

Read and write operations

Events triggered by the STEP 7 user program that have logged on for the notification:

Table 8-177 Events: Read and write operations

SFB	Name	API method (alarm)	API event for triggering the SFB
52	RDREC	ReadRecordDone (Page 174)	OnDataRecordRead (Page 206)
53	WRREC	WriteRecordDone (Page 174)	OnDataRecordWrite (Page 206)

Flowchart

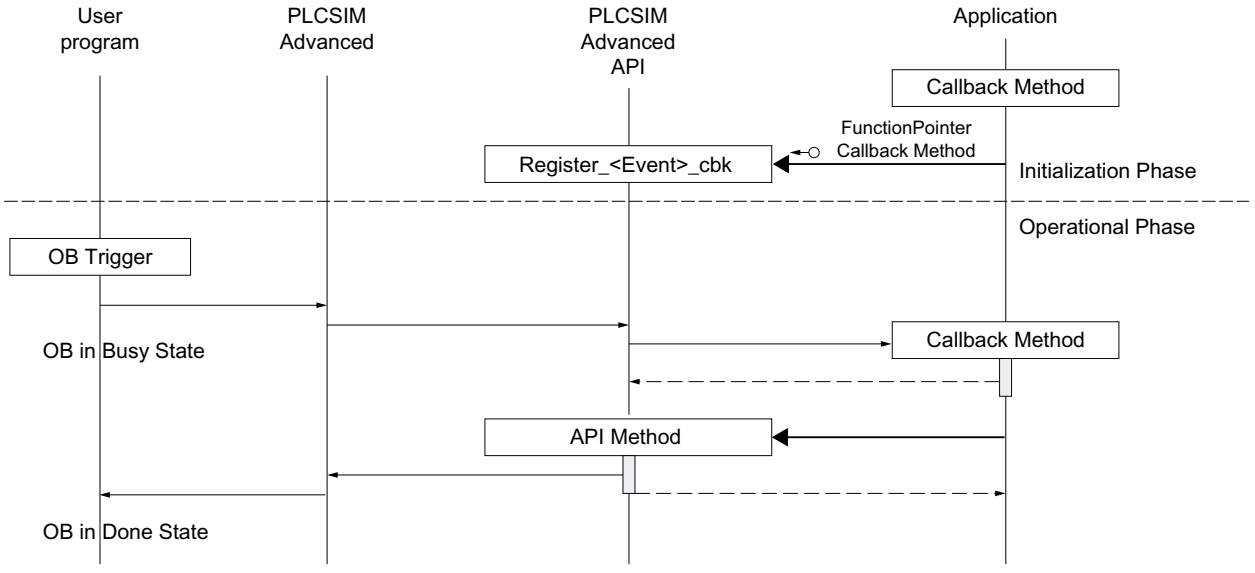


Figure 8-3 Read and write operations flowchart

API methods and associated events

Events which are triggered by the I/O modules and associated API methods:

Table 8-178 API methods and associated events

OB	Name	API methods for triggering the OB (query)	API event after OB execution (alarm)
82	Diagnostic error Interrupt	AlarmNotification (Page 175)	OnAlarmNotificationDone (Page 208)
4x	Hardware Interrupt	ProcessEvent (Page 178)	OnProcessEventDone (Page 209)
83	Pull or Plug of module	PullOrPlugEvent (Page 179)	OnPullOrPlugEventDone (Page 210)
55	Status	StatusEvent (Page 181)	OnStatusEventDone (Page 211)
57	Profile	ProfileEvent (Page 182)	OnProfileEventDone (Page 212)
56	Update	UpdateEvent (Page 183)	OnUpdateEventDone (Page 213)
86	Rack or station failure	RackOrStationFaultEvent (Page 184)	OnRackOrStationFaultEventDone (Page 214)

Flowchart

Flowchart for the simulation of events which are triggered by the I/O modules.

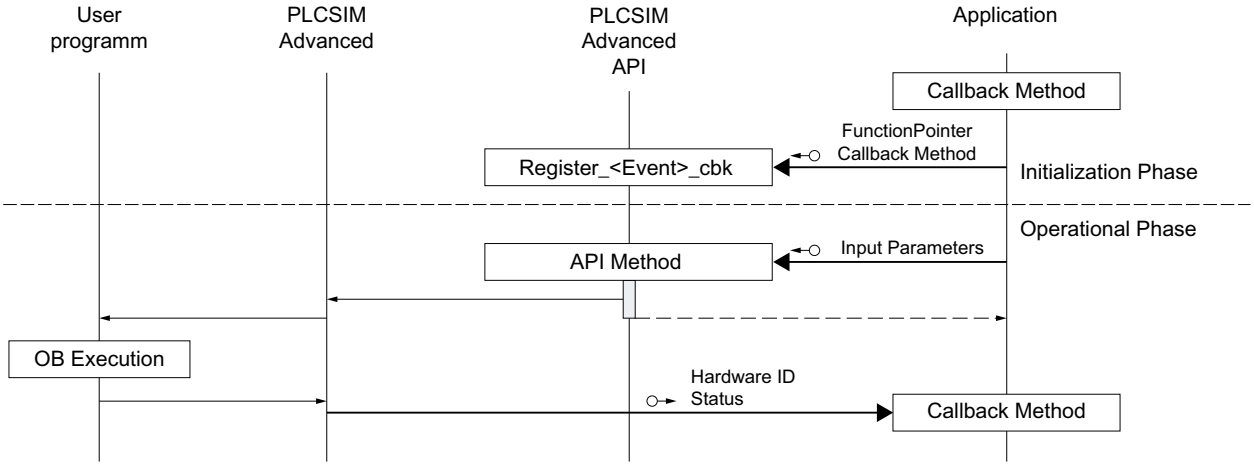


Figure 8-4 Flowchart for the simulation of events

8.10.2 ReadRecordDone/WriteRecordDone

ReadRecordDone()

With this API method, the simulation of an I/O module signals to the CPU that the asynchronous reading of a data record has been completed. The simulation hereby makes the read information available.

Table 8-179 ReadRecordDone() - Native C++

Syntax	`ERuntimeErrorCode ReadRecordDone(SDataRecordInfo in_RecordInfo, BYTE* in_Data, UINT32 in_Status);`	
Parameters	• `SDataRecordInfo in_RecordInfo:` Structure which contains the data record information. See SDataRecordInfo (Page 251). • `BYTE* in_Data:` Byte array of the read data record with the length defined by `DataSize` in the structure `SDataRecordInfo`. • `UINT32 in_Status:` Status of the job execution	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_DOES_NOT_EXIST`	No hardware identifier of the module.
	`SREC_INDEX_OUT_OF_RANGE`	The byte array of the read data record exceeds the length `DDATARECORD_MAX_SIZE` = 64000.
	`SREC_TIMEOUT`	The function does not return on time.

Table 8-180 ReadRecordDone() - .NET (C#)

Syntax	`void ReadRecordDone(SDataRecordInfo in_RecordInfo, BYTE[] in_Data, UInt32 in_Status);`	
Parameters	• `SDataRecordInfo in_RecordInfo:` Structure which contains the data record information. See SDataRecordInfo (Page 251). • `BYTE[] in_Data:` Byte array of the read data record with the length defined by `DataSize` in the structure `SDataRecordInfo`. • `UInt32 in_Status:` Status of the job execution	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.DoesNotExist`	No hardware identifier of the module.
	`ERuntimeErrorCode.IndexOutOfRange`	The byte array of the read data record exceeds the length `DataRecordMaxSize` = 64000.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.

WriteRecordDone()

With this API method, the simulation of an I/O module signals to the CPU that the asynchronous writing of a data record has been completed.

Table 8-181 WriteRecordDone() - Native C++

Syntax	`ERuntimeErrorCode WriteRecordDone(SDataRecordInfo in_RecordInfo, UINT32 in_Status);`	
Parameters	• `SDataRecordInfo in_RecordInfo:` Structure which contains the data record information. See SDataRecordInfo (Page 251). • `UINT32 in_Status:` Status of the job execution	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_DOES_NOT_EXIST`	No hardware identifier of the module.
	`SREC_TIMEOUT`	The function does not return on time.

Table 8-182 WriteRecordDone() - .NET (C#)

Syntax	`void WriteRecordDone(SDataRecordInfo in_RecordInfo, UInt32 in_Status);`	
Parameters	• `SDataRecordInfo in_RecordInfo:` Structure which contains the data record information. See SDataRecordInfo (Page 251). • `UInt32 in_Status:` Status of the job execution	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.DoesNotExist`	No hardware identifier of the module.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.

NOTE

Return values of a written data record

The return values of the status of a written data record in S7-PLCSIM Advanced can differ from the return values of a real hardware CPU.

8.10.3 AlarmNotification

AlarmNotification()

This function triggers diagnostic alarms according to the PROFINET standard.

Each call of this function calls the diagnostic interrupt OB (OB 82) once, regardless of the number and the severity level of the transferred diagnostic entries.

Table 8-183 AlarmNotification() - Native C++

Syntax	`ERuntimeErrorCode AlarmNotification(UINT16 in_HardwareIdentifier, UINT16 in_ModuleState, UINT16 in_NumberOfDiagnosisEvents, SDiagExtChannelDescription* in_ArrayOfDiagnosisEvents, UINT16* out_SequenceNumber);`	
Parameters	• `UINT16 in_HardwareIdentifier:` The hardware identifier of the module or submodule which sends the diagnostics entry. The identifier must belong to a hardware component in the currently loaded project. • `UINT16 in_ModuleState:` Module status. The following statuses are valid: DMODULE_STATE_OK = 0, DMODULE_STATE_ERROR = 1, DMODULE_STATE_MAINT_DEMANDED = 2, DMODULE_STATE_MAINT_REQUIRED = 4 The `in_ModuleState` parameter is derived from the sum (ORed) of the severity level in the `SDiagExtChannelDescription` field. If a diagnostic interrupt should be generated for both "Maintenance demanded" as well as "Maintenance required", select "6" as the module status. • `UINT16 in_NumberOfDiagnosisEvents:` Multiple diagnostic entries can be sent to the CPU with a single API call. Valid range: 0 to 16. 0 means that no diagnostics entry should appear for the submodule or the channel. • `SDiagExtChannelDescription* in_ArrayOfDiagnosisEvents:` Pointer to a field with diagnostic entries. The field must match the number of diagnostic entries. It can also be a null pointer. For definitions, see SDiagExtChannelDescription (Page 253). • `UINT16* out_SequenceNumber:` S7-PLCSIM Advanced assigns a unique consecutive number to each interrupt event. According to PROFINET standard the sequence number is 10 bits wide (1 to $7FF_H$). When the highest number is reached the numbering starts again at 1.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_WRONG_MODULE_STATE`	The module is currently unplugged.
	`SREC_DOES_NOT_EXIST`	No hardware identifier of the module.
	`SREC_WRONG_MODULE_TYPE`	The channel number does not exist for the module.
	`SREC_WRONG_ARGUMENT`	The value for the module status is invalid.
	`SREC_TIMEOUT`	The function does not return on time.

Table 8-184 AlarmNotification() - .NET (C#)

Syntax	`void AlarmNotification(ushort in_HardwareIdentifier, ushort in_ModuleState, ushort in_NumberOfDiagnosisEvents, SDiagExtChannelDescription [] in_ArrayOfDiagnosisEvents, Out ushort out_SequenceNumber);`
Parameters	• `ushort in_HardwareIdentifier:` The hardware identifier of the module or submodule which sends the diagnostics entry. The identifier must belong to a hardware component in the currently loaded project. • `ushort in_ModuleState:` Module status. The following statuses are valid: ModuleState.Ok = 0, ModuleState.Error = 1, ModuleState.MaintenanceDemanded = 2, ModuleState.MaintenanceRequired = 4

<table>
<tr><td></td><td colspan="2">The <code>in_ModuleState</code> parameter is derived from the sum (ORed) of the severity level in the <code>SDiagExtChannelDescription</code> field.
If a diagnostic interrupt should be generated for both "Maintenance demanded" as well as "Maintenance required", select "6" as the module status.
• <code>ushort in_NumberOfDiagnosisEvents</code>
Multiple diagnostic entries can be sent to the CPU with a single API call.
Valid range: 0 to 16. 0 means that no diagnostics entry should appear for the submodule or the channel.
• <code>SDiagExtChannelDescription [] in_ArrayOfDiagnosisEvents:</code>
Pointer to a field with diagnostic entries. The field must match the number of diagnostic entries. It can also be a null pointer. For definitions, see SDiagExtChannelDescription (Page 253).
• <code>Out ushort out_SequenceNumber:</code>
S7-PLCSIM Advanced assigns a unique consecutive number to each alarm event.
According to PROFINET standard the sequence number is 10 bits wide (1 to $7FF_H$). When the highest number is reached the numbering starts again at 1.</td></tr>
<tr><td rowspan="8">Exceptions</td><td colspan="2"><code>Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException</code></td></tr>
<tr><td>Runtime error code</td><td>Description</td></tr>
<tr><td><code>ERuntimeErrorCode.InterfaceRemoved</code></td><td>The instance is not registered in Runtime Manager.</td></tr>
<tr><td><code>ERuntimeErrorCode.WrongModuleState</code></td><td>The module is currently unplugged.</td></tr>
<tr><td><code>ERuntimeErrorCode.DoesNotExist</code></td><td>No hardware identifier of the module.</td></tr>
<tr><td><code>ERuntimeErrorCode.WrongArgument</code></td><td>The value for the module status is invalid.</td></tr>
<tr><td><code>ERuntimeErrorCode.WrongModuleType</code></td><td>The channel number does not exist for the module.</td></tr>
<tr><td><code>ERuntimeErrorCode.Timeout</code></td><td>The function does not return on time.</td></tr>
<tr><td>Example</td><td colspan="2">

```
ushort seqNumber;
  var In_ArrayOfDiagnosisEvent = new SDiagExtChannelDescription[] {
   new SDiagExtChannelDescription() {ChannelNumber = 0x8000, ErrorType =
0x0001,
ExtErrorType = 0, Direction = EDiagProperty.Appear,Severity
=EDiagSeverity.MaintDemanded},

   new SDiagExtChannelDescription() {ChannelNumber = 0x8000, ErrorType =
0x0002,
ExtErrorType = 0, Direction = EDiagProperty.Appear,Severity
=EDiagSeverity.Failure},

   new SDiagExtChannelDescription() {ChannelNumber = 0x8000, ErrorType =
0x0003,
ExtErrorType = 0, Direction = EDiagProperty.Appear,Severity
=EDiagSeverity.MaintRequired},

Instance.AlarmNotification(269, 7, 3, In_ArrayOfDiagnosisEvent, out
seqNumber);
//ModuleState parameter is sum of the severities in the
SDiagExtChannelDescription array above: 4+2+1
```

</td></tr>
</table>

8.10.4 ProcessEvent

ProcessEvent()

Process events from central and distributed input modules can be simulated with this function.

Table 8-185 ProcessEvent() - Native C++

Syntax	`ERuntimeErrorCode ProcessEvent(UINT16 in_HardwareIdentifier, UINT16 in_Channel, EProcessEventType in_ProcessEventType, UINT16* out_SequenceNumber);`	
Parameters	• `UINT16 in_HardwareIdentifier:` The hardware identifier of the module or submodule which sends the process event. The identifier must belong to a hardware component in the currently loaded project. • `UINT16 in_Channel:` The channel of the I/O module which sends the process event. • `EProcessEventType in_ProcessEventType:` A value from the list of predefined types of events for S7 modules, see EProcessEventType (Page 272). • `UINT16* out_SequenceNumber:` S7-PLCSIM Advanced assigns a unique consecutive number to each interrupt event. According to PROFINET standard the sequence number is 10 bits wide (1 to $7FF_H$). When the highest number is reached the numbering starts again at 1.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_WRONG_MODULE_STATE`	The module is currently unplugged.
	`SREC_DOES_NOT_EXIST`	No hardware identifier of the module.
	`SREC_NOT_SUPPORTED_BY_MODULE`	The module is not supported by this user action.
	`SREC_TIMEOUT`	The function does not return on time.

Table 8-186 ProcessEvent() - .NET (C#)

Syntax	`void ProcessEvent(ushort in_HardwareIdentifier, ushort in_Channel, EProcessEventType in_ProcessEventType, Out ushort out_SequenceNumber);`	
Parameters	• `ushort in_HardwareIdentifier:` The hardware identifier of the module or submodule which generates the process event. The identifier must belong to a hardware component in the currently loaded project. • `ushort in_Channel:` The channel of the I/O module which generates the process event. • `EProcessEventType in_ProcessEventType:` A value from the list of predefined types of events for S7 modules, see EProcessEventType (Page 272). • `Out ushort out_SequenceNumber:` S7-PLCSIM Advanced assigns a unique consecutive number to each interrupt event. According to PROFINET standard the sequence number is 10 bits wide (1 to $7FF_H$). When the highest number is reached the numbering starts again at 1.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.WrongModuleState`	The module is currently unplugged.

Exceptions	`ERuntimeErrorCode.DoesNotExist`	No hardware identifier of the module.
	`ERuntimeErrorCode.NotSupportedByMod-ule`	The module is not supported by this user action.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.

Priority for hardware interrupt OB

The hardware interrupts triggered via the S7-PLCSIM Advanced API are transmitted in sequence to the user program.

The priority of the assigned hardware interrupt OB determines the sequence of execution only if events occur simultaneously.

8.10.5 PullOrPlugEvent

PullOrPlugEvent()

This function triggers pull/plug events. The interrupt OB (OB 83) "Pull or plug of modules" is executed for these events.

NOTE

Removing and inserting head modules of distributed I/O systems and R1 devices

An actual PLC cannot detect a pull or plug of a head module in a distributed I/O system, except for R1 devices.

In contrast, a simulated PLC in S7-PLCSIM Advanced does allow the removal and insertion of head modules in distributed I/O systems and R1 devices. Your program can use this event for such a case, but it does not correspond to an event that would occur with an actual PLC.

NOTE

PN/PC Coupler devices

With a physical PN/PN Coupler device, you cannot pull or plug the module.
S7-PLCSIM Advanced allows a pull or plug of PN/PN Couplers if Proxy HwID is used as input for pull/plug events.

Table 8-187 PullOrPlugEvent() - Native C++

Syntax	`ERuntimeErrorCode PullOrPlugEvent(UINT16 in_HardwareIdentifier, EPullOrPlugEventType in_PullOrPlugEventType, UINT16* out_SequenceNumber);`	
Parameters	• `UINT16 in_HardwareIdentifier:` The hardware identifier of the module or submodule which generates the pull/plug event. The identifier must belong to a hardware component in the currently loaded project. • `EPullOrPlugEventType in_PullOrPlugEventType:` A value from the list of predefined types of pull/plug events, see EPullOrPlugEventType (Page 272). • `UINT16* out_SequenceNumber:` S7-PLCSIM Advanced assigns a unique consecutive number to each interrupt event. According to PROFINET standard the sequence number is 10 bits wide (1 to $7FF_H$). When the highest number is reached the numbering starts again at 1.	
Return values	Runtime error code	Description

	SREC_OK	The function is successful.
	SREC_INTERFACE_REMOVED	The instance is not registered in Runtime Manager.
	SREC_WRONG_MODULE_STATE	The module is currently unplugged.
	SREC_WRONG_MODULE_TYPE	The wrong module type was selected, e.g. if an onboard I/O of a compact CPU is to be pulled.
	SREC_NOT_SUPPORTED_BY_MODULE	The module is not supported by this user action.
	SREC_DOES_NOT_EXIST	No hardware identifier of the module.
	SREC_TIMEOUT	The function does not return on time.

Table 8-188 PullOrPlugEvent() - .NET (C#)

Syntax	`void PullOrPlugEvent(ushort in_HardwareIdentifier, EPullOrPlugEventType in_PullOrPlugEventType, Out ushort out_SequenceNumber);`	
Parameters	• `ushort in_HardwareIdentifier:` The hardware identifier of the module or submodule which generates the pull/plug event. The identifier must belong to a hardware component in the currently loaded project. • `EPullOrPlugEventType in_PullOrPlugEventType:` A value from the list of predefined types of pull/plug events, see EPullOrPlugEventType (Page 272). • `Out ushort out_SequenceNumber` S7-PLCSIM Advanced assigns a unique consecutive number to each interrupt event. According to PROFINET standard the sequence number is 10 bits wide (1 to $7FF_H$). When the highest number is reached the numbering starts again at 1.	
Return values	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.WrongModuleState`	The module is currently unplugged.
	`ERuntimeErrorCode.WrongModuleType`	The wrong module type was selected, e.g. if an onboard I/O of a compact CPU is to be pulled.
	`ERuntimeErrorCode.DoesNotExist`	No hardware identifier of the module.
	`ERuntimeErrorCode.NotSupportedByMod-ule`	The module is not supported by this user action.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.

8.10.6 StatusEvent

StatusEvent()

This function is used to trigger the status event OB (OB 55). Status events are only supported for modules in a distributed I/O system.

Table 8-189 StatusEvent() - Native C++

Syntax	`ERuntimeErrorCode StatusEvent(UINT16 in_HardwareIdentifier, UINT16 in_Specifier);`	
Parameters	• `UINT16 in_HardwareIdentifier:` The hardware identifier of the module that generates the status event. The identifier must belong to a hardware component in the currently loaded project. • `UINT16 in_Specifier:` The parameter is transferred to the interrupt frame as interrupt specifier. It is available as input parameter of the OB 55 call.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_WRONG_MODULE_STATE`	The module is currently unplugged.
	`SREC_NOT_SUPPORTED_BY_MODULE`	The module is not supported by this user action.
	`SREC_DOES_NOT_EXIST`	No hardware identifier of the module.
	`SREC_TIMEOUT`	The function does not return on time.

Table 8-190 StatusEvent() - .NET (C#)

Syntax	`void StatusEvent(ushort in_HardwareIdentifier, ushort in_Specifier);`	
Parameters	• `ushort in_HardwareIdentifier:` The hardware identifier of the module that generates the status event. The identifier must belong to a hardware component in the currently loaded project. • `ushort in_Specifier:` The parameter is transferred to the interrupt frame as interrupt specifier. It is available as input parameter of the OB 55 call.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.WrongModuleState`	The module is currently unplugged.
	`ERuntimeErrorCode.DoesNotExist`	No hardware identifier of the module.
	`ERuntimeErrorCode.NotSupportedByMod-ule`	The module is not supported by this user action.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.

8.10.7 ProfileEvent

ProfileEvent()

This function is used to trigger the Profile event OB (OB 57). Profile events are only supported for modules in a distributed I/O system.

Table 8-191 ProfileEvent() - Native C++

Syntax	`ERuntimeErrorCode ProfileEvent(UINT16 in_HardwareIdentifier, UINT16 in_Specifier);`	
Parameters	• `UINT16 in_HardwareIdentifier:` The hardware identifier of the module that generates the profile event. The identifier must belong to a hardware component in the currently loaded project. • `UINT16 in_Specifier:` The parameter is transferred to the interrupt frame as interrupt specifier. It is available as input parameter of the OB 57 call.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_WRONG_MODULE_STATE`	The module is currently unplugged.
	`SREC_NOT_SUPPORTED_BY_MODULE`	The module is not supported by this user action.
	`SREC_DOES_NOT_EXIST`	No hardware identifier of the module.
	`SREC_TIMEOUT`	The function does not return on time.

Table 8-192 ProfileEvent() - .NET (C#)

Syntax	`void ProfileEvent(ushort in_HardwareIdentifier, ushort in_Specifier);`	
Parameters	• `ushort in_HardwareIdentifier:` The hardware identifier of the module that generates the profile event. The identifier must belong to a hardware component in the currently loaded project. • `ushort in_Specifier:` The parameter is transferred to the interrupt frame as interrupt specifier. It is available as input parameter of the OB 57 call.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.WrongModuleState`	The module is currently unplugged.
	`ERuntimeErrorCode.DoesNotExist`	No hardware identifier of the module.
	`ERuntimeErrorCode.NotSupportedByMod-ule`	The module is not supported by this user action.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.

8.10.8 UpdateEvent

UpdateEvent()

This function is used to trigger the Update event OB (OB 56). Update events are only supported for modules in a distributed I/O system.

Table 8-193 UpdateEvent() - Native C++

Syntax	`ERuntimeErrorCode UpdateEvent(UINT16 in_HardwareIdentifier, UINT16 in_Specifier);`	
Parameters	• `UINT16 in_HardwareIdentifier:` The hardware identifier of the module that triggers the update event. The identifier must belong to a hardware component in the currently loaded project. • `UINT16 in_Specifier:` The parameter is transferred to the interrupt frame as interrupt specifier. It is available as input parameter of the OB 56 call.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_WRONG_MODULE_STATE`	The module is currently unplugged.
	`SREC_NOT_SUPPORTED_BY_MODULE`	The module is not supported by this user action.
	`SREC_DOES_NOT_EXIST`	No hardware identifier of the module.
	`SREC_TIMEOUT`	The function does not return on time.

Table 8-194 UpdateEvent() - .NET (C#)

Syntax	`void UpdateEvent(ushort in_HardwareIdentifier, ushort in_Specifier);`	
Parameters	• `ushort in_HardwareIdentifier:` The hardware identifier of the module that triggers the update event. The identifier must belong to a hardware component in the currently loaded project. • `ushort in_Specifier:` The parameter is transferred to the interrupt frame as interrupt specifier. It is available as input parameter of the OB 56 call.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.WrongModuleState`	The module is currently unplugged.
	`ERuntimeErrorCode.DoesNotExist`	No hardware identifier of the module.
	`ERuntimeErrorCode.NotSupportedByMod-ule`	The module is not supported by this user action.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.

8.10.9 GetConfiguredProcessEvent

GetConfiguredProcessEvents()

With this API method, the process events configured in the TIA Portal can be read out during runtime.

If no process events are present, `SREC_OK` is returned. The value for `EventsCount` is then 0.

Table 8-195 GetConfiguredProcessEvents() - Native C++

Syntax	`ERuntimeErrorCode GetConfiguredProcessEvents(UINT16* out_EventsCount,);`	
Parameters	• `SConfiguredProcessEvents* inout_ProcessEvents:` Pointer or reference to a user-defined memory which contains the field with the downloaded configured process events. The structure SConfiguredProcessEvents (Page 252) contains information about these process events. • `UINT16* out_EventsCount:` Pointer or reference to a tag which contains the number of configured process events.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.

Table 8-196 GetConfiguredProcessEvents() - .NET (C#)

Syntax	`SConfiguredProcessEvents [] GetConfiguredProcessEvents();`	
Parameters	None	
Return values	Field with configured process events and field size provide the number of configured process events.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.

8.10.10 RackOrStationFaultEvent

Description

This function is used to trigger the RackOrStationFault event OB (OB 86). These events are only supported for distributed devices.

Table 8-197 RackOrStationFaultEvent() - Native C++

Syntax	`ERuntimeErrorCode RackOrStationFaultEvent(UINT16 in_HardwareIdentifier, ERackOrStationFaultType in_EventType);`	
Parameter	• `UINT16 in_HardwareIdentifier:` The hardware identifier of the device that sends the event. Use the hardware identifier of the Hw_Device type. • `ERackOrStationFaultType in_EventType:` A value from the list of predefined types of events, see ERackOrStationFaultType (Page 274).	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.

Return values	SREC_WRONG_MODULE_TYPE	The specified HW identifier is not that of a distributed device.
	SREC_WRONG_MODULE_STATE	The device with the specified HW identifier already reports the status Fault/Return.
	SREC_DOES_NOT_EXIST	The specified HW identifier of the device does not exist.
	SREC_TIMEOUT	The function does not return on time.

Table 8-198 RackOrStationFaultEvent() - .NET (C#)

Syntax	`void RackOrStationFaultEvent(ushort in_HardwareIdentifier, ERackOrStationFaultType in_EventType);`	
Parameter	• `ushort in_HardwareIdentifier:` The hardware identifier of the device that sends the event. Use the hardware identifier of the Hw_Device type. • `ERackOrStationFaultType in_EventType:` A value from the list of predefined types of events, see ERackOrStationFaultType (Page 274).	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.DoesNotExist`	The specified HW identifier of the device does not exist.
	`ERuntimeErrorCode.WrongModuleType`	The specified HW identifier is not that of a distributed device.
	`ERuntimeErrorCode.WrongModuleState`	The device with the specified HW identifier already reports the status Fault/Return.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.

More information

You can find more information on the HW identifier in the STEP 7 online help.

8.11 Events for IInstances

8.11.1 Events for operating state and cycle control

The following events are triggered for the IInstances interface:

Table 8-199 Events for IInstances

Event	Cause
OnOperatingStateChanged (Page 191)	The operating state of the simulated PLC instance has changed.
OnLedChanged (Page 195)	The LED display of the simulated PLC instance has changed.
OnSoftwareConfigurationChanged (Page 197)	The configuration of the simulated PLC instance changes: • During power up from the Virtual SIMATIC Memory Card • At the start of a download When this event is triggered, the stored tag list is reset.

Event	Cause
OnSoftwareConfigurationChanged (-Page 197)	The configuration of the simulated PLC instance has changed: • After power up from the Virtual SIMATIC Memory Card • At the end of a download • When the IP address changes
OnSyncPointReached (Page 201)	The simulated PLC instance has reached a synchronization point. If the simulated PLC instance is being operated in `Default` mode, the `SendSyncEventInDefaultMode` flag must be set to receive the event. See SendSyncEventInDefaultMode (Page 163).
OnHardwareConfigChanged (Page 203)	When the hardware configuration of the simulated PLC instance has changed, the event is triggered at the following times: • After power up from the Virtual SIMATIC Memory Card • During the Hardware Config download

8.11.1.1 Synchronize simulation partner

Synchronize simulation partner cycle-controlled

SingleStep operating modes

The SingleStep modes of operation of the simulated PLC instance are used to synchronize multiple simulation partners (clients) using synchronization points. The operating modes define the synchronization point at which the instance changes to the freeze state and sends the `OnSyncPointReached` event.

Table 8-200 Cycle-controlled operating modes (SingleStep)

Operating mode[1]	Synchronization point		Minimum cycle time[2]	Send clock "Bus"[3]
	Cycle control point	Before reading in the process image partition		
	"C"	"P"	"T"	
`SingleStep_C`[4]	✓			
`SingleStep_P`		✓		
`SingleStep_CP`[4]	✓	✓		
`SingleStep_CT`[4]	✓		✓	
`SingleStep_CPT`[4]	✓	✓	✓	
`SingleStep_Bus`	✓			✓

[1] A change from one SingleStep operating mode to another during operation would result in the input parameter `TimeSinceSameSyncPoint_ns / TimeSinceAnySyncPoint_ns` being set to 0 in the first SyncPointReached callback event in the new operating mode. We recommend that the SingleStep operating mode not be changed during operation.

[2] In addition, the minimum scan cycle time of OB 1 is overwritten in this operating mode. When you define a minimum cycle time of 200 ms, the minimum distance between two cycle control points is 200 virtual milliseconds. The default setting is 100 ms.

[3] Send clock of the I/O system (PROFIBUS or PROFINET) that is to be used for the cycle-controlled synchronization of the simulated PLC instance. You set the send clock of the respective I/O system in the STEP 7 properties, for example, in the properties of the PROFINET interface of the CPU (Advanced options > Real time settings > I/O communication > Send clock).

[4] During the execution of these operating modes, the input parameter of the first callback event returns the value 0 for TimeSinceSameSyncPoint_ns / TimeSinceAnySyncPoint_ns. Subsequent callback events for TimeSinceSameSyncPoint_ns / TimeSinceAnySyncPoint_ns then return values other than 0.

API functions / events

- GetOverwrittenMinimalCycleTime_ns() (Page 163)
- SetOverwrittenMinimalCycleTime_ns() (Page 163)
- OverwrittenMinimalCycleTime_ns { get; set; } (Page 163)
- RunToNextSyncPoint() (Page 201)
- OnSyncPointReached (Page 201)
- EventCallback_II_SREC_ST_UINT32_UINT32_EPET_UINT32 (Page 227) / Delegate_II_EREC_DT_UINT32_INT64_INT64_UINT32 (Page 232)

Conditions for the `SingleStep_Bus` operating mode:

The following conditions must be met for the `SingleStep_Bus` operating mode:

- You must have configured outputs on the controller.
- To synchronize the process image of the outputs, you must call the SYNC_PO instruction in the isochronous mode interrupt OB.

NOTE

In addition to the send clock of the bus, the cycle control point is also used as a synchronization point. In the cycle control point, the transferred ID has the value 0xFFFF. The in_parameters "in_TimeSinceSameSyncPoint_ns" and "in_TimeSinceAnySyncPoint_ns" have the value 0.

A synchronization of OBs and process image partitions is only possible if the OBs call the instructions "SYNC_PI" and "SYNC_PO". To use process image partition 1 (PIP 1) for an F-runtime group, configure the "Pre/Post processing of the F-runtime group" in the Safety Administration editor.

To configure the preprocessing and postprocessing of the F-runtime group you need new standard FCs for "SYNC_PI" and "SYNC_PO" with the following properties:

- Name of the FC: SYNC_PI

  #RET_VAL_PI := SYNC_PI(PART := "PIP 1", FLADDR => #FLADDR_PI);

- Name of the FC: SYNC_PO

  #RET_VAL_PO := SYNC_PO(PART := "PIP 1", FLADDR => #FLADDR_PO);

To synchronize the process image partition 1 (PIP 1), add the FCs in the Safety Administration editor under "Pre/Post processing of the F-runtime group".

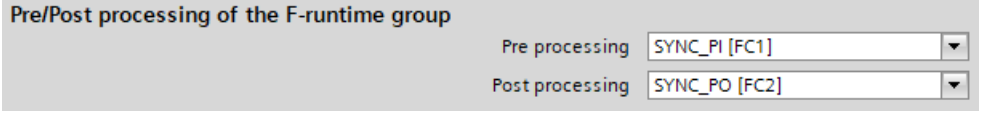


Figure 8-5 Safety Administration Editor

After compiling these changes, process image partition 1 (PIP 1) is also available for the fail-safe organization block (FOB).

Notes regarding the SingleStep_P, SingleStep_Bus, and SingleStep_CP operating modes

NOTE

Loss of a OnSyncPointReached event when downloading a clock-synchronous OB in RUN mode.

If you are registered for the OnSyncPointReached event in the SingleStep_P and SingleStep_Bus operating modes, the simulated PLC instance does not send this event to the API client when downloading a clock-synchronous OB.

Solution:

1. Use the OnSoftwareConfigurationChanged event to determine the start and end of a download
2. Register and log off the OnSyncPointReached event, or handle the loss of the OnSyncPointReached event accordingly.

NOTE

Note on SingleStep_CP operating mode

Downloading software changes in RUN mode.

If you perform a software download in RUN mode in SingleStep_CP operating mode, the download may not be possible and the simulation may stop. An overload during the download in RUN may be the reason for this.

Solution:

1. Use the "Virtual Time Scaling" function and select a low scaling factor (e.g. scaling factor of 0.25).
2. Provide sufficient resources to increase the runtime performance and the S7-PLCSIM Advanced communication performance.

Terminating the freeze state

The RunToNextSyncPoint() function cancels the freeze state and induces the simulated PLC instance to continue running until the next synchronization point.

Switching to the Default operating mode also terminates the freeze state.

Example

The figure schematically shows the sequence in the SingleStep_CP operating mode.

In addition to the OnSyncPointReached event the simulated PLC instance also sends the virtual time since the last synchronization point of the same process image partition ID or of any process image partition ID has been reached (TimeSinceSameSyncPoint_ns / TimeSinceAnySyncPoint_ns).

The `RunToNextSyncPoint()` function cancels the freeze state.

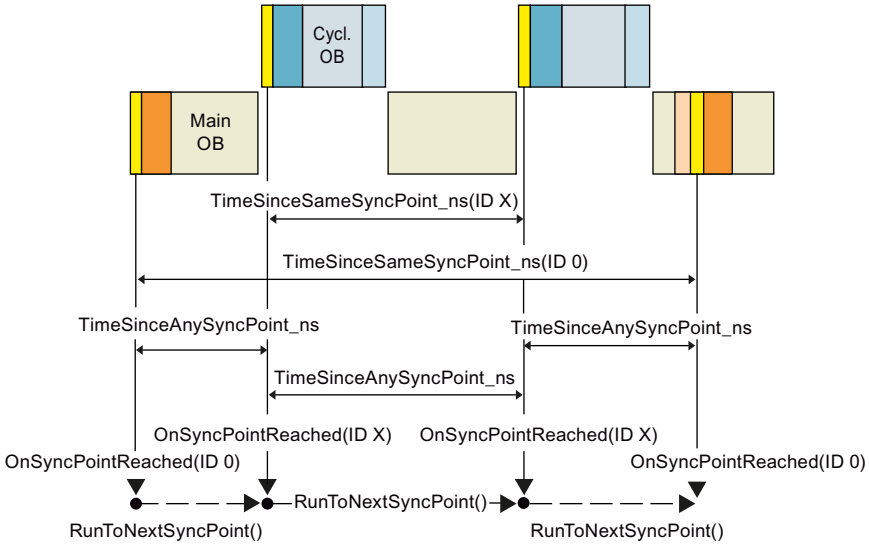


Figure 8-6 Example: Sequence in the SingleStep_CP operating mode

Changing the settings in the watch table

NOTE

Selecting triggers for monitoring of tags in the SingleStep operating modes

In TIA Portal the watch table in basic mode shows the values for outputs and bit memories **before** the processing.

To display the tag values **after** processing, proceed as follows:

1. Select the extended mode for the watch table.
2. In the "Monitor with trigger" column, select "End of cycle, permanent".

Synchronize simulation partner time-controlled

TimespanSynchronized operating modes

Several simulation partners (clients) are synchronized time-controlled with the TimespanSynchronized operating modes of the simulated PLC instance. The operating modes define the synchronization point at which the instance changes to the freeze state and sends the `OnSyncPointReached` event.

Table 8-201 Time-controlled operating modes (TimespanSynchronized)

Operating mode	Synchronization point	
	Cycle control point	Before reading in the process image partition
	"C"	"P"
`TimespanSynchronized_C`	✓	
`TimespanSynchronized_CP`	✓	✓
`TimespanSynchronized_P`		✓

API functions / events

- Settings for cycle control (Page 163)
- StartProcessing() (Page 163)
- OnSyncPointReached (Page 201)

Terminating the freeze state

The `StartProcessing(t)` function cancels the freeze state and induces the simulated PLC instance to continue running at least as long as required (on the basis of the virtual time) before it changes back to the freeze state at the next synchronization point.

Switching to the Default operating mode also terminates the freeze state.

Example

The figure schematically shows the sequence in the `TimespanSynchronized_CP` operating mode.

In addition to the `OnSyncPointReached` event the simulated PLC instance also sends the runtime since the last call of the `StartProcessing`(t) (`TimeSinceSameSyncPoint_ns` / `TimeSinceAnySyncPoint_ns`) function.

The `StartProcessing()` function cancels the freeze state.

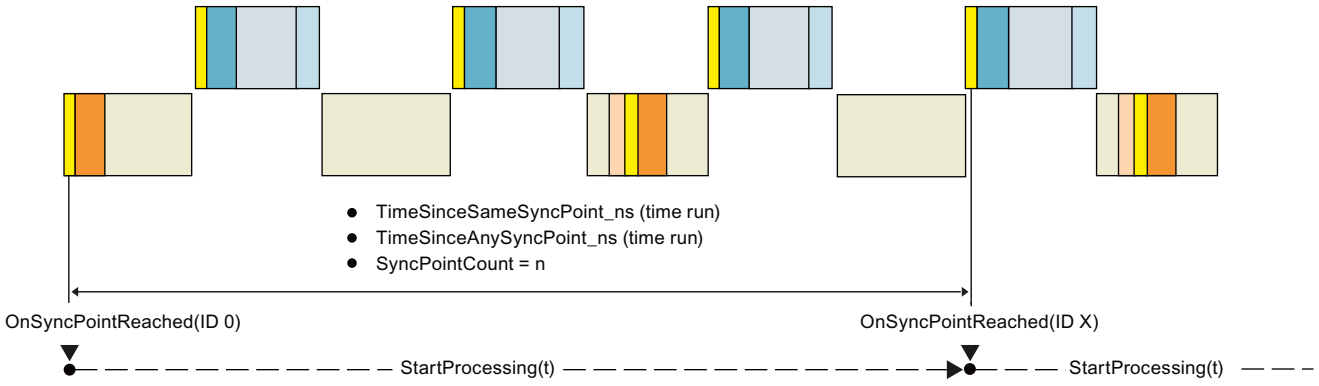


Figure 8-7 Example: Sequence in the TimespanSynchronized_CP operating mode

Description

At least two clients are synchronized on the basis of a virtual period for the time-controlled operating modes. A client can be an instance of a simulated PLC instance or an application that uses the Runtime API (API client). The synchronization must be performed by a synchronization master.

The synchronization master instructs a client to run for a specific period. The time period is specified by the master in nanoseconds. The client then runs for the expected period before the client goes into the freeze state at the next synchronization point. Before switching to the freeze state, the client sends the master the exact amount of time that he currently needed. Thereafter, the master signals the next client to catch up.

API client as master

The API client as master signals each client when it should start. The master receives events from every client when they occur.

An API client can only "time manage" instances of a simulated PLC instance. The API client does not receive events from other API clients. It cannot send messages to other API clients.

8.11.1.2 OnOperatingStateChanged events

OnOperatingStateChanged

Registers or unregisters an event handler method.

Table 8-202 OnOperatingStateChanged - .NET (C#)

Syntax	`event Delegate_II_EREC_DT_EOS_EOS OnOperatingStateChanged;`
Parameters	None. See Delegate_II_EREC_DT_EOS_EOS (Page 231).
Return values	None
Exceptions	None
Note	The event handler method runs in a separate thread.

RegisterOnOperatingStateChangedCallback()

When the event occurs, the registered callback function is called. Multiple callback functions can be registered for the event.

Table 8-203 RegisterOnOperatingStateChangedCallback() - Native C++

Syntax	`void RegisterOnOperatingStateChangedCallback(EventCallback_II_SREC_ST_SROS_SROS in_CallbackFunction);`
Parameters	• `EventCallback_II_SREC_ST_SROS_SROS in_CallbackFunction:` A callback function that subscribes to the event. See EventCallback_II_SREC_ST_SROS_SROS (Page 223).
Return values	None
Note	The callback function runs in a separate thread.

RegisterOnOperatingStateChangedEvent()

When the event occurs, the registered event object is set to the signaled state. Only one event object can be registered for the event. Registering a new event object causes the previous event object to be deleted.

Table 8-204 RegisterOnOperatingStateChangedEvent() - Native C++

Syntax	`void RegisterOnOperatingStateChangedEvent();` `void RegisterOnOperatingStateChangedEvent(HANDLE* in_Event);`
Parameters	• `None:` An internal event object is registered. • `HANDLE* in_Event:` A handle for a user-specific event object. The event object is registered.
Return values	None

Example C++	`// Thread 1 ----------`
	`ISimulationRuntimeManager * api = NULL;`
	`ERuntimeErrorCode result = InitializeApi(&api);`
	
	`IInstance* psa = NULL;`
	`if (result == SREC_OK)`
	`{`
	` result = api->RegisterInstance(&psa);`
	`}`
	
	`// Register the internal event object`
	`psa->`**`RegisterOnOperatingStateChangedEvent`**`();`
	
	`// Thread 2 ----------`
	`while (condition)`
	`{`
	`// Wait for the event to be set (timeout after 10s)`
	`bool isEventSet = psa->WaitForOnOperatingStateChangedEvent(10000);`
	`if (isEventSet)`
	`{`
	`// Do Something`
	`…`
	`}`
	`}`

Example C++	`// Thread 1 ----------------------------------------------------`
	`ISimulationRuntimeManager * api = NULL;`
	`ERuntimeErrorCode result = InitializeApi(&api);`
	
	`IInstance* psa = NULL;`
	`if (result == SREC_OK)`
	`{`
	` result = api->RegisterInstance(&psa);`
	`}`
	
	`// Create an event object`
	`HANDLE eventHandle = CreateEvent(NULL, FALSE, FALSE, NULL);`
	
	`// Register the user created event object`
	`psa->`**`RegisterOnOperatingStateChangedEvent`**`(&eventHandle);`
	
	`// Do Something`
	`…`
	`// Clean up the handle`
	`CloseHandle(eventHandle);`
	
	`// Thread 2 ----------------------------------------------------`
	`while (condition)`
	`{`
	`// Wait for the event to be set //OR:`
	`WaitForSingleObject(eventHandle, INFINITE); //psa-`
	`>WaitForOnOperatingStateChangedEvent();`
	
	`// Do Something`
	`…`
	`}`

Table 8-205 RegisterOnOperatingStateChangedEvent() – .NET (C#)

Syntax	`void RegisterOnOperatingStateChangedEvent();`
Parameters	None
Return values	None

UnregisterOnOperatingStateChangedCallback()

Unregisters the callback function.

Table 8-206 UnregisterOnOperatingStateChangedCallback() - Native C++

Syntax	`void UnregisterOnOperatingStateChangedCallback(EventCallback_II_SREC_ST_SROS-_SROS in_CallbackFunction = nullptr);`
Parameters	Optional parameter of the same type as for the Register*() function If you use the parameter, it uninstalls the specific function; otherwise, the last registered callback function is unregistered.
Return values	None

UnregisterOnOperatingStateChangedEvent()

Unregisters the event object.

Table 8-207 UnregisterOnOperatingStateChangedEvent() - Native C++

Syntax	`void UnregisterOnOperatingStateChangedEvent();`
Parameters	None
Return values	None

Table 8-208 UnregisterOnOperatingStateChangedEvent() - .NET (C#)

Syntax	`void UnregisterOnOperatingStateChangedEvent();`
Parameters	None
Return values	None

WaitForOnOperatingStateChangedEvent()

The function blocks the program until the registered event object is in the signaled state or the timeout interval is exceeded.

Table 8-209 WaitForOnOperatingStateChangedEvent() - Native C++

Syntax	`bool WaitForOnOperatingStateChangedEvent();` `bool WaitForOnOperatingStateChangedEvent(UINT32 in_Time_ms);`
Parameters	• `None`: The time limit is set to `INFINITE`. • `UINT32 in_Time_ms`: Value for the time limit in milliseconds.
Return values	• `true`: If the event object was set to the signaled state. • `false`: If no event was received during the defined time limit.

Table 8-210 WaitForOnOperatingStateChangedEvent() - .NET (C#)

Syntax	`bool WaitForOnOperatingStateChangedEvent();` `bool WaitForOnOperatingStateChangedEvent(UInt32 in_Time_ms);`
Parameters	• `None`: The time limit is set to `INFINITE`. • `UInt32 in_Time_ms`: Value for the time limit in milliseconds.
Return values	• `true`: If the event object was set to the signaled state. • `false`: If no event was received during the defined time limit.

8.11.1.3 OnLedChanged events

OnLedChanged

Registers or unregisters an event handler method.

Table 8-211 OnLedChanged - .NET (C#)

Syntax	`event Delegate_II_EREC_DT_ELT_ELM OnLedChanged;`
Parameters	None. See Delegate_II_EREC_DT_ELT_ELM (Page 232).
Return values	None
Exceptions	None
Note	The event handler method runs in a separate thread.

RegisterOnLedChangedCallback()

When the event occurs, the registered callback function is called. Multiple callback functions can be registered for the event.

Table 8-212 RegisterOnLedChangedCallback() - Native C++

Syntax	`void RegisterOnLedChangedCallback(EventCallback_II_SREC_ST_SRLT_SRLM in_CallbackFunction);`
Parameters	• `EventCallback_II_SREC_ST_SRLT_SRLM in_CallbackFunction:` A callback function that subscribes to an event. See EventCallback_II_SREC_ST_SRLT_SRLM (Page 225).
Return values	None
Note	The callback function runs in a separate thread.

RegisterOnLedChangedEvent()

When the event occurs, the registered event object is set to the signaled state. Only one event object can be registered for the event. Registering a new event object causes the previous event object to be deleted.

Table 8-213 RegisterOnLedChangedEvent() - Native C++

Syntax	`void RegisterOnLedChangedEvent();` `void RegisterOnLedChangedEvent(HANDLE* in_Event);`
Parameters	• `None:` An internal event object is registered. • `HANDLE* in_Event:` A handle for a user-specific event object. The event object is registered.
Return values	None

Table 8-214 RegisterOnLedChangedEvent() – .NET (C#)

Syntax	`void RegisterOnLedChangedEvent();`
Parameters	None
Return values	None

UnregisterOnLedChangedCallback()

Unregisters the callback function.

Table 8-215 UnregisterOnLedChangedCallback() - Native C++

Syntax	`void UnregisterOnLedChangedCallback();`
Parameters	Optional parameter of the same type as for the Register*() function If you use the parameter, it uninstalls the specific function; otherwise, the last registered callback function is unregistered.
Return values	None

UnregisterOnLedChangedEvent()

Unregisters the event object.

Table 8-216 UnregisterOnLedChangedEvent() - Native C++

Syntax	`void UnregisterOnLedChangedEvent();`
Parameters	None
Return values	None

Table 8-217 UnregisterOnLedChangedEvent() - .NET (C#)

Syntax	`void UnregisterOnLedChangedEvent();`
Parameters	None
Return values	None

WaitForOnLedChangedEvent()

The function blocks the program until the registered event object is in the signaled state or the timeout interval is exceeded.

Table 8-218 WaitForOnLedChangedEvent() - Native C++

Syntax	`bool WaitForOnLedChangedEvent();` `bool WaitForOnLedChangedEvent(UINT32 in_Time_ms);`
Parameters	• `None:` The time limit is set to `INFINITE`. • `UINT32 in_Time_ms:` Value for the time limit in milliseconds.
Return values	• `true:` If the event object was set to the signaled state. • `false:` If no event was received during the defined time limit.

Table 8-219 WaitForOnLedChangedEvent() - .NET (C#)

Syntax	`bool WaitForOnLedChangedEvent();` `bool WaitForOnLedChangedEvent(UInt32 in_Time_ms);`
Parameters	• `None:` The time limit is set to `INFINITE`. • `UInt32 in_Time_ms:` Value for the time limit in milliseconds.
Return values	• `true:` If the event object was set to the signaled state. • `false:` If no event was received during the defined time limit.

8.11.1.4 OnSoftwareConfigurationChanged events

OnSoftwareConfigurationChanged()

Reports software changes in STOP and RUN modes.

Table 8-220 OnSoftwareConfigurationChanged

Syntax	`event Delegate_SRSCC OnSoftwareConfigurationChanged();`
Parameters	IInstance, ERuntimeErrorCode, SYSTEMTIME, ESoftwareConfigChanged
Return values	None
Exceptions	None
Note	The event handler method runs in a separate thread.

RegisterOnSoftwareConfigurationChangedCallback()

When the event occurs, the registered callback function is called. Only one callback function can be registered for the event. The registration of a new callback function causes the previous callback function to be deleted.

Table 8-221 RegisterOnSoftwareConfigurationChangedCallback() - Native C++

Syntax	`void RegisterOnSoftwareConfigurationChangedCallback();`
Parameters	CallbackFunction
Return values	None
Note	The event handler method runs in a separate thread.

RegisterOnSoftwareConfigurationChangedEvent()

When the event occurs, the registered event object is set to the signaled state. Only one event object can be registered for the event. Registration of a new event object causes the previous event object to be deleted.

Table 8-222 RegisterOnSoftwareConfigurationChangedEvent() - Native C++

Syntax	`void RegisterOnSoftwareConfigurationChangedEvent();`
Parameters	• `None` An internal event object is registered. • `HANDLE*` A handle for a user-specific event object. The event object is registered.
Return values	None

UnregisterOnSoftwareConfigurationChangedCallback()

Unregisters the callback function. When the event occurs, no callback function is called.

Table 8-223 UnregisterOnSoftwareConfigurationChangedCallback() - Native C++

Syntax	`void UnregisterOnSoftwareConfigurationChangedCallback();`
Parameters	None
Return values	None

UnregisterOnConfigurationChangedEvent()

Unregisters the event object.

Table 8-224 UnregisterOnConfigurationChangedEvent() - Native C++

Syntax	`void UnregisterOnSoftwareConfigurationChangedEvent();`
Parameters	None
Return values	None

Table 8-225 UnregisterOnConfigurationChangedEvent() - .NET (C#)

Syntax	`void UnregisterOnSoftwareConfigurationChangedEvent();`
Parameters	None
Return values	None

WaitForOnConfigurationChangedEvent()

The function blocks the program until the registered event object is set to the signaled state or the timeout interval is exceeded.

Table 8-226 WaitForOnSoftwareConfigurationChangedEvent() - Native C++

Syntax	`void WaitForOnSoftwareConfigurationChangedEvent();`
Parameters	• `None:` The time limit is set to `INFINITE`. • `UINT32 in_Time_ms:` Value for the time limit in milliseconds.
Return values	• `true:` If the event object was set to the signaled state. • `false:` If no event was received during the defined time limit.
Note	The function blocks the CoSimulation program until a software download is set to the signaled state or the timeout interval is exceeded.

Table 8-227 WaitForOnSoftwareConfigurationChangedEvent() - .NET (C#)

Syntax	`void WaitForOnSoftwareConfigurationChangedEvent();`
Parameters	• `None:` The time limit is set to `INFINITE`. • `UINT32 in_Time_ms:` Value for the time limit in milliseconds.
Return values	• `true:` If the event object was set to the signaled state. • `false:` If no event was received during the defined time limit.
Note	The function blocks the CoSimulation program until a software download is set to the signaled state or the timeout interval is exceeded.

8.11.1.5 OnIPAddressChanged events

RegisterOnIPAddressChangedCallback()

Register function for receiving callback functions when an IP address has changed.

Table 8-228 RegisterOnIPAddressChangedCallback() - Native C++

Syntax	`RegisterOnIPAddressChangedCallback();`
Parameters `in_CallbackFunction`	• `IInstance* in_Sender` Instance object • `ERuntimeErrorCode in_ErrorCode` Result of operation • `SYSTEMTIME in_SystemTime` CPU system time of the event • `UINT8 in_InterfaceId` ID of the CPU interface involved • `const SIPSuite4& in_SIP` Reference of the SIPSuite4 structure
Return values	None
Note	The event handler method runs in a separate thread.

UnregisterOnIPAddressChangedCallback()

Unregister function for receiving callback functions when an IP address has changed.

Table 8-229 UnregisterOnIPAddressChangedCallback() - Native C++

Syntax	`void UnregisterOnIPAddressChangedCallback();`
Parameters `in_CallbackFunction`	Address of the specific function or nullptr for removing the last registered callback function
Return values	None

RegisterOnIPAddressChangedEvent()

Register function for receiving events when an IP address has changed.

Table 8-230 RegisterOnIPAddressChangedEvent() - Native C++

Syntax	`void IInstance::RegisterOnIPAddressChangedEvent()` `void IInstance::RegisterOnIPAddressChangedEvent(HANDLE* in_Event);`
Parameters `in_Event`	HANDLE* (a user-specific event object is registered)
Return values	None

UnregisterOnIPAddressChangedEvent()

Unregister function for receiving events when an IP address has changed.

Table 8-231 UnregisterOnIPAddressChangedEvent() - Native C++

Syntax	`void IInstance::UnregisterOnIPAddressChangedEvent();`
Parameters	None
Return values	None

OnIPAddressChanged()

The event is called when the IP configuration of the CPU interface has changed.

Table 8-232 OnIPAddressChanged() - .NET (C#)

Syntax	`void IInstance::OnIPAddressChanged();`
Parameters	• `IInstance in_Sender` Instance object • `ERuntimeErrorCode in_ErrorCode` Result of operation • `DateTime in_SystemTime` CPU system time of the event • `UINT8 in_InterfaceId` ID of the CPU interface involved • `SIPSuite4 in_SIP` IP address of the interface
Return values	None
Note	The event handler method runs in a separate thread.

WaitForOnIPAddressChangedEvent()

Blocking function up to the change of the IP address.

Table 8-233 WaitForOnIPAddressChangedEvent() - Native C++

Syntax	`bool IInstance::WaitForOnIPAddressChangedEvent();` `bool IInstance::WaitForOnIPAddressChangedEvent(UINT32 in_Time_ms)`
Parameters `in_Time_ms`	Time in milliseconds
Return values	Returns `True` if the change was made within a set period, otherwise `False`
Note	None

Table 8-234 WaitForOnIPAddressChangedEvent() - .NET (C#)

Syntax	`bool IInstance::WaitForOnIPAddressChangedEvent();` `bool IInstance::WaitForOnIPAddressChangedEvent(uint in_Time_ms);`
Parameters `in_Time_ms`	Time in milliseconds
Return values	Returns `True` if the change was made within a set period, otherwise `False`
Note	The event handler method runs in a separate thread.

8.11.1.6 OnSyncPointReached events

OnSyncPointReached

Registers or unregisters an event handler method.

Table 8-235 OnSyncPointReached - .NET (C#)

Syntax	`event Delegate_II_EREC_DT_UINT32_INT64_INT64_UINT32 OnSyncPointReached;`
Parameters	None. See Delegate_II_EREC_DT_UINT32_INT64_INT64_UINT32 (Page 232).
Return values	None
Exceptions	None
Note	The event handler method runs in a separate thread.

RegisterOnSyncPointReachedCallback()

When the event occurs, the registered callback function is called. Multiple callback functions can be registered for the event.

Table 8-236 RegisterOnSyncPointReachedCallback() - Native C++

Syntax	`void RegisterOnSyncPointReachedCallback(EventCallback_II_SREC_ST_UINT32_INT64_INT64_UINT32 in_CallbackFunction);`
Parameters	• `EventCallback_II_SREC_ST_UINT32_INT64_INT64_UINT32 in_CallbackFunction:` A callback function that subscribes to an event. See EventCallback_II_SREC_ST_UINT32_INT64_INT64_UINT32 (Page 224).
Return values	None
Note	The callback function runs in a separate thread.

RegisterOnSyncPointReachedEvent()

When the event occurs, the registered event object is set to the signaled state. Only one event object can be registered for the event. Registering a new event object causes the previous event object to be deleted.

Table 8-237 RegisterOnSyncPointReachedEvent() - Native C++

Syntax	`void RegisterOnSyncPointReachedEvent();` `void RegisterOnSyncPointReachedEvent(HANDLE* in_Event);`
Parameters	• `None`: An internal event object is registered. • `HANDLE* in_Event`: A handle for a user-specific event object. The event object is registered.
Return values	None

Table 8-238 RegisterOnSyncPointReachedEvent() – .NET (C#)

Syntax	`RegisterOnSyncPointReachedEvent() – .NET (C#)`
Parameters	None
Return values	None

UnregisterOnSyncPointReachedCallback()

Unregisters the callback function.

Table 8-239 UnregisterOnSyncPointReachedCallback() - Native C++

Syntax	`void UnregisterOnSyncPointReachedCallback();`
Parameters	Optional parameter of the same type as for the Register*() function If you use the parameter, it uninstalls the specific function; otherwise, the last registered callback function is unregistered.
Return values	None

UnregisterOnSyncPointReachedEvent()

Unregisters the event object.

Table 8-240 UnregisterOnSyncPointReachedEvent() - Native C++

Syntax	`void UnregisterOnSyncPointReachedEvent();`
Parameters	None
Return values	None

Table 8-241 UnregisterOnSyncPointReachedEvent() - .NET (C#)

Syntax	`void UnregisterOnSyncPointReachedEvent();`
Parameters	None
Return values	None

WaitForOnSyncPointReachedEvent()

The function blocks the program until the registered event object is in the signaled state or the timeout interval is exceeded.

Table 8-242 WaitForOnSyncPointReachedEvent() - Native C++

Syntax	`SOnSyncPointReachedResult WaitForOnSyncPointReachedEvent();` `SOnSyncPointReachedResult WaitForOnSyncPointReachedEvent(UINT32 in_Time_ms);`
Parameters	• `None`: The time limit is set to `INFINITE`. • `UINT32 in_Time_ms`: Value for the time limit in milliseconds.
Return values	• `SOnSyncPointReachedResult`: A structure that supplies information about the event. See SOnSyncPointReachedResult (Page 249).

Table 8-243 WaitForOnSyncPointReachedEvent() - .NET (C#)

Syntax	`SOnSyncPointReachedResult WaitForOnSyncPointReachedEvent();` `SOnSyncPointReachedResult WaitForOnSyncPointReachedEvent(UInt32 in_Time_ms);`
Parameters	• `None`: The time limit is set to `INFINITE`. • `UInt32 in_Time_ms`: Value for the time limit in milliseconds.
Return values	• `SOnSyncPointReachedResult`: A structure that supplies information about the event. See SOnSyncPointReachedResult (Page 249).

8.11.1.7 OnHardwareConfigChanged events

OnHardwareConfigChanged

Registers or unregisters an event handler method.

Table 8-244 OnHardwareConfigChanged - .NET (C#)

Syntax	`event Delegate_II_EREC_DT OnHardwareConfigChanged;`
Parameters	None; see Delegate_II_EREC_DT
Return values	None
Exceptions	None
Note	The event handler method runs in a separate thread.

RegisterOnHardwareConfigChangedCallback()

When the event occurs, the registered callback function is called. Multiple callback functions can be registered for the event.

Table 8-245 Table: RegisterOnHardwareConfigChangedCallback() – Native C++

Syntax	`void RegisterOnHardwareConfigChangedCallback(EventCallback_II_SREC_ST in_CallbackFunction);`
Parameters	`EventCallback_II_SREC_ST in_CallbackFunction` A callback function that subscribes to an event; see EventCallback_II_SREC_ST()
Return values	None
Note	The callback function runs in a separate thread.

RegisterOnHardwareConfigChangedEvent()

When the event occurs, the registered event object is set to the signaled state. Only one event object can be registered for the event. Registration of a new event object causes the previous event object to be deleted.

Table 8-246 RegisterOnHardwareConfigChangedEvent() – Native C++

Syntax	`void RegisterOnHardwareConfigChangedEvent();` `void RegisterOnHardwareConfigChangedEvent(Handle* in_Event);`
Parameters	• `None`: An internal event object is registered. • `Handle* in_Event` A handle for a user-specific event object. The event object is registered.
Return values	None

Table 8-247 RegisterOnHardwareConfigChangedEvent() – .NET (C#)

Syntax	`void RegisterOnHardwareConfigChangedEvent();`
Parameters	None
Return values	None

UnregisterOnHardwareConfigChangedCallback()

UnregisterOnHardwareConfigChangedCallback()Unregisters the callback function.

Table 8-248 UnregisterOnHardwareConfigChangedCallback() – Native C++

Syntax	`void UnregisterOnHardwareConfigChangedCallback();`
Parameters	Optional parameter of the same type as for the Register*() function If you use the parameter, it uninstalls the specific function; otherwise, the last registered callback function is unregistered.
Return values	None

UnregisterOnHardwareConfigChangedEvent()

Unregisters the event object.

Table 8-249 UnregisterOnHardwareConfigurationChangedEvent() – Native C++

Syntax	`void UnregisterOnHardwareConfigChangedEvent();`
Parameters	None
Return values	None

Table 8-250 UnregisterOnHardwareConfigurationChangedEvent() – .NET (C#)

Syntax	`void UnregisterOnHardwareConfigChangedEvent();`
Parameters	None
Return values	None

WaitForOnHardwareConfigurationChangedEvent()

The function blocks the program until the registered event object is in the signaled state or the timeout interval is exceeded.

Table 8-251 WaitForOnHardwareConfigChangedEvent() – Native C++

Syntax	`bool WaitForOnHardwareConfigChangedEvent();` `bool WaitForOnHardwareConfigChangedEvent (in_Time_ms);`
Parameters	• `None:` The time limit is set to INFINITE. • `UINT32 in_Time_ms:` Value for the time limit in milliseconds.
Return values	• `true`: If the event object was set to the signaled state. • `false`: If no event was received during the defined time limit.

Table 8-252 WaitForOnHardwareConfigChangedEvent() – .NET (C#)

Syntax	`bool WaitForOnHardwareConfigChangedEvent();` `bool WaitForOnHardwareConfigChangedEvent (UINT32 in_Time_ms);`
Parameters	• `None:` The time limit is set to `INFINITE`. • `UINT32 in_Time_ms:` Value for the time limit in milliseconds.
Return values	• `true`: If the event object was set to the signaled state. • `false`: If no event was received during the defined time limit.

8.11.2 Events for acyclic services

OnDataRecordRead

Registers or unregisters an event handler method.

Table 8-253 OnDataRecordRead - .NET (C#)

Syntax	`event Delegate_II_EREC_DT_SDRI OnDataRecordRead;`
Parameter	None. See Delegate_II_EREC_DT_SDRI (Page 234).
Return values	None
Exceptions	None
Note	The Event-Handler Methode runs in a separate thread.

OnDataRecordWrite

Registers or unregisters an event handler method.

Table 8-254 OnDataRecordWrite - .NET (C#)

Syntax	`event Delegate_II_EREC_DT_SDR OnDataRecordWrite;`
Parameter	None. See Delegate_II_EREC_DT_SDR (Page 234).
Return values	None
Exceptions	None
Note	The event handler method runs in a separate thread.

RegisterOnDataRecordReadCallback()

When the event occurs, the registered callback function is called. Multiple callback functions can be registered for the event.

Table 8-255 RegisterOnDataRecordReadCallback() - Native C++

Syntax	`void RegisterOnDataRecordReadCallback (Event Callback_II_SREC_ST_SDRI in_CallbackFunction);`
Parameters	• `EventCallback_II_SREC_ST_SDRI in_CallbackFunction:` A callback function that subscribes to the event. See EventCallback_II_SREC_ST_SDRI (Page 226).
Return values	None
Note	The callback function runs in a separate thread.

UnregisterOnDataRecordReadCallback()

Unregisters the callback function.

Table 8-256 UnregisterOnDataRecordReadCallback() - Native C++

Syntax	`void UnregisterOnDataRecordReadCallback();`
Parameters	Optional parameter of the same type as for the Register*() function If you use the parameter, it uninstalls the specific function; otherwise, the last registered callback function is unregistered.
Return values	None

RegisterOnDataRecordWriteCallback()

When the event occurs, the registered callback function is called. Multiple callback functions can be registered for the event.

Table 8-257 RegisterOnDataRecordWriteCallback() - Native C++

Syntax	`void RegisterOnDataRecordWriteCallback (EventCallback_II_SREC_ST_SDRI_BYTE in_CallbackFunction);`
Parameters	• `EventCallback_II_SREC_ST_SDRI_BYTE in_CallbackFunction:` A callback function that subscribes to the event. See EventCallback_II_SREC_ST_SDRI_BYTE (Page 226).
Return values	None
Note	The callback function runs in a separate thread.

UnregisterOnDataRecordWriteCallback()

Unregisters the callback function. When the event occurs, no callback function is called.

Table 8-258 UnregisterOnDataRecordWriteCallback() - Native C++

Syntax	`void UnregisterOnDataRecordWriteCallback();`
Parameters	Optional parameter of the same type as for the Register*() function If you use the parameter, it uninstalls the specific function; otherwise, the last registered callback function is unregistered.
Return values	None

8.11.2.1 OnAlarmNotification events

OnAlarmNotificationDone()

Registers or unregisters an event handler method.

Table 8-259 OnAlarmNotificationDone() - .NET (C#)

Syntax	`event Delegate_SREC_ST_UINT32_UINT32 OnAlarmNotificationDone;`
Parameters	None. See Delegate_SREC_ST_UINT32_UINT32 (Page 236).
Return values	None
Exceptions	None
Note	The event handler method runs in a separate thread.

RegisterOnAlarmNotificationDoneCallback()

When the event occurs, the registered callback function is called. Multiple callback functions can be registered for the event.

Table 8-260 RegisterOnAlarmNotificationDoneCallback() - Native C++

Syntax	`void RegisterOnAlarmNotificationDoneCallback (Event Callback_II_SREC_ST_SDRI in_CallbackFunction);`
Parameters	• `EventCallback_II_SREC_ST_UINT32_UINT32 in_CallbackFunction:` A callback function that subscribes to the event. See EventCallback_II_SREC_ST_UINT32_UINT32 (Page 227).
Return values	None
Note	The callback function runs in a separate thread.

UnregisterOnAlarmNotificationDoneCallback()

Unregisters the callback function.

Table 8-261 UnregisterOnAlarmNotificationDoneCallback() - Native C++

Syntax	`void UnregisterOnAlarmNotificationDoneCallback ();`
Parameters	Optional parameter of the same type as for the Register*() function If you use the parameter, it uninstalls the specific function; otherwise, the last registered callback function is unregistered.
Return values	None

8.11.2.2 OnProcessEvent events

OnProcessEventDone()

Registers or unregisters an event handler method.

Table 8-262 OnProcessEventDone() - .NET (C#)

Syntax	`event Delegate_SREC_ST_UINT32_UINT32_EPET_UINT32 OnProcessEventDone;`
Parameters	None. See Delegate_SREC_ST_UINT32_UINT32_EPET_UINT32. (Page 235)
Return values	None
Exceptions	None
Note	The event handler method runs in a separate thread.

RegisterOnProcessEventDoneCallback()

When the event occurs, the registered callback function is called. Multiple callback functions can be registered for the event.

Table 8-263 RegisterOnProcessEventDoneCallback() - Native C++

Syntax	`void RegisterOnProcessEventDoneCallback (EventCallback_II_SREC_ST_UINT32_UINT32_EPET_UINT32 in_CallbackFunction);`
Parameters	`EventCallback_II_SREC_ST_UINT32_UINT32_EPET_UINT32 in_CallbackFunction:` A callback function that subscribes to the event. See EventCallback_II_SREC_ST_UINT32_UINT32_EPET_UINT32 (Page 227)
Return values	None
Note	The callback function runs in a separate thread.

UnregisterOnProcessEventDoneCallback()

Unregisters the callback function.

Table 8-264 UnregisterOnProcessEventDoneCallback() - Native C++

Syntax	`void UnregisterOnProcessEventDoneCallback ();`
Parameters	Optional parameter of the same type as for the Register*() function If you use the parameter, it uninstalls the specific function; otherwise, the last registered callback function is unregistered.
Return values	None

8.11.2.3 OnPullOrPlugEvent events

OnPullOrPlugEventDone()

Registers or unregisters an event handler method.

Table 8-265 OnPullOrPlugEventDone() - .NET (C#)

Syntax	`event Delegate_SREC_ST_UINT32_EPPET_UINT32 OnPullOrPlugEventDone;`
Parameters	None. See Delegate_SREC_ST_UINT32_EPPET_UINT32 (Page 235).
Return values	None
Exceptions	None
Note	The event handler method runs in a separate thread.

RegisterOnPullOrPlugEventDoneCallback()

When the event occurs, the registered callback function is called. Multiple callback functions can be registered for the event.

Table 8-266 RegisterOnPullOrPlugEventDoneCallback() - Native C++

Syntax	`void RegisterOnPullOrPlugEventDoneCallback (EventCallback_II_SREC_ST_UINT32_EPPET_UINT32 in_CallbackFunction);`
Parameters	`EventCallback_II_SREC_ST_UINT32_EPPET_UINT32 in_CallbackFunction:` A callback function that subscribes to the event. See EventCallback_II_SREC_ST_UINT32_EPPET_UINT32 (Page 228).
Return values	None
Note	The callback function runs in a separate thread.

UnregisterOnPullOrPlugEventDoneCallback()

Unregisters the callback function.

Table 8-267 UnregisterOnPullOrPlugEventDoneCallback() - Native C++

Syntax	`void UnregisterOnPullOrPlugEventDoneCallback ();`
Parameters	Optional parameter of the same type as for the Register*() function If you use the parameter, it uninstalls the specific function; otherwise, the last registered callback function is unregistered.
Return values	None

8.11.2.4 OnStatusEvent events

OnStatusEventDone()

Registers or unregisters an event handler method.

Table 8-268 OnStatusEventDone() - .NET (C#)

Syntax	`event Delegate_SREC_ST_UINT32 OnStatusEventDone;`
Parameters	None. See Delegate_SREC_ST_UINT32 (Page 236).
Return values	None
Exceptions	None
Note	The event handler method runs in a separate thread.

RegisterOnStatusEventDoneCallback()

When the event occurs, the registered callback function is called. Multiple callback functions can be registered for the event.

Table 8-269 RegisterOnStatusEventDoneCallback() - Native C++

Syntax	`void RegisterOnStatusEventDoneCallback (EventCallback_II_SREC_ST_UINT32 in_CallbackFunction);`
Parameters	`EventCallback_II_SREC_ST_UINT32 in_CallbackFunction:` A callback function that subscribes to the event. See EventCallback_II_SREC_ST_UINT32 (Page 229).
Return values	None
Note	The callback function runs in a separate thread.

UnregisterOnStatusEventDoneCallback()

Unregisters the callback function.

Table 8-270 UnregisterOnStatusEventDoneCallback() - Native C++

Syntax	`void UnregisterOnStatusEventDoneCallback ();`
Parameters	Optional parameter of the same type as for the Register*()function. If you use the parameter, it uninstalls the specific function; otherwise, the last registered callback function is unregistered.
Return values	None

8.11.2.5 OnProfileEvent events

OnProfileEventDone()

Registers or unregisters an event handler method.

Table 8-271 OnProfileEventDone() - .NET (C#)

Syntax	`event Delegate_SREC_ST_UINT32 OnProfileEventDone;`
Parameters	None. See Delegate_SREC_ST_UINT32 (Page 236).
Return values	None
Exceptions	None
Note	The event handler method runs in a separate thread.

RegisterOnProfileEventDoneCallback()

When the event occurs, the registered callback function is called. Multiple callback functions can be registered for the event.

Table 8-272 RegisterOnProfileEventDoneCallback() - Native C++

Syntax	`void RegisterOnProfileEventDoneCallback (EventCallback_II_SREC_ST_UINT32 in_CallbackFunction);`
Parameters	`EventCallback_II_SREC_ST_UINT32 in_CallbackFunction:` A callback function that subscribes to the event. See EventCallback_II_SREC_ST_UINT32 (Page 229).
Return values	None
Note	The callback function runs in a separate thread.

UnregisterOnProfileEventDoneCallback()

Unregisters the callback function. When the event occurs, no callback function is called.

Table 8-273 UnregisterOnProfileEventDoneCallback() - Native C++

Syntax	`void UnregisterOnProfileEventDoneCallback ();`
Parameters	Optional parameter of the same type as for the Register*() function If you use the parameter, it uninstalls the specific function; otherwise, the last registered callback function is unregistered.
Return values	None

8.11.2.6 OnUpdateEvent events

OnUpdateEventDone()

Registers or unregisters an event handler method.

Table 8-274 OnUpdateEventDone() - .NET (C#)

Syntax	`event Delegate_SREC_ST_UINT32 OnUpdateEventDone;`
Parameters	None. See Delegate_SREC_ST_UINT32 (Page 236).
Return values	None
Exceptions	None
Note	The event handler method runs in a separate thread.

RegisterOnUpdateEventDoneCallback()

When the event occurs, the registered callback function is called. Multiple callback functions can be registered for the event.

Table 8-275 RegisterOnUpdateEventDoneCallback() - Native C++

Syntax	`void RegisterOnUpdateEventDoneCallback (EventCallback_II_SREC_ST_UINT32 in_CallbackFunction);`
Parameters	`EventCallback_II_SREC_ST_UINT32 in_CallbackFunction:` A callback function that subscribes to the event. See EventCallback_II_SREC_ST_UINT32 (Page 229).
Return values	None
Note	The callback function runs in a separate thread.

UnregisterOnUpdateEventDoneCallback()

Unregisters the callback function.

Table 8-276 UnregisterOnUpdateEventDoneCallback() - Native C++

Syntax	`void UnregisterOnUpdateEventDoneCallback ();`
Parameters	Optional parameter of the same type as for the Register*() function If you use the parameter, it uninstalls the specific function; otherwise, the last registered callback function is unregistered.
Return values	None

8.11.2.7 RackOrStationFault events

OnRackOrStationFaultEvent

Registers or unregisters an event handler method.

Table 8-277 OnRackOrStationFaultEvent - .NET (C#)

Syntax	`event Delegate_SREC_ST_UINT32_ERSFET OnRackOrStationFault;`
Parameter	None. See Delegate_SREC_ST_UINT32_ERSFET (Page 237).
Return values	None
Exceptions	None
Note	The event handler method runs in a separate thread.

RegisterOnRackOrStationFaultEventCallback()

When the event occurs, the registered callback function is called. Multiple callback functions can be registered for the event.

Table 8-278 RegisterOnRackOrStationFaultEventCallback() - Native C++

Syntax	`void RegisterOnRackOrStationFaultEventCallback ( EventCallback_II_SREC_ST_UINT32_ERSFET in_CallbackFunction );`
Parameter	`EventCallback_II_ SREC_ST_UINT32_ERSFET in_CallbackFunction.` A callback function that subscribes to the event. See EventCallback_II_SREC_ST_UINT32_ERSFET (Page 228)
Return values	None
Note	The callback function runs in a separate thread.

UnregisterOnRackOrStationFaultEventCallback()

Unregisters the callback function.

Table 8-279 UnregisterOnRackOrStationFaultEventCallback() - Native C++

Syntax	`void UnregisterOnRackOrStationFaultEventCallback ();`
Parameter	Optional parameter of the same type as for the Register*() function If you use the parameter, it uninstalls the specific function; otherwise, the last registered callback function is unregistered.
Return values	None

IRemoteRuntimeManager

9

9.1 Interfaces - Information and settings

Dispose()

Deletes the managed interface and unloads the native components of the user interfaces.

NOTE

When the interface of the Remote Runtime Manager is deleted, no IInstance interface which was generated by the IRemoteRuntimeManager interface can be used.

The .NET Garbage Collector clears its IRemoteRuntimeManager interface when no active references are present.

Table 9-1 Dispose() - .NET (C#)

Syntax	`void Dispose()`
Parameters	None
Return values	None

GetIP() / IP { get; }

Returns the IP address of the PC on which the remote Runtime Manager is running. If the function fails, the return value is 0.

Table 9-2 GetIP() - Native C++

Syntax	`UIP GetIP();`
Parameters	None
Return values	`UIP`: Returns the IP address of the PC on which the Runtime Manager is running.

Table 9-3 IP { get; } - .NET (C#)

Syntax	`SIP IP { get; }`
Parameters	None
Return values	`SIP`: Returns the IP address of the PC on which the Runtime Manager is running.

GetRemoteComputerName() / RemoteComputerName { get; }

This method returns the NetBIOS name of the PC on which the remote Runtime Manager is running.

When the name of the PC cannot be identified, the method returns the IP address.

Table 9-4 GetRemoteComputerName() - Native C++

Syntax	`ERuntimeErrorCode GetRemoteComputerName(WCHAR* inout_Name,UINT32 in_ArrayLength);`	
Parameters	• `WCHAR* inout_Name:` A user-allocated array for the computer name. • `UINT32 in_ArrayLength:` The array length. The array should be longer than `MAX_COMPUTERNAME_LENGTH`.	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The interface is disconnected from the remote Runtime Manager.
	`SREC_INDEX_OUT_OF_RANGE`	The array is too small to accommodate the computer name.

Table 9-5 RemoteComputerName { get; } - .NET (C#)

Syntax	`string RemoteComputerName { get; }`	
Parameters	None	
Return values	`string`: Name of the PC on which the remote Runtime Manager is running.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The interface is disconnected from the remote Runtime Manager.
	`ERuntimeErrorCode.IndexOutOfRange`	The array is too small to accommodate the computer name.

Disconnect()

Closes the connection to the remote Runtime Manager.

NOTE

All applications that are connected to the remote Runtime Manager lose this connection.

Table 9-6 Disconnect() - Native C++

Syntax	`ERuntimeErrorCode Disconnect();`	
Parameters	None	
Return values	Runtime error code	Description
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The interface is disconnected from the remote Runtime Manager.
	`SREC_TIMEOUT`	The function does not return on time.

Table 9-7 Disconnect() - .NET (C#)

Syntax	`void Disconnect();`	
Parameters	None	
Return values	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Description
	`ERuntimeErrorCode.InterfaceRemoved`	The interface is disconnected from the remote Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.

9.2 Events for IRemoteRuntimeManager

9.2.1 OnConnectionLost events

Description

The event is triggered when the connection to the Remote Runtime Manager has been terminated.

OnConnectionLost

Registers or unregisters an event handler method.

Table 9-8 OnConnectionLost - .NET (C#)

Syntax	`event Delegate_IRRTM OnConnectionLost;`
Parameters	None. See Delegate_IRRTM (Page 233)
Return values	None
Exceptions	None
Note	The event handler method runs in a separate thread.

RegisterOnConnectionLostCallback()

When the event occurs, the registered callback function is called. Multiple callback functions can be registered for the event.

Table 9-9 RegisterOnConnectionLostCallback() - Native C++

Syntax	`void RegisterOnConnectionLostCallback(EventCallback_IRRTM in_CallbackFunction);`
Parameters	`EventCallback_IRRTM in_CallbackFunction:` A callback function that subscribes to an event. See EventCallback_IRRTM (Page 223).
Return values	None
Note	The callback function runs in a separate thread.

RegisterOnConnectionLostEvent()

When the event occurs, the registered event object is set to the signaled state. Only one event object can be registered for the event. Registration of a new event object causes the previous event object to be deleted.

Table 9-10 RegisterOnConnectionLostEvent() - Native C++

Syntax	`void RegisterOnConnectionLostEvent();` `void RegisterOnConnectionLostEvent(HANDLE* in_Event);`
Parameters	• `None:` An internal event object is registered. • `HANDLE* in_Event:` A handle for a user-specific event object. The event object is registered.
Return values	None

Table 9-11 RegisterOnConnectionLostEvent() - .NET (C#)

Syntax	`void RegisterOnConnectionLostEvent();`
Parameters	None
Return values	None

UnregisterOnConnectionLostCallback()

Unregisters the callback function. When the event occurs, no callback function is called.

Table 9-12 UnregisterOnConnectionLostCallback() - Native C++

Syntax	`void UnregisterOnConnectionLostCallback();`
Parameters	Optional parameter of the same type as for the Register*() function If you use the parameter, it uninstalls the specific function; otherwise, the last registered callback function is unregistered.
Return values	None

UnregisterOnConnectionLostEvent()

Unregisters the event object.

Table 9-13 UnregisterOnConnectionLostEvent() - Native C++

Syntax	`void UnregisterOnConnectionLostEvent();`
Parameters	None
Return values	None

Table 9-14 UnregisterOnConnectionLostEvent() - .NET (C#)

Syntax	`void UnregisterOnConnectionLostEvent();`
Parameters	None
Return values	None

WaitForOnConnectionLostEvent()

The function blocks the program until the registered event object is in the signaled state or the timeout interval is exceeded.

Table 9-15 WaitForOnConnectionLostEvent() - Native C++

Syntax	`bool WaitForOnConnectionLostEvent();` `bool WaitForOnConnectionLostEvent(UINT32 in_Time_ms);`
Parameters	• `None`: The time limit is set to `INFINITE`. • `UINT32 in_Time_ms`: Value for the time limit in milliseconds.
Return values	• `true`: If the event object was set to the signaled state. • `false`: If no event was received during the defined time limit.

Table 9-16 WaitForOnConnectionLostEvent() - .NET (C#)

Syntax	`bool WaitForOnConnectionLostEvent();` `bool WaitForOnConnectionLostEvent(UInt32 in_Time_ms);`
Parameters	• `None`: The time limit is set to `INFINITE`. • `UInt32 in_Time_ms`: Value for the time limit in milliseconds.
Return values	• `true`: If the event object was set to the signaled state. • `false`: If no event was received during the defined time limit.

Data types

10

Supported data types

In S7-PLCSIM Advanced V6.0, the Runtime API supports the data types of the S7-1500 CPUs.

Converting data types

When writing, data types are not transferred BCD-coded but mapped onto primitive data types.

The data types Counter, Date and Time must be transferred to the API BCD-coded so that the values are written to the counter and no incorrect values are returned when reading.

For these data types, you must perform a BCD conversion before writing and a BCD back-conversion after reading.

Example:
If the value 999 is transferred to the API as 2457_H, then `Write` modifies the value 2457_H to 999. Without BCD conversion, there is no `UInt16` value and `Write` writes no value at all.

Additional information

For information on data types and conversion, refer to section "Data types" in the SIMATIC STEP 7 Basic/Professional (https://support.industry.siemens.com/cs/ww/en/view/109755202) System Manual.

10.1 DLL import functions (Native C++)

10.1.1 ApiEntry_Initialize

Description

Type of the central entry point for the API library (DLL).

Table 10-1 ApiEntry_Initialize - Native C++

Syntax	`typedef ERuntimeErrorCode(*ApiEntry_Initialize)(ISimulationRuntimeManager** out_RuntimeManagerInterface);`	
Parameters	• `ISimulationRuntimeManager** out_SimulationRuntimeManagerInterface:` Pointer to a Runtime Manager interface pointer. The pointer must be initialized with NULL. The interface is created within the function. • `UINT32 in_InterfaceVersion:` Version of the API interface to be downloaded: `API_DLL_INTERFACE_VERSION`.	
Return values	Runtime error code	Condition

Return values	SREC_OK	The function is successful.
	SREC_WRONG_ARGUMENT	The pointer to the Runtime Manager interface is NULL.
	SREC_WRONG_VERSION	The version of the interface in use does not match the version of the API library (DLL).
	SREC_CONNECTION_ERROR	Unable to establish a connection to the Runtime Manager.
	SREC_CONFIG_FILE_ERROR	Operation regarding the configuration file "UserInterfaceConfiguration.xml" has failed, for example, create, read, write.

10.1.2 ApiEntry_DestroyInterface

Description

Type of the entry point for DestroyInterface (Page 38).

Table 10-2 ApiEntry_DestroyInterface - Native C++

Syntax	`typedef ERuntimeErrorCode(*ApiEntry_DestroyInterface)(IBaseInterface* in_Interface);`	
Parameters	`IBaseInterface* in_Interface:` The interface to be deleted.	
Return values	Runtime error code	Condition
	SREC_OK	The function is successful.
	SREC_WRONG_ARGUMENT	The pointer to the interface is NULL.

10.2 Event callback functions (Native C++)

10.2.1 EventCallback_VOID

Description

Table 10-3 EventCallback_VOID - Native C++

Syntax	`typedef void (*EventCallback_VOID)();`
Parameters	None
Return values	None

10.2.2 EventCallback_SRCC_UINT32_UINT32_INT32

Description

Table 10-4 EventCallback_SRCC_UINT32_UINT32_INT32 - Native C++

Syntax	`ERuntimeConfigChanged in_RuntimeConfigChanged, UINT32 in_Param1, UINT32 in_Param2, INT32 in_Param3);`			
Parameters	`ERuntimeConfigChanged in_RuntimeConfigChanged`	`UInt32 in_Param1`	`UInt32 in_Param2`	`Int32 in_Param3`
	`SRCC_INSTANCE_REGISTERED`	–	–	ID of the registered instance
	`SRCC_INSTANCE_UNREGISTERED`	–	–	ID of the unregistered instance
	`SRCC_CONNECTION_OPENED`	IP of the remote Runtime Manager	Port of the remote Runtime Manager	–
	`SRCC_CONNECTION_CLOSED`	IP of the remote Runtime Manager	Port of the remote Runtime Manager	–
	`SRCC_PORT_OPENED`	The open port	–	–
	`SRCC_PORT_CLOSED`	–	–	–
Return values	None			

10.2.3 EventCallback_SRRSI_AD

Description

Table 10-5 EventCallback_SRRSI_AD - Native C++

Syntax	`typedef void (*EventCallback_SRRSI_AD)(EAutodiscoverType in_AutodiscoverMsg, SAutodiscoverData in_AutodiscoverData);`
Parameters	• `in_AutodiscoverMsg`: A value from the list of predefined types of events, see EAutodiscoverType (Page 275). – `SRRSI_DISCOVER_STARTED`, if the identification process was started by successfully calling the function `RunAutodisover()`. – `SRRSI_DISCOVER_DATA`, if a Runtime Manager in the network was determined by the identification process. For detailed information about the found Runtime Manager, see parameter `in_AutodiscoverData`. – `SRRSI_DISCOVER_FINISHED`, if the identification process was completed after the time defined by the "in_Timeout" parameter had elapsed. – `SRRSI_DISCOVER_STARTED` and `SRRSI_DISCOVER_FINISHED` are always triggered, even if no data is received. • `in_AutodiscoverData`: Data from the Remote Runtime Manager. The parameter contains valid data only if `in_AutodiscoverMsg` = `SRRSI_DISCOVER_DATA`. Otherwise it is initialized with 0. See SAutodiscoverData (Page 254).
Return values	None

10.2.4 EventCallback_IRRTM

Description

Table 10-6 EventCallback_IRRTM - Native C++

Syntax	`typedef void (*EventCallback_IRRTM)(IRemoteRuntimeManager* in_Sender);`
Parameters	• `IRemoteRuntimeManager* in_Sender:` An interface of the remote Runtime Manager that receives this event.
Return values	None

10.2.5 EventCallback_II_SREC_ST_SROS_SROS

Description

Table 10-7 EventCallback_II_SREC_ST_SROS_SROS - Native C++

Syntax	`typedef void (*EventCallback_II_SREC_ST_SROS_SROS)(IInstance* in_Sender, ERuntimeErrorCode in_ErrorCode, SYSTEMTIME in_SystemTime, EOperatingState in_PrevState, EOperatingState in_OperatingState);`	
Parameters	• `IInstance* in_Sender:` An interface of the instance that receives this event. • `ERuntimeErrorCode in_ErrorCode:` A possible error code. • `SYSTEMTIME in_SystemTime:` The virtual system time of the simulated PLC instance at the time when this event was triggered. • `EOperatingState in_PrevState:` The operating state before the change. • `EOperatingState in_OperatingState:` The current operating state.	
Return values	None	
Error codes	Runtime error code	Condition
	`SREC_OK`	The function is successful.
	`SREC_WARNING_TRIAL_MODE_ACTIVE`	No license available. You can use the instance without restrictions with the Trial License. Afterwards, the instance is shut down.
	`SREC_LICENSE_NOT_FOUND`	Test mode has expired.
	`SREC_COMMUNICATION_INTERFACE_NOT_AVAILABLE`	A problem has occurred with the selected communication interface. Check your settings.

10.2.6 EventCallback_II_SREC_ST_UINT32_INT64_INT64_UINT32

Description

Table 10-8 EventCallback_II_SREC_ST_UINT32_INT64_INT64_UINT32 - Native C++

Syntax	`typedef void (*EventCallback_II_SREC_ST_UINT32_INT64_INT64_UINT32)(IInstance* in_Sender, ERuntimeErrorCode in_ErrorCode, SYSTEMTIME in_SystemTime, UINT32 in_Id, INT64 in_TimeSinceSameSyncPoint_ns, INT64 in_TimeSinceAnySyncPoint_ns, UINT32 in_SyncPointCount);`
Parameters	• `IInstance* in_Sender:` An interface of the instance that receives this event. • `ERuntimeErrorCode in_ErrorCode:` A possible error code. • `SYSTEMTIME in_SystemTime:` The virtual system time of the simulated PLC instance at the time when this event was triggered. • `UINT32 in_Id:` For the `SingleStep_Bus` operating mode: The ID of the I/O system that is used for cycle-controlled synchronization of the simulated PLC instance. For all other operating modes: The ID of the process image partition (PIP) that triggers this event. 0 for the cycle control point (End of cycle). • `INT64 in_TimeSinceSameSyncPoint_ns:` The virtual time (in nanoseconds) since the last synchronization point of the same process image partition ID was reached. For the time-controlled operating modes: The runtime since the last call of the `StartProcessing()` function. • `INT64 in_TimeSinceAnySyncPoint_ns:` The virtual time (in nanoseconds) since the last synchronization point of any process image partition ID was reached. For the time-controlled operating modes: The runtime since the last call of the `StartProcessing()` function. • `UINT32 in_SyncPointCount:` The number of synchronization points since the last event. If the events are triggered faster than they are received, multiple events are combined into one event. In this case, this value contains the number of cycles since the last event was received.
Return values	None

10.2.7 EventCallback_II_SREC_ST

Description

Table 10-9 EventCallback_II_SREC_ST - Native C++

Syntax	`typedef void (*EventCallback_II_SREC_ST)(IInstance* in_Sender, ERuntimeErrorCode in_ErrorCode, SYSTEMTIME in_SystemTime);`
Parameters	• `IInstance* in_Sender:` An interface of the instance that receives this event. • `ERuntimeErrorCode in_ErrorCode:` A possible error code. • `SYSTEMTIME in_SystemTime:` The virtual system time of the simulated PLC instance at the time when this event was triggered.
Return values	None

10.2.8 EventCallback_II_SREC_ST_SRICC_UINT32_UINT32_UINT32_UINT32

Description

Table 10-10 EventCallback_II_SREC_ST_SRICC_UINT32_UINT32_UINT32_UINT32 - Native C++

<table>
<tr><td>Syntax</td><td colspan="5"><code>typedef void (*EventCallback_II_SREC_ST_SRICC_UINT32_UINT32_UINT32_UINT32)(IInstance* in_Sender, ERuntimeErrorCode in_ErrorCode, SYSTEMTIME in_SystemTime, EInstanceConfigChanged in_InstanceConfigChanged, UINT32 in_Param1, UINT32 in_Param2, UINT32 in_Param3, UINT32 in_Param4);</code></td></tr>
<tr><td>Parameters</td><td colspan="5">• <code>IInstance in_Sender:</code>
An interface of the instance that receives this event.
• <code>ERuntimeErrorCode in_ErrorCode:</code>
A possible error code.
• <code>SYSTEMTIME in_SystemTime:</code>
The virtual system time of the simulated PLC instance at the time when this event was triggered.
<table>
<tr><td><code>EInstanceConfigChanged in_InstanceConfigChanged</code></td><td><code>UINT32 in_Param1</code></td><td><code>UINT32 in_Param2</code></td><td><code>UINT32 in_Param3</code></td><td><code>UINT32 in_Param4</code></td></tr>
<tr><td><code>SRICC_HARDWARE_SOFTWARE_CHANGED</code></td><td>–</td><td>–</td><td>–</td><td>–</td></tr>
<tr><td><code>SRICC_IP_CHANGED</code></td><td>The ID of the interface</td><td>The new IP</td><td>The new subnet mask</td><td>The new standard gateway</td></tr>
</table></td></tr>
<tr><td>Return values</td><td colspan="5">None</td></tr>
</table>

10.2.9 EventCallback_II_SREC_ST_SRLT_SRLM

Description

Table 10-11 EventCallback_II_SREC_ST_SRLT_SRLM - Native C++

Syntax	`typedef void (*EventCallback_II_SREC_ST_SRLT_SRLM)(IInstance* in_Sender, ERuntimeErrorCode in_ErrorCode, SYSTEMTIME in_SystemTime, ELEDType in_LEDType, ELEDMode in_LEDMode);`
Parameters	• `IInstance* in_Sender:` An interface of the instance that receives this event. • `ERuntimeErrorCode in_ErrorCode:` A possible error code. • `SYSTEMTIME in_SystemTime:` The virtual system time of the simulated PLC instance at the time when this event was triggered. • `ELEDType in_LEDType:` The LED type that changed its state. • `ELEDMode in_LEDMode:` The new state of the LED display.
Return values	None

10.2.10 EventCallback_II_SREC_ST_SDRI

Description

Table 10-12 EventCallback_II_SREC_ST_SDRI - Native C++

Syntax	`typedef void (*EventCallback_II_SREC_ST_SDRI)(IInstance* in_Sender, ERuntimeErrorCode in_ErrorCode, SYSTEMTIME in_SystemTime, SDataRecordInfo in_DataRecordInfo);`
Parameters	• `IInstance* in_Sender:` An interface of the instance that receives this event. • `ERuntimeErrorCode in_ErrorCode:` A possible error code. • `SYSTEMTIME in_SystemTime:` The virtual system time of the simulated PLC instance at the time when this event was triggered. • `SDataRecordInfo in_DataRecordInfo:` The structure `SDataRecordInfo` contains the following information: – The HW identifier from which the CPU is to read the data record – The index of the collected data record – The maximum size of the data record which the IO device can transfer.
Return values	None

10.2.11 EventCallback_II_SREC_ST_SDRI_BYTE

Description

Table 10-13 EventCallback_II_SREC_ST_SDRI_BYTE - Native C++

Syntax	`typedef void (*EventCallback_II_SREC_ST_SDRI_BYTE)(IInstance* in_Sender, ERuntimeErrorCode in_ErrorCode, SYSTEMTIME in_SystemTime, SDataRecordInfo in_DataRecordInfoconst BYTE* in_Data);`
Parameters	• `IInstance* in_Sender:` An interface of the instance that receives this event. • `ERuntimeErrorCode in_ErrorCode:` A possible error code. • `SYSTEMTIME in_SystemTime:` The virtual system time of the simulated PLC instance at the time when this event was triggered. • `SDataRecordInfo in_DataRecordInfo:` The structure `SDataRecordInfo` contains the following information: – The HW identifier to which the CPU wants to write the data record – The index of the supplied data record – Size of data record • `const BYTE* in_Data:` The data record. This pointer becomes invalid after the callback function has returned.
Return values	None

10.2.12 EventCallback_II_SREC_ST_UINT32_UINT32

Description

Table 10-14 EventCallback_II_SREC_ST_UINT32_UINT32 - Native C++

Syntax	`typedef void (*EventCallback_II_SREC_ST_UINT32_UINT32)(IInstance* in_Sender, ERuntimeErrorCode in_ErrorCode, SYSTEMTIME in_SystemTime, UINT32 in_HardwareIdentifier), UINT32 in_SequenceNumber);`
Parameters	• `IInstance* in_Sender:` An interface of the instance that receives this event. • `ERuntimeErrorCode in_ErrorCode:` A possible error code. • `SYSTEMTIME in_SYSTEMTIME:` The virtual system time of the simulated PLC instance at the time when this event was triggered. • `UINT32 in_HardwareIdentifier:` The hardware identifier of the module or submodule which sends the diagnostics event. • `UINT32 in_SequenceNumber:` S7-PLCSIM Advanced assigns a unique consecutive number to each interrupt event. According to PROFINET standard the sequence number is 10 bits wide (1 to $7FF_H$). When the highest number is reached the numbering starts again at 1. **Note** In a real hardware system the IO controller uses the sequence number to check if it has lost a hardware interrupt. During the simulation, the sequence number creates the relation between interrupt request and the associated acyclic alarm.
Return values	None

10.2.13 EventCallback_II_SREC_ST_UINT32_UINT32_EPET_UINT32

Description

Table 10-15 EventCallback_II_SREC_ST_UINT32_UINT32_EPET_UINT32 - Native C++

Syntax	`typedef void (*EventCallback_II_SREC_ST_UINT32_UINT32_EPET_UINT32)(IInstance* in_Sender, ERuntimeErrorCode in_ErrorCode, SYSTEMTIME in_SystemTime, UINT32 in_HardwareIdentifier, UINT32 in_Channel, EProcessEventType in_ProcessEventType, UINT32 in_SequenceNumber);`
Parameters	• `IInstance* in_Sender:` An interface of the instance that receives this event. • `ERuntimeErrorCode in_ErrorCode:` A possible error code. • `SYSTEMTIME in_SystemTime:` The virtual system time of the simulated PLC instance at the time when this event was triggered. • `UINT32 in_HardwareIdentifier:` The hardware identifier of the I/O module that sends the process event. • `UINT32 in_Channel:` The channel of the I/O module which sends the process event. • `EProcessEventType in_ProcessEventType:` A value from the list of predefined types of events for S7 modules, see EProcessEventType (Page 272). • `UINT32 in_SequenceNumber:` S7-PLCSIM Advanced assigns a unique consecutive number to each interrupt event. According to PROFINET standard the sequence number is 10 bits wide (1 to $7FF_H$). When the highest number is reached the numbering starts again at 1. **Note**

	In a real hardware system the I/O controller uses the sequence number to check if it has lost a hardware interrupt. During the simulation, the sequence number creates the relation between interrupt request and the associated acyclic alarm.
Return values	None

10.2.14 EventCallback_II_SREC_ST_UINT32_EPPET_UINT32

Description

Table 10-16 EventCallback_II_SREC_ST_UINT32_EPPET_UINT32 - Native C++

Syntax	`typedef void (*EventCallback_II_SREC_ST_UINT32_EPPET_UINT32)(IInstance* in_Sender, ERuntimeErrorCode in_ErrorCode, SYSTEMTIME in_SYSTEMTIME, UINT32 in_HardwareIdentifier, EPullOrPlugEventType in_PullOrPlugEventType, UINT32 in_SequenceNumber);`
Parameters	• `IInstance* in_Sender:` An interface of the instance that receives this event. • `ERuntimeErrorCode in_ErrorCode:` A possible error code. • `SYSTEMTIME in_SYSTEMTIME:` The virtual system time of the simulated PLC instance at the time when this event was triggered. • `UINT32 in_HardwareIdentifier:` The hardware identifier of the module or submodule which sends the pull/plug event. • `EPullOrPlugEventType in_PullOrPlugEventType:` A value from the list of predefined types of events for S7 modules, see EPullOrPlugEventType (Page 272). • `UINT32 in_SequenceNumber:` S7-PLCSIM Advanced assigns a unique consecutive number to each interrupt event. According to PROFINET standard the sequence number is 10 bits wide (1 to $7FF_H$). When the highest number is reached the numbering starts again at 1. **Note** In a real hardware system the IO controller uses the sequence number to check if it has lost a hardware interrupt. During the simulation, the sequence number creates the relation between interrupt request and the associated acyclic alarm.
Return values	None

10.2.15 EventCallback_II_SREC_ST_UINT32_ERSFET

Description

Table 10-17 EventCallback_II_SREC_ST_UINT32_ERSFET - Native C++

Syntax	`typedef void (*EventCallback_II_SREC_ST_UINT32_ERSFET)(IInstance* in_Sender, ERuntimeErrorCode in_ErrorCode, SYSTEMTIME in_SystemTime, UINT32 in_HardwareIdentifier, ERackOrStationFaultType in_EventType);`
Parameters	• `IInstance* in_Sender:` An interface of the instance that receives this event. • `ERuntimeErrorCode in_ErrorCode:` A possible error code. • `SYSTEMTIME in_SYSTEMTIME:` The virtual system time of the simulated PLC instance at the time when this event was triggered.

	• `UINT32 in_HardwareIdentifier:` The hardware identifier of the module or submodule which sends the diagnostics event. • `ERackOrStationFaultType in_EventType:` A value from the list of predefined RackOrStationFault event types. See ERackOrStationFaultType (Page 274).
Return values	None

10.2.16 EventCallback_II_SREC_ST_UINT32

Description

Table 10-18 EventCallback_II_SREC_ST_UINT32 - Native C++

Syntax	`typedef void (*EventCallback_II_SREC_ST_UINT32)(IInstance* in_Sender, ERuntimeErrorCode in_ErrorCode, SYSTEMTIME in_SystemTime, UINT32 in_HardwareIdentifier);`
Parameters	• `IInstance* in_Sender:` An interface of the instance that receives this event. • `ERuntimeErrorCode in_ErrorCode:` A possible error code. • `SYSTEMTIME in_SYSTEMTIME:` The virtual system time of the simulated PLC instance at the time when this event was triggered. • `UINT32 in_HardwareIdentifier:` The hardware identifier of the module or submodule which generates the status, update or Profile event. The identifier must belong to a hardware component in the currently loaded project.
Return values	None

10.3 Delegate definitions (managed code)

10.3.1 Delegate_Void

Description

Table 10-19 Delegate_Void - .NET (C#)

Syntax	`delegate void Delegate_Void();`
Parameters	None
Return values	None

10.3.2 Delegate_SRCC_UINT32_UINT32_INT32

Description

Table 10-20 Delegate_SRCC_UINT32_UINT32_INT32 - .NET (C#)

Syntax	`delegate void Delegate_SRCC_UINT32_UINT32_INT32(ERuntimeConfigChanged in_RuntimeConfigChanged, UInt32 in_Param1, UInt32 in_Param2, Int32 in_Param3);`			
Parameters	`ERuntimeConfigChanged in_RuntimeConfigChanged`	`UInt32 in_Param1`	`UInt32 in_Param2`	`Int32 in_Param3`
	`InstanceRegistered`	–	–	ID of the registered instance
	`InstanceUnregistered`	–	–	ID of the unregistered instance
	`ConnectionOpened`	IP of the Remote Runtime Manager	Port of the remote Runtime Manager	–
	`ConnectionClosed`	IP of the Remote Runtime Manager	Port of the remote Runtime Manager	–
	`PortOpened`	The open port	–	–
	`PortClosed`	–	–	–
Return values	None			

10.3.3 Delegate_SRRSI_AD

Description

Table 10-21 Delegate_SRRSI_AD - .NET (C#)

Syntax	`delegate void Delegate_SRRSI_AD(EAutodiscoverType in_AutodiscoverType, SAutodiscoverData in_AutodiscoverData);`
Parameters	• `in_AutodiscoverType` A value from the list of predefined types of events, see EAutodiscoverType (Page 275). – `AutodiscoverStarted`, if the identification process was started by successfully calling the function `RunAutodisover()`. – `AutodiscoverData`, if a Runtime Manager in the network was determined by the identification process. For detailed information about the found Runtime Manager, see parameter `in_AutodiscoverData`. – `AutodiscoverFinished`, if the identification process is completed after the time defined by the parameter "in_Timeout" has elapsed. – `AutodiscoverStarted` and `AutodiscoverFinished` are always triggered, even if no data is received. • `in_AutodiscoverData` Data from the Remote Runtime Manager. The parameter contains valid data only if `in_AutodiscoverType` = `AutodiscoverData`. Otherwise it is initialized with 0. See SAutodiscoverData (Page 254).
Return values	None

10.3.4 Delegate_II_EREC_DT

Description

Table 10-22 Delegate_II_EREC_DT - .NET (C#)

Syntax	`delegate void Delegate_II_EREC_DT (IInstance in_Sender, ERuntimeErrorCode in_ErrorCode, DateTime in_DateTime);`
Parameters	• `IInstance in_Sender:` An interface of the instance that receives this event. • `ERuntimeErrorCode in_ErrorCode:` A possible error code. • `DateTime in_DateTime:` The virtual system time of the simulated PLC instance at the time when this event was triggered.
Return values	None

10.3.5 Delegate_II_EREC_DT_EOS_EOS

Description

Table 10-23 Delegate_II_EREC_DT_EOS_EOS - .NET (C#)

Syntax	`delegate void Delegate_II_EREC_DT_EOS_EOS(IInstance in_Sender, ERuntimeErrorCode in_ErrorCode, DateTime in_DateTime, EOperatingState in_PrevState, EOperatingState in_OperatingState);`	
Parameters	• `IInstance in_Sender:` An interface of the instance that receives this event. • `ERuntimeErrorCode in_ErrorCode:` A possible error code. • `DateTime in_DateTime:` The virtual system time of the simulated PLC instance at the time when this event was triggered. • `EOperatingState in_PrevState:` The operating state before the change. • `EOperatingState in_OperatingState:` The current operating state.	
Return values	None	
Error codes	Runtime error code	Condition
	`ERuntimeErrorCode.OK`	The function is successful.
	`ERuntimeErrorCode.WarningTrialMode-Active`	No license available. You can use the instance without restrictions with the Trial License. Afterwards, the instance is shut down.
	`ERuntimeErrorCode.LicenseNotFound`	Test mode has expired.
	`ERuntimeErrorCode.CommunicationInterfaceNotAvailable`	A problem has occurred with the selected communication interface. Check your settings.

10.3.6 Delegate_II_EREC_DT_ELT_ELM

Description

Table 10-24 Delegate_II_EREC_DT_ELT_ELM - .NET (C#)

Syntax	`delegate void Delegate_II_EREC_DT_ELT_ELM(IInstance in_Sender, ERuntimeErrorCode in_ErrorCode, DateTime in_DateTime, ELEDType in_LEDType, ELEDMode in_LEDMode);`
Parameters	• `IInstance in_Sender:` An interface of the instance that receives this event. • `ERuntimeErrorCode in_ErrorCode:` A possible error code. • `DateTime in_DateTime:` The virtual system time of the simulated PLC instance at the time when this event was triggered. • `ELEDType in_LEDType:` The LED type that changed its state. • `ELEDMode in_LEDMode:` The new state of the LED display.
Return values	None

10.3.7 Delegate_II_EREC_DT_UINT32_INT64_INT64_UINT32

Description

Table 10-25 Delegate_II_EREC_DT_UINT32_INT64_INT64_UINT32 - .NET (C#)

Syntax	`delegate void Delegate_II_EREC_DT_UINT32_INT64_INT64_UINT32 (IInstance in_Sender, ERuntimeErrorCode in_ErrorCode, DateTime in_DateTime, UInt32 in_Id, Int64 in_TimeSinceSameSyncPoint_ns, Int64 in_TimeSinceAnySyncPoint_ns, UInt32 in_SyncPointCount);`
Parameters	• `IInstance in_Sender:` An interface of the instance that receives this event. • `ERuntimeErrorCode in_ErrorCode:` A possible error code. • `DateTime in_DateTime:` The virtual system time of the simulated PLC instance at the time when this event was triggered. • `UInt32 in_Id:` For the `SingleStep_Bus` operating mode: The ID of the I/O system that is used for cycle-controlled synchronization of the simulated PLC instance. For all other operating modes: The ID of the process image partition (PIP) that triggers this event. 0 for the cycle control point (End of cycle). Note: In the SingleStep operating modes, the transferred ID "in_Id" for the cycle control point has the value 0xFFFF. In this case, the time values are set to 0. • `Int64 in_TimeSinceSameSyncPoint_ns:` The virtual time (in nanoseconds) since the last synchronization point of the same process image partition ID was reached. Or the process time for the time-controlled operating modes.

	• `Int64 in_TimeSinceAnySyncPoint_ns:` The virtual time (in nanoseconds) since the last synchronization point of any process image partition ID was reached. Or the process time for the time-controlled operating modes. • `UInt32 in_SyncPointCount:` The number of synchronization points since the last event. If the events are triggered faster than they are received, multiple events are combined into one event. In this case, this value contains the number of cycles since the last event was received.
Return values	None

10.3.8 Delegate_IRRTM

Description

Table 10-26 Delegate_IRRTM - .NET (C#)

Syntax	`delegate void Delegate_IRRTM(IRemoteRuntimeManager in_Sender);`
Parameters	• `IRemoteRuntimeManager in_Sender:` An interface of the remote Runtime Manager that receives this event.
Return values	None

10.3.9 Delegate_II_EREC_DT_SRICC_UINT32_UINT32_UINT32_UINT32

Description

Table 10-27 Delegate_II_EREC_DT_SRICC_UINT32_UINT32_UINT32_UINT32 - .NET (C#)

<table>
<tr><td>Syntax</td><td colspan="5"><code>delegate void Delegate_II_EREC_DT_SRICC_UINT32_UINT32_UINT32_UINT32(IInstance in_Sender, ERuntimeErrorCode in_ErrorCode, DateTime in_DateTime, EInstanceConfigChanged in_InstanceConfigChanged, UInt32 in_Param1, UInt32 in_Param2, UInt32 in_Param3, UInt32 in_Param4);</code></td></tr>
<tr><td>Parameters</td><td colspan="5">• <code>IInstance in_Sender:</code>
An interface of the instance that receives this event.
• <code>ERuntimeErrorCode in_ErrorCode:</code>
A possible error code.
• <code>DateTime in_DateTime:</code>
The virtual system time of the simulated PLC instance at the time when this event was triggered.</td></tr>
<tr><td></td><td><code>EInstanceConfigChanged in_InstanceConfigChanged</code></td><td><code>UInt32 in_Param1</code></td><td><code>UInt32 in_Param2</code></td><td><code>UInt32 in_Param3</code></td><td><code>UInt32 in_Param4</code></td></tr>
<tr><td></td><td><code>HardwareSoftwareChanged</code></td><td>–</td><td>–</td><td>–</td><td>–</td></tr>
<tr><td></td><td><code>IPChanged</code></td><td>The ID of the interface</td><td>The new IP</td><td>The new subnet mask</td><td>The new default gateway</td></tr>
<tr><td>Return values</td><td colspan="5">None</td></tr>
</table>

10.3.10 Delegate_II_EREC_DT_SDRI

Description

Table 10-28 Delegate_II_EREC_DT_SDRI - .NET (C#)

Syntax	`delegate void Delegate_II_EREC_DT_SDRI (IInstance in_Sender, ERuntimeErrorCode in_ErrorCode, DateTime in_DateTime, SDataRecordInfo in_DataRecordInfo);`
Parameters	• `IInstance in_Sender:` An interface of the instance that receives this event. • `ERuntimeErrorCode in_ErrorCode:` A possible error code. • `DateTime in_DateTime:` The virtual system time of the simulated PLC instance at the time when this event was triggered. • `SDataRecordInfo in_DataRecordInfo:` The structure `SDataRecordInfo` contains the following information: – The HW identifier to which the CPU wants to write the data record – The index of the supplied data record – Size of data record – The data record
Return values	None

10.3.11 Delegate_II_EREC_DT_SDR

Description

Table 10-29 Delegate_II_EREC_DT_SDR - .NET (C#)

Syntax	`delegate void Delegate_II_EREC_DT_SDR (IInstance in_Sender, ERuntimeErrorCode in_ErrorCode, DateTime in_DateTime, SDataRecord in_DataRecord);`
Parameters	• `IInstance in_Sender:` An interface of the instance that receives this event. • `ERuntimeErrorCode in_ErrorCode:` A possible error code. • `DateTime in_DateTime:` The virtual system time of the simulated PLC instance at the time when this event was triggered. • `SDataRecord in_DataRecord:` The structure `SDataRecord` contains the following information: – The HW identifier to which the CPU writes the data record – The index of the supplied data record – Size of the data record – The data record
Return values	None

10.3.12 Delegate_SREC_ST_UINT32_EPPET_UINT32

Description

Table 10-30 Delegate_SREC_ST_UINT32_EPPET_UINT32 - .NET (C#)

Syntax	`delegate void Delegate_SREC_ST_UINT32 (IInstance in_Sender, ERuntimeErrorCode in_ErrorCode, DateTime in_DateTime, UInt32 in_HardwareIdentifier, EPullOrPlugEventType in_PullOrPlugEventType, UInt32 in_SequenceNumber);`
Parameters	• `IInstance in_Sender:` An interface of the instance that receives this event. • `ERuntimeErrorCode in_ErrorCode:` A possible error code. • `DateTime in_DateTime:` The virtual system time of the simulated PLC instance at the time when this event was triggered. • `UInt32 in_HardwareIdentifier:` The hardware identifier of the module or submodule that generated the pull/plug event. • `EPullOrPlugEventType in_PullOrPlugEventType:` A value from the list of predefined types of events for S7 modules (Page 272). • `UInt32 in_SequenceNumber:` S7-PLCSIM Advanced assigns a unique consecutive number to each interrupt event. According to the PROFINET standard, the sequence number is only 10 bits wide, from 1 to 0x7FF. When the highest number is reached the numbering starts again at 1. **Note** In a real hardware system the IO controller uses the sequence number to check if it has lost a hardware interrupt. During the simulation, the sequence number creates the relation between interrupt request and the associated acyclic alarm.
Return values	None

10.3.13 Delegate_SREC_ST_UINT32_UINT32_EPET_UINT32

Description

Table 10-31 Delegate_SREC_ST_UINT32_UINT32_EPET_UINT32 - Native C++

Syntax	`delegate void Delegate_SREC_ST_UINT32_UINT32_EPET_UINT32(IInstance in_Sender, ERuntimeErrorCode in_ErrorCode, DateTime in_DateTime, UInt32 in_HardwareIdentifier, UInt32 in_Channel, EProcessEventType in_ProcessEventType, UInt32 in_SequenceNumber);`
Parameters	• `IInstance in_Sender:` An interface of the instance that receives this event. • `ERuntimeErrorCode in_ErrorCode:` A possible error code. • `DateTime in_DateTime:` The virtual system time of the simulated PLC instance at the time when this event was triggered. • `UInt32 in_HardwareIdentifier:` The hardware identifier of the module or submodule that generates the process event. • `UInt32 in_Channel:` The channel of the I/O module which sends the process event. • `EProcessEventType in_ProcessEventType:` A value from the list of predefined types of events for S7 modules, see EProcessEventType (Page 272).

	• `UInt32 in_SequenceNumber:` S7-PLCSIM Advanced assigns a unique consecutive number to each interrupt event. According to PROFINET standard the sequence number is 10 bits wide (1 to $7FF_H$). When the highest number is reached the numbering starts again at 1. **Note** In a real hardware system the I/O controller uses the sequence number to check if it has lost a hardware interrupt. During the simulation, the sequence number creates the relation between interrupt request and the associated acyclic alarm.
Return values	None

10.3.14 Delegate_SREC_ST_UINT32

Description

Table 10-32 Delegate_SREC_ST_UINT32 - .NET (C#)

Syntax	`delegate void Delegate_SREC_ST_UINT32 (IInstance in_Sender, ERuntimeErrorCode in_ErrorCode, DateTime in_DateTime, UInt32 in_HardwareIdentifier);`
Parameters	• `IInstance in_Sender:` An interface of the instance that receives this event. • `ERuntimeErrorCode in_ErrorCode:` A possible error code. • `DateTime in_DateTime:` The virtual system time of the simulated PLC instance at the time when this event was triggered. • `UInt32 in_HardwareIdentifier:` The ID of the module that generates the status event, update event, or profile event.
Return values	None

10.3.15 Delegate_SREC_ST_UINT32_UINT32

Description

Table 10-33 Delegate_SREC_ST_UINT32_UINT32 - .NET (C#)

Syntax	`delegate void Delegate_SREC_ST_UINT32 (IInstance in_Sender, ERuntimeErrorCode in_ErrorCode, DateTime in_DateTime, UInt32 in_HardwareIdentifier, UInt32 in_SequenceNumber);`
Parameters	• `IInstance in_Sender:` An interface of the instance that receives this event. • `ERuntimeErrorCode in_ErrorCode:` A possible error code. • `DateTime in_DateTime:` The virtual system time of the simulated PLC instance at the time when this event was triggered. • `UInt32 in_HardwareIdentifier:` The hardware identifier of the module or submodule which sends the diagnostics entry. • `UInt32 in_SequenceNumber:` S7-PLCSIM Advanced assigns a unique consecutive number to each interrupt event. According to PROFINET standard the sequence number is 10 bits wide (1 to $7FF_H$). When the highest number is reached the numbering starts again at 1. **Note** In a real hardware system the IO controller uses the sequence number to check if it has lost a hardware interrupt.

	During the simulation, the sequence number creates the relation between interrupt request and the associated acyclic alarm.
Return values	None

10.3.16 Delegate_SREC_ST_UINT32_ERSFET

Description

Table 10-34 Delegate_SREC_ST_UINT32_ERSFET - .NET (C#)

Syntax	`delegate void Delegate_SREC_ST_UINT32_ERSFET(IInstance in_Sender, ERuntimeErrorCode in_ErrorCode, DateTime in_DateTime, UInt32 in_HardwareIdentifier, ERackOrStationFaultType in_EventType);`
Parameters	• `IInstance in_Sender:` An interface of the instance that receives this event. • `ERuntimeErrorCode in_ErrorCode:` A possible error code. • `DateTime in_DateTime:` The virtual system time of the simulated PLC instance at the time when this event was triggered. • `UInt32 in_HardwareIdentifier:` The hardware identifier of the module or submodule which sends the diagnostics entry. • `ERackOrStationFaultType in_EventType:` A value from the list of predefined RackOrStationFault event types (Page 274).
Return values	None

10.4 Definitions and constants

The following identifiers are used in the API:

Table 10-35 Definitions - Native C++

Identifier	Value	Description
DINSTANCE_NAME_MAX_LENGTH	64	The unique name of an instance must be less than this value.
DSTORAGE_PATH_MAX_LENGTH	130	The maximum path length to the virtual memory card. Including NULL termination.
DTAG_NAME_MAX_LENGTH	300	The maximum length of the name of a PLC tag. Including NULL termination.
DTAG_ARRAY_DIMENSION	6	The maximum number of dimensions for a multi-dimensional field.
DCONTROLLER_NAME_MAX_LENGTH	128	The maximum length of the controller name. Including NULL termination.
DCONTROLLER_SHORT_DESIGNATION_MAX_LENGTH	32	The maximum length of the abbreviation of the controller (CPU type). Including NULL termination.
DALARM_NOTIFICATION_MAX_DIAG_EVENTS	100	The maximum number of diagnostic events that are sent in a diagnostic alarm.
DPROCESS_EVENT_NAME_MAX_LENGTH	123	The maximum length of the name for the process event. Including NULL termination.
DPROCESS_EVENTS_MAX_ITEMS	256	The maximum number of configurable process events.
DMODULE_STATE_OK	0	AlarmNotification: Module status OK

Identifier	Value	Description
DMODULE_STATE_ERROR	1	AlarmNotification: Module status faulty
DMODULE_STATE_MAINT_DEMANDED	2	AlarmNotification: Maintenance demanded
DMODULE_STATE_MAINT_REQUIRED	4	AlarmNotification: Maintenance required

Table 10-36 Constants - .NET (C#)

Identifier	Value	Description
RuntimeConstants.InstanceNameLength	64	The unique name of an instance must be less than this value.
RuntimeConstants.StoragePath-MaxLength	130	The maximum path length to the virtual memory card. Including NULL termination.
RuntimeConstants.TagNameMaxLength	300	The maximum length of the name of a PLC tag. Including NULL termination.
RuntimeConstants.TagArrayDimension	6	The maximum number of dimensions for a multi-dimensional field.
RuntimeConstants.Controller-NameMaxLength	128	The maximum length of the controller name. Including NULL termination.
RuntimeConstants.ControllerShortDes-ignationMaxLength	32	The maximum length of the abbreviation of the controller (CPU type). Including NULL termination.
ModuleState.Ok	0	AlarmNotification: Module status OK
ModuleState.Error	1	AlarmNotification: Module status faulty
ModuleState. MaintenanceDemanded	2	AlarmNotification: Maintenance demanded
ModuleState. MaintenanceRequired	4	AlarmNotification: Maintenance required

10.5 Unions (Native C++)

10.5.1 UIP

Description

Contains an IPv4 address.

Table 10-37 UIP - Native C++

Syntax	`union UIP`
	`{`
	`DWORD IP;`
	`BYTE IPs[4];`
	`};`

Member	• `DWORD IP:` The IP address in a single DWORD • `BYTE IPs[4]:` The four elements of IP in descending order
Example	Example for an IP address: 192.168.0.1 UIP.IP = 0xC0A80001 UIP.IPs[3] = 192, UIP.IPs[2] = 168, UIP.IPs[1] = 0, UIP.IPs[0] = 1

10.5.2 UDataValue

Description

Contains the value of a PLC tag.

Table 10-38 UDataValue - Native C++

Syntax	`union UDataValue`
	`{`
	`bool Bool;`
	`INT8 Int8;`
	`INT16 Int16;`
	`INT32 Int32;`
	`INT64 Int64;`
	`UINT8 UInt8;`
	`UINT16 UInt16;`
	`UINT32 UInt32;`
	`UINT64 UInt64;`
	`float Float;`
	`double Double;`
	`CHAR Char;`
	`WCHAR WChar;`
	`};`

Member	• `bool Bool`: 1 byte boolean value • `INT8 Int8`: 1 byte integer with sign • `INT16 Int16`: 2 byte integer with sign • `INT32 Int32`: 4 byte integer with sign • `INT64 Int64`: 8 byte integer with sign • `UINT8 UInt8`: 1 byte integer without sign • `UINT16 UInt16`: 2 byte integer without sign • `UINT32 UInt32`: 4 byte integer without sign • `UINT64 UInt64`: 8 byte integer without sign • `float Float`: 4 byte floating-point value • `double Double`: 8 byte floating-point value • `CHAR Char`: 1 byte value character • `WCHAR WChar`: 2 byte value character

10.6 Structures

The following structures are available:

- SDataValue (Page 240)
- SDVBNI (Page 241)
- SDataValueByAddress (Page 242)
- SDataValueByAddressWithCheck (Page 243)
- SDataValueByName (Page 244)
- SDataValueByNameWithCheck (Page 245)
- SConnectionInfo (Page 245)
- SInstanceInfo (Page 246)
- SDimension (Page 246)
- STagInfo (Page 247)
- SIP (Page 248)
- SIPSuite4 (Page 249)
- SOnSyncPointReachedResult (Page 249)
- SDataRecordInfo (Page 251)
- SDataRecord (Page 252)
- SConfiguredProcessEvents (Page 252)
- SDiagExtChannelDescription (Page 253)
- SAutodiscoverData (Page 254)

10.6.1 SDataValue

Description

The structure contains the value and type of a PLC tag.

Table 10-39 SDataValue - Native C++

Syntax	`struct SDataValue`
	`{`
	`UDataValue Value;`
	`EPrimitiveDataType Type;`
	`};`

Member	• `UDataValue Value:` The value of the PLC tags • `EPrimitiveDataType Type:` Type of PLC tag

Table 10-40 SDataValue - .NET (C#)

Syntax	`struct SDataValue`
	`{`
	`bool Bool { get; set; }`
	`Int8 Int8 { get; set; }`
	`Int16 Int16 { get; set; }`
	`Int32 Int32 { get; set; }`
	`Int64 Int64 { get; set; }`
	`UInt8 UInt8 { get; set; }`
	`UInt16 UInt16 { get; set; }`
	`UInt32 UInt32 { get; set; }`
	`UInt64 UInt64 { get; set; }`
	`float Float { get; set; }`
	`double Double { get; set; }`
	`sbyte Char { get; set; }`
	`char WChar { get; set; }`
	`EPrimitiveDataType Type { get; set; }`
	`}`

Member	• `bool Bool:` 1 byte boolean value • `Int8 Int8:` 1 byte integer with sign • `Int16 Int16:` 2 byte integer with sign • `Int32 Int32:` 4 byte integer with sign • `Int64 Int64:` 8 byte integer with sign • `UntT8 UInt8:` 1 byte integer without sign • `UInt16 UInt16:` 2 byte integer without sign • `UInt32 UInt32:` 4 byte integer without sign • `UInt64 UInt64:` 8 byte integer without sign • `float Float:` 4 byte floating-point value • `double Double:` 8 byte floating-point value • `sbyte Char:` 1 byte value character • `char WChar:` 2 byte value character • `EPrimitiveDataType Type:` Type of PLC tag

10.6.2 SDVBNI

Description

This structure is for internal use only. Do not change this structure.

Table 10-41 SDVBNI - Native C++

Syntax	`struct SDVBNI`

Table 10-42 SDVBNI - .NET (C#)

Syntax	`struct SDVBNI`

10.6.3 SDataValueByAddress

Description

This structure represents a PLC tag that is accessed via its address.

Table 10-43 SDataValueByAddress - Native C++

Syntax	`struct SDataValueByAddress`
	`{`
	`UINT32 Offset;`
	`UINT8 Bit;`
	`SDataValue DataValue;`
	`ERuntimeErrorCode ErrorCode;`
	`}`

Member	• `UINT32 Offset` The byte offset of the tag if it is not located in a data block. • `UINT8 Bit` The bit offset of the tag if it is not located in a data block. • `SDataValue DataValue` The value of the PLC tags • `ERuntimeErrorCode ErrorCode` – `SREC_TIMEOUT` if no event was triggered during the defined time interval. – `SREC_WARNING_INVALID_CALL` if no function RegisterOnSyncPointReachedEvent was called before, see `ERuntimeErrorCode`.

Table 10-44 SDataValueByAddress - .NET (C#)

Syntax	`struct SDataValueByAddress`
	`{`
	`UInt32 Offset;`
	`UInt8 Bit;`
	`SDataValue DataValue;`
	`ERuntimeErrorCode ErrorCode;`
	`}`

Member	• `UINT32 Offset` The byte offset of the tag if it is not located in a data block. • `UINT8 Bit` The bit offset of the tag if it is not located in a data block. • `SDataValue DataValue` The value of the PLC tags • `ERuntimeErrorCode ErrorCode` – `SREC_TIMEOUT` if no event was triggered during the defined time interval. – `SREC_WARNING_INVALID_CALL` if no function RegisterOnSyncPointReachedEvent was called before, see `ERuntimeErrorCode`.

10.6.4 SDataValueByAddressWithCheck

Description

This structure represents a PLC tag that is accessed via its address.

Table 10-45 SDataValueByAddressWithCheck - Native C++

Syntax	`struct SDataValueByAddressWithCheck`
	`{`
	`UINT32 Offset;`
	`UINT8 Bit;`
	`SDataValue DataValue;`
	`ERuntimeErrorCode ErrorCode;`
	`bool ValueHasChanged;`
	`}`

Member	• `UINT32 Offset` The byte offset of the tag if it is not located in a data block. • `UINT8 Bit` The bit offset of the tag if it is not located in a data block. • `SDataValue DataValue` The value of the PLC tags • `ERuntimeErrorCode ErrorCode` – `SREC_TIMEOUT` if no event was triggered during the defined time interval. – `SREC_WARNING_INVALID_CALL` if no function RegisterOnSyncPointReachedEvent was called before, see `ERuntimeErrorCode`. • `bool ValueHasChanged` _____

Table 10-46 SDataValueByAddressWithCheck - .NET (C#)

Syntax	`struct SDataValueByAddressWithCheck`
	`{`
	`UInt32 Offset;`
	`UInt8 Bit;`
	`SDataValue DataValue;`
	`ERuntimeErrorCode ErrorCode;`
	`bool ValueHasChanged;`
	`}`

Member	• `UINT32 Offset` The byte offset of the tag if it is not located in a data block. • `UINT8 Bit` The bit offset of the tag if it is not located in a data block. • `SDataValue DataValue` The value of the PLC tags • `ERuntimeErrorCode ErrorCode` – `SREC_TIMEOUT` if no event was triggered during the defined time interval. – `SREC_WARNING_INVALID_CALL` if no function RegisterOnSyncPointReachedEvent was called before, see `ERuntimeErrorCode`. • `bool ValueHasChanged` _____

10.6.5 SDataValueByName

Description

This structure represents a PLC tag that is called by its name.

Table 10-47 SDataValueByName - Native C++

Syntax	`struct SDataValueByName`
	`{`
	`WCHAR Name[DTAG_NAME_MAX_LENGTH];`
	`SDataValue DataValue;`
	`ERuntimeErrorCode ErrorCode;`
	`SDVBNI Internal;`
	`}`

Member	• `WCHAR Name[DTAG_NAME_MAX_LENGTH]` The name of the tag • `SDataValue DataValue` The value of the PLC tags • `ERuntimeErrorCode ErrorCode` – `SREC_TIMEOUT` if no event was triggered during the defined time interval. – `SREC_WARNING_INVALID_CALL` if no function RegisterOnSyncPointReachedEvent was called before, see `ERuntimeErrorCode`. • `SDVBNI Internal` _____

Table 10-48 SDataValueByName - .NET (C#)

Syntax	`struct SDataValueByName`
	`{`
	`String Name;`
	`SDataValue DataValue;`
	`ERuntimeErrorCode ErrorCode;`
	`SDVBNI Internal;`
	`}`

Member	• `String Name` The name of the tag • `SDataValue DataValue` The value of the PLC tags • `ERuntimeErrorCode ErrorCode` – `SREC_TIMEOUT` if no event was triggered during the defined time interval. – `SREC_WARNING_INVALID_CALL` if no function RegisterOnSyncPointReachedEvent was called before, see `ERuntimeErrorCode`. • `SDVBNI Internal` _____

10.6.6 SDataValueByNameWithCheck

Description

This structure represents a PLC tag that is called by its name.

Table 10-49 SDataValueByNameWithCheck - Native C++

Syntax	`struct SDataValueByNameWithCheck`
	`{`
	`WCHAR Name[DTAG_NAME_MAX_LENGTH];`
	`SDataValue DataValue;`
	`ERuntimeErrorCode ErrorCode;`
	`SDVBNI Internal;`
	`bool ValueHasChanged;`
	`};`

Table 10-50 SDataValueByNameWithCheck - .NET (C#)

Syntax	`struct SDataValueByNameWithCheck`
	`{`
	`String Name;`
	`SDataValue DataValue;`
	`ERuntimeErrorCode ErrorCode;`
	`SDVBNI Internal;`
	`bool ValueHasChanged;`
	`}`

10.6.7 SConnectionInfo

Description

This structure contains the IP address and port of a TCP/IP connection.

NOTE

For a Software/Open controller instance, the IP address must be the same as in the TIA Portal project.

In the device configuration in the TIA Portal, select "Set IP address in the project" and configure a static IP address to the PLC Interface. Do not assign IP addresses with 0.0.0.0 and 192.168.0.1 values.

When you download to your instance, select the appropriate "Direct at slot 1 Xn" variant in the Connection to interface/subnet dropdown in the Extended download to device window. The TIA Portal Load Preview window displays an error message if an IP Address mismatch occurs. In this case, you must cancel the download and recreate your instance.

Table 10-51 SConnectionInfo - Native C++

Syntax	`struct SConnectionInfo`
	`{`
	`UIP IP;`
	`UINT16 Port;`
	`};`

Table 10-52 SConnectionInfo - .NET (C#)

Syntax	`struct SConnectionInfo`
	`{`
	`SIP IP;`
	`UInt16 Port;`
	`}`

10.6.8 SInstanceInfo

Description

This structure contains an IPv4 address.

Table 10-53 SInstanceInfo - Native C++

Syntax	`struct SInstanceInfo`
	`{`
	`INT32 ID;`
	`WCHAR Name[DINSTANCE_NAME_MAX_LENGTH];`
	`};`

Member	• `INT32 ID:` The ID of the instance • `WCHAR Name[DINSTANCE_NAME_MAX_LENGTH]:` The name of the instance

Table 10-54 SInstanceInfo - .NET (C#)

Syntax	`struct SInstanceInfo`
	`{`
	`Int32 ID;`
	`String Name;`
	`}`

Member	• `Int32 ID:` The ID of the instance • `String name:` The name of the instance

10.6.9 SDimension

Description

This structure contains information about the dimension of a field.

Table 10-55 SDimension - Native C++

Syntax	`struct SDimension`
	`{`
	`INT32 StartIndex;`
	`UINT32 Count;`
	`};`

Table 10-56 SDimension - .NET (C#)

Syntax	`struct SDimension`
	`{`
	`Int32 StartIndex;`
	`UInt32 Count;`
	`}`

10.6.10 STagInfo

Description

This structure contains information about a PLC tag.

Table 10-57 STagInfo - Native C++

Syntax	`struct STagInfo`
	`{`
	`WCHAR Name[DTAG_NAME_MAX_LENGTH];`
	`EArea Area;`
	`EDataType DataType;`
	`EPrimitiveDataType PrimitiveDataType;`
	`UINT16 Size;`
	`UINT32 Offset;`
	`UINT8 Bit;`
	`UINT8 DimensionCount;`
	`UINT32 Index;`
	`UINT32 ParentIndex;`
	`SDimension Dimension[DTAG_ARRAY_DIMENSION];`
	`};`

Member	• `WCHAR Name[DTAG_NAME_MAX_LENGTH]:` The name of the tag • `EArea area:` The CPU area where the tag is located. • `EDataType DataType:` The CPU data type of the tag • `EPrimitiveDataType PrimitiveDataType:` The primitive data type of the tag • `UINT16 size:` The size of the tag in bytes • `UINT32 offset:` The byte offset of the tag if it is not located in a data block. • `UINT8 bit:` The bit offset of the tag if it is not located in a data block. • `UINT8 DimensionCount:` The number of dimensions of the array. 0 if the tag is not a field. • `UINT32 index:` The index of the tag • `UINT32 ParentIndex:` If this tag is embedded in another tag (for example, an element of a structure), this value then displays the index of the parent tag. The value is 0 if the tag has no parent tag. • `SDimension Dimension[DTAG_ARRAY_DIMENSION]:` Information about each dimension of the field

Table 10-58 STagInfo - .NET (C#)

Syntax	`public struct STagInfo`
	`{`
	`String Name;`
	`EArea Area;`
	`EDataType DataType;`
	`EPrimitiveDataType PrimitiveDataType;`
	`UInt16 Size;`
	`UInt32 Offset;`
	`UInt8 Bit;`
	`UInt32 Index;`
	`UInt32 ParentIndex;`
	`SDimension[] Dimension;`
	`}`

Member	• `String name:` The name of the tag • `EArea area:` The CPU area where the tag is located. • `EDataType DataType:` The CPU data type of the tag • `EPrimitiveDataType PrimitiveDataType:` The primitive data type of the tag • `UInt16 size:` The size of the tag in bytes. • `UInt32 offset:` The byte offset of the tag if it is not located in a data block. • `UInt8 bit:` The bit offset of the tag if it is not located in a data block. • `UInt32 index:` The index of the tag • `UInt32 ParentIndex:` If this tag is embedded in another tag (for example, an element of a structure), this value then displays the index of the parent tag. The value is 0 if the tag has no parent tag. • `SDimension[] Dimension:` Information about each dimension of the field. Empty, if the tag is not an array.

10.6.11 SIP

Description

This structure contains an IPv4 address.

Table 10-59 SIP - .NET (C#)

Syntax	`struct SIP`
	`{`
	`byte[] IPArray { get; set; }`
	`UInt32 IPDWord { get; set; }`
	`string IPString { get; set; }`
	`}`

Member	• `UInt32 IPDWord:` The IP address in a single DWORD • `byte[] IPArray:` The four elements of IP in descending order • `string IPString:` The IPv4 address as a string
Example	Example for an IP address: 192.168.0.1 SIP.IPDWord = 0xC0A80001 SIP.IPArray[3] = 192, SIP.IPArray[2] = 168, SIP.IPArray[1] = 0, SIP.IPArray[0] = 1 SIP.IPString = "192.168.0.1"

10.6.12 SIPSuite4

Description

This structure contains an IPv4 suite.

Table 10-60 SIPSuite4 - Native C++

Syntax	`struct SIPSuite4`
	`{`
	`UIP IPAddress;`
	`UIP SubnetMask;`
	`UIP DefaultGateway;`
	`};`

Member	• `UIP IPAddress:` The IP address • `UIP SubnetMask:` The subnet mask • `UIP DefaultGateway:` The standard gateway

Table 10-61 SIPSuite4 - .NET (C#)

Syntax	`struct SIPSuite4`
	`{`
	`SIP IPAddress;`
	`SIP SubnetMask;`
	`SIP DefaultGateway;`
	`}`

Member	• `SIP IPAddress:` The IP address • `SIP SubnetMask:` The subnet mask • `SIP DefaultGateway:` The standard gateway

10.6.13 SOnSyncPointReachedResult

Description

This structure contains the results of the OnSyncPointReached event.

Table 10-62 SOnSyncPointReachedResult - Native C++

Syntax	`struct SOnSyncPointReachedResult`
	`{`
	`ERuntimeErrorCode ErrorCode;`
	`SYSTEMTIME SystemTime;`
	`UINT32 Id;`
	`INT64 TimeSinceSameSyncPoint_ns;`
	`INT64 TimeSinceAnySyncPoint_ns;`
	`UINT32 SyncPointCount;`
	`};`

Member	• `ERuntimeErrorCode ErrorCode:` – `SREC_TIMEOUT` if no event was triggered during the defined time interval. In such a case, the struct values are not used. – `SREC_WARNING_INVALID_CALL`, if no function `RegisterOnSyncPointReachedEvent` was called before. See ERuntimeErrorCode. • `SYSTEMTIME SystemTime:` The virtual system time of the simulated PLC instance at the time when this event was triggered.

	• `UINT32 Id:` For the `SingleStep_Bus` operating mode: The ID of the I/O system that is used for cycle-controlled synchronization of the simulated PLC instance. For all other operating modes: The ID of the process image partition (PIP) that triggers this event. 0 for the cycle control point (End of cycle). Note: In the SingleStep operating modes, the transferred ID for the cycle control point has the value 0xFFFF. In this case, the time values are set to 0. • `INT64 TimeSinceSameSyncPoint_ns:` The virtual time (in nanoseconds) since the last synchronization point of the same process image partition ID was reached. For the time-controlled operating modes: The runtime since the last call of the `StartProcessing()` function. • `INT64 TimeSinceAnySyncPoint_ns:` The virtual time (in nanoseconds) since the last synchronization point of any process image partition ID was reached. For the time-controlled operating modes: The runtime since the last call of the `StartProcessing()` function. • `UINT32 SyncPointCount:` The number of synchronization points since the last event. If the events are triggered faster than they are received, multiple events are combined into one event. In this case, this value contains the number of cycles since the last event was received.

Table 10-63 SOnSyncPointReachedResult - .NET (C#)

Syntax	`struct SOnSyncPointReachedResult`
	`{`
	`ERuntimeErrorCode ErrorCode;`
	`DateTime SystemTime;`
	`UInt32 Id;`
	`Int64 TimeSinceSameSyncPoint_ns;`
	`Int64 TimeSinceAnySyncPoint_ns;`
	`UInt32 SyncPointCount;`
	`}`

Member	• `ERuntimeErrorCode ErrorCode:` – `ERuntimeErrorCode.Timeout`, if no event was triggered during the defined time interval. – `WarningInvalidCall`, if no function `RegisterOnSyncPointReachedEvent` was called before. See ERuntimeErrorCode. • `DateTime DateTime:` The virtual system time of the simulated PLC instance at the time when this event was triggered. • `UInt32 Id:` For the `SingleStep_Bus` operating mode: The ID of the I/O system that is used for cycle-controlled synchronization of the simulated PLC instance. For all other operating modes: The ID of the process image partition (PIP) that triggers this event. 0 for the cycle control point (End of cycle). • `Int64 TimeSinceSameSyncPoint_ns:` The virtual time (in nanoseconds) since the last synchronization point of the same process image partition ID was reached. For the time-controlled operating modes: The runtime since the last call of the `StartProcessing()` function.

	• `Int64 TimeSinceAnySyncPoint_ns:` The virtual time (in nanoseconds) since the last synchronization point of any process image partition ID was reached. For the time-controlled operating modes: The runtime since the last call of the `StartProcessing()` function. • `UInt32 SyncPointCount:` The number of synchronization points since the last event. If the events are triggered faster than they are received, multiple events are combined into one event. In this case, this value contains the number of cycles since the last event was received.

10.6.14 SDataRecordInfo

Description

This structure contains read/write data record information.

Table 10-64 SDataRecordInfo - Native C++

Syntax	`struct SDataRecordInfo`
	`{`
	`UINT32 HardwareId;`
	`UINT32 RecordIdx;`
	`UINT32 DataSize;`
	`};`

Member	• `UINT32 HardwareId:` The ID of the hardware module (hardware identifier) • `UINT32 RecordIdx:` The data record number • `UINT32 DataSize:` The data record size

Table 10-65 SDataRecordInfo - .NET (C#)

Syntax	`struct SDataRecordInfo`
	`{`
	`UInt32 HardwareId;`
	`UInt32 RecordIdx;`
	`UInt32 DataSize;`
	`}`

Member	• `UInt32 ID:` The ID of the hardware module • `UInt32 RecordIdx:` The data record number • `UInt32 DataSize:` The data record size

10.6.15 SDataRecord

Description

This structure contains read/write data record information and data records.

Table 10-66 SDataRecord - .NET (C#)

Syntax	`struct SDataRecord`
	`{`
	`UInt32 HardwareId;`
	`byte[] Data`
	`}`

Member	• `SDataRecordInfo Info:` The data record information, see SDataRecordInfo (Page 251) • `byte[] Data:` The array length

10.6.16 SConfiguredProcessEvents

Description

This structure contains information about the configured process events.

Table 10-67 SConfiguredProcessEvents - Native C++

Syntax	`struct SConfiguredProcessEvents`
	`{`
	`UINT16 HardwareIdentifier;`
	`UINT16 Channel;`
	`EProcessEventType ProcessEventType;`
	`WCHAR Name[DPROCESS_EVENT_NAME_MAX_LENGTH];`
	`};`

Member	• `UINT16 HardwareIdentifier:` The HW identifier • `UINT16 Channel:` The channel of the I/O module which generates the process event. • `EProcessEventType ProcessEventType:` The type of the configured process event • `WCHAR Name[DPROCESS_EVENT_NAME_MAX_LENGTH]:` The name of the process event

Table 10-68 SConfiguredProcessEvents - .NET (C#)

Syntax	`public struct SConfiguredProcessEvents`
	`{`
	`ushort HardwareIdentifier;`
	`ushort Channel;`
	`EProcessEventType ProcessEventType;`
	`string Name;`
	`}`

Member	• `ushort HardwareIdentifier:` The HW identifier • `ushort Channel:` The channel of the I/O module which generates the process event. • `EProcessEventType ProcessEventType:` The type of the configured process event • `String name:` The name of the process event

10.6.17 SDiagExtChannelDescription

Description

This structure contains read/write data record information and data records.

Table 10-69 SDiagExtChannelDescription - Native C++

Syntax	`struct SDiagExtChannelDescription`
	`{`
	`UINT16 ChannelNumber;`
	`UINT16 ErrorType;`
	`UINT16 ExtErrorType;`
	`EDiagSeverity Severity;`
	`EDiagProperty Direction;`
	`};`

Member	• `UINT16 ChannelNumber:` If the interrupt relates to a specific channel of the IO device (e.g. short circuit), this parameter must contain the number of the faulty channel. If the interrupt was generated by a module or submodule, the number of the channel must be set to 0x8000. • `UINT16 ErrorType:` The parameter defines error types according to PROFINET standard, see "Error types" section. • `EDiagSeverity Severity:` The value of the severity for the diagnostics, see EDiagSeverity (Page 274). • `EDiagProperty Direction:` The value for the incoming/outgoing information, see EDiagProperty (Page 273). • `UINT16 ExtErrorType:` This parameter provides the option of defining more details for the diagnostic interrupt. This is helpful in combination with PDEV error types which are generated for CPU-internal modules. Should be 0 by default.

Table 10-70 SDiagExtChannelDescription - .NET (C#)

Syntax	`struct SDiagExtChannelDescription`
	`{`
	`UInt16 ChannelNumber;`
	`UInt16 ErrorType;`
	`UInt16 ExtErrorType;`
	`EDiagSeverity Severity;`
	`EDiagProperty Direction;`
	`};`

Member	• `UInt16 ChannelNumber:` If the interrupt relates to a specific channel of the IO device (e.g. short circuit), this parameter must contain the number of the faulty channel. If the interrupt was generated by a module or submodule, the number of the channel must be set to 0x8000. • `UInt16 ErrorType:` The parameter defines error types according to PROFINET standard, see "Error types" section. • `EDiagSeverity Severity:` The value of the severity for the diagnostics, see EDiagSeverity (Page 274). • `EDiagProperty Direction:` The value for the incoming/outgoing information, see EDiagProperty (Page 273). • `UInt16 ExtErrorType:` This parameter provides the option of defining more details for the diagnostic interrupt. This is helpful in combination with PDEV error types which are generated for CPU-internal modules. Should be 0 by default.

Error types

The following table contains important error types (`ErrorType`) according to PROFINET standard:

Table 10-71 Error types according to PROFINET standard

Value	Meaning
0x0000	Reserved / unknown error
0x0001	Short-circuit
0x0002	Undervoltage
0x0003	Overvoltage
0x0004	Overload
0x0005	Overtemperature
0x0006	Wire break
0x0007	High limit violated
0x0008:	Low limit violated
0x0009	Error

The following table contains error types `ExtChannelErrorType` for `ChannelErrorType` "Remote mismatch":

Table 10-72 ExtChannelErrType error types

Value	Meaning	Use
0x0000	Reserved	-
0x0001 to 0x7FFF	Manufacturer ID	Interrupt/diagnostics
0x8000	Peer name of station mismatch	Interrupt/diagnostics
0x8001	Peer name of port mismatch	Interrupt/diagnostics
0x8002	Peer RT_CLASS_3 mismatch	Interrupt/diagnostics
0x8003	Peer MAU Type mismatch	Interrupt/diagnostics

10.6.18 SAutodiscoverData

Description

This structure contains the IP address, the port, the Runtime version, and the name of the computer that has a Runtime Manager ready to make a remote connection.

Table 10-73 SAutodiscoverData - Native C++

Syntax	`public struct SAutodiscoverData`
	`{`
	`UIP IP;`
	`UINT16 Port;`
	`DWORD RuntimeVersion;`
	`WCHAR ComputerName[MAX_COMPUTERNAME_LENGTH + 1];`
	`};`

Table 10-74 SAutodiscoverData - .NET (C#)

Syntax	`public struct SAutodiscoverData`
	`{`
	`public SIP IP;`
	`public ushort Port;`
	`public uint RuntimeVersion;`
	`public string ComputerName;`
	`}`

10.6.19 SMAC address

Description

Defines the structure of a MAC address.

Table 10-75 SMACAddress - Native C++

Syntax	`#define DMAC_ADDRESS_LEN 6`
	
	`struct SMACAddress`
	`{`
	`BYTE Octets[DMAC_ADDRESS_LEN];`
	`};`

10.6.20 SNetInterfaceInfo

Description

Describes the structure of a network interface.

Table 10-76 SNetInterfaceInfo - Native C++

Syntax	`struct SNetInterfaceInfo`
	`{`
	`DWORD interfaceIndex;`
	`UMACAddress MACAddress;`
	`WCHAR interfaceName[64];`
	`WCHAR interfaceDescription[64];`
	`};`

10.7 Enumerations

The following enumerations are available:

- ERuntimeErrorCode (Page 256)
- EArea (Page 259)
- EOperatingState (Page 259)
- EOperatingMode (Page 260)
- ECPUType (Page 261)
- ECommunicationInterface (Page 263)
- ENetworkMode (Page 263)
- EPLCInterface (Page 264)

- ELEDType (Page 264)
- ELEDMode (Page 265)
- EPrimitiveDataType (Page 265)
- EDataType (Page 267)
- ETagListDetails (Page 270)
- ERuntimeConfigChanged (Page 271)
- EInstanceConfigChanged (Page 271)
- EPullOrPlugEventType (Page 272)
- EProcessEventType (Page 272)
- ESoftwareConfigChanged (Page 273)
- EDirection (Page 273)
- EDiagProperty (Page 273)
- EDiagSeverity (Page 274)
- ERackOrStationFaultType (Page 274)
- ECycleTimeMonitoringMode (Page 274)
- EAutodiscoverType (Page 275)

10.7.1 ERuntimeErrorCode

Description

This enumeration contains all error codes that are used by the Simulation Runtime API. Most API functions return one of these error codes. If the function is successful, the return value is always `SREC_OK` / `OK`. Errors are returned with negative values, warnings with positive values.

Table 10-77 ERuntimeErrorCode - Native C++

Syntax	`enum ERuntimeErrorCode`
	`{`
	`SREC_OK = 0,`
	`SREC_INVALID_ERROR_CODE = -1,`
	`SREC_NOT_IMPLEMENTED = -2,`
	`SREC_INDEX_OUT_OF_RANGE = -3,`
	`SREC_DOES_NOT_EXIST = -4,`
	`SREC_ALREADY_EXISTS = -5,`
	`SREC_UNKNOWN_MESSAGE_TYPE = -6,`
	`SREC_INVALID_MESSAGE_ID = -7,`
	`SREC_WRONG_ARGUMENT = -8,`
	`SREC_WRONG_PIPE = -9,`
	`SREC_CONNECTION_ERROR = -10,`
	`SREC_TIMEOUT = -11,`
	`SREC_MESSAGE_CORRUPT = -12,`
	`SREC_WRONG_VERSION = -13,`
	`SREC_INSTANCE_NOT_RUNNING = -14,`
	`SREC_INTERFACE_REMOVED = -15,`
	`SREC_SHARED_MEMORY_NOT_INITIALIZED = -16,`
	`SREC_API_NOT_INITIALIZED = -17,`
	`SREC_WARNING_ALREADY_EXISTS = 18,`
	`SREC_NOT_SUPPORTED = -19,`
	`SREC_WARNING_INVALID_CALL = 20,`
	`SREC_ERROR_LOADING_DLL = -21,`

```
SREC_SIGNAL_NAME_DOES_NOT_EXIST = -22,
SREC_SIGNAL_TYPE_MISMATCH = -23,
SREC_SIGNAL_CONFIGURATION_ERROR = -24,
SREC_NO_SIGNAL_CONFIGURATION_LOADED = -25,
SREC_CONFIGURED_CONNECTION_NOT_FOUND = -26,
SREC_CONFIGURED_DEVICE_NOT_FOUND = -27,
SREC_INVALID_CONFIGURATION = -28,
SREC_TYPE_MISMATCH = -29,
SREC_LICENSE_NOT_FOUND = -30,
SREC_NO_LICENSE_AVAILABLE = -31,
SREC_WRONG_COMMUNICATION_INTERFACE = -32,
SREC_LIMIT_REACHED = -33,
SREC_NO_STORAGE_PATH_SET = -34,
SREC_STORAGE_PATH_ALREADY_IN_USE = -35,
SREC_MESSAGE_INCOMPLETE = -36,
SREC_ARCHIVE_STORAGE_NOT_CREATED = -37,
SREC_RETRIEVE_STORAGE_FAILURE = -38,
SREC_INVALID_OPERATING_STATE = -39,
SREC_INVALID_ARCHIVE_PATH = -40,
SREC_DELETE_EXISTING_STORAGE_FAILED = -41,
SREC_CREATE_DIRECTORIES_FAILED = -42,
SREC_NOT_ENOUGH_MEMORY = -43,
SREC_WARNING_TRIAL_MODE_ACTIVE = 44,
SREC_NOT_RUNNING = -45,
SREC_NOT_EMPTY = -46,
SREC_NOT_UP_TO_DATE = -47,
SREC_COMMUNICATION_INTERFACE_NOT_AVAILABLE = -48,
SREC_WARNING_NOT_COMPLETE = 49,
SREC_VIRTUAL_SWITCH_MISCONFIGURED = -50,
SREC_RUNTIME_NOT_AVAILABLE = -51,
SREC_IS_EMPTY = -52,
SREC_WRONG_MODULE_STATE = -53,
SREC_WRONG_MODULE_TYPE = -54,
SREC_NOT_SUPPORTED_BY_MODULE = -55,
SREC_INTERNAL_ERROR = -56,
SREC_STORAGE_TRANSFER_ERROR = -57,
SREC_ANOTHER_VARIANT_OF_PLCSIM_RUNNING = -58,
SREC_ACCESS_DENIED = -59,
SREC_NOT_ALLOWED_DURING_DOWNLOAD = -60,
SREC_AUTODISCOVER_ALREADY_RUNNING = -61,
SREC_INVALID_STORAGE = -62,
SREC_WARNING_UNSUPPORTED_PCAP_DRIVER = 63, // (Deprecated since V4)
SREC_WARNING_RUNNING_ON_TIA_PORTAL_TEST_SUITE = 64,
SREC_PCAP_DRIVER_NOT_RUNNING = -65,
SREC_UNSUPPORTED_PCAP_DRIVER = -66,
SREC_CONFIG_FILE_ERROR = -67,
SREC_WARNING_PASSWORD_PROTECTION_ERROR = 68,
SREC_BUFFER_OVERFLOW = -69,
SREC_NOT_ALLOWED = -70,
SREC_WARNING_NO_EXTERNAL_COMMUNICATION = 71
SREC_INSTANCE_ALREADY_RUNNING = -72
SREC_OUT_OF_MEMORY = -73
};
```

Table 10-78 ERuntimeErrorCode - .NET (C#)

Syntax

```
enum ERuntimeErrorCode
{
OK = 0,
InvalidErrorCode = -1,
NotImplemented = -2,
IndexOutOfRange = -3,
DoesNotExist = -4,
AlreadyExists = -5,
UnknownMessageType = -6,
InvalidMessageId = -7,
WrongArgument = -8,
WrongPipe = -9,
ConnectionError = -10,
Timeout = -11,
```

```
MessageCorrupt = -12,
WrongVersion = -13,
InstanceNotRunning = -14,
InterfaceRemoved = -15,
SharedMemoryNotInitialized = -16,
ApiNotInitialized = -17,
WarningAlreadyExists = 18,
NotSupported = -19,
WarningInvalidCall = 20,
ErrorLoadingDll = -21,
SignalNameDoesNotExist = -22,
SignalTypeMismatch = -23,
SignalConfigurationError = -24,
NoSignalConfigurationLoaded = -25,
ConfiguredConnectionNotFound = -26,
ConfiguredDeviceNotFound = -27,
InvalidConfiguration = -28,
TypeMismatch = -29,
LicenseNotFound = -30,
NoLicenseAvailable = -31,
WrongCommunicationInterface = -32,
LimitReached = -33,
NoStartupPathSet = -34,
StartupPathAlreadyInUse = -35,
MessageIncomplete = -36,
ArchiveStorageNotCreated = -37,
RetrieveStorageFailure = -38,
InvalidOperatingState = -39,
InvalidArchivePath = -40,
DeleteExistingStorageFailed = -41,
CreateDirectoriesFailed = -42,
NotEnoughMemory = -43,
WarningTrialModeActive = 44,
NotRunning = -45,
NotEmpty = -46,
NotUpToDate = -47,
CommunicationInterfaceNotAvailable = -48,
WarningNotComplete = 49,
VirtualSwitchMisconfigured = -50,
RuntimeNotAvailable = -51,
IsEmpty = -52,
WrongModuleState = -53,
WrongModuleType = -54,
NotSupportedByModule = -55,
InternalError = -56,
StorageTransferError = -57,
AnotherVariantOfPlcsimRunning = -58,
AccessDenied = -59,
NotAllowedDuringDownload = -60,
AutodiscoverAlreadyRunning = -61,
InvalidStorage = -62,
WarningUnsupportedPcapDriver = 63, //(Deprecated since V4)
WarningRunningOnTiaPortalTestSuite = 64,
PCAPDriverNotRunning = -65,
UnsupportedPcapDriver = -66,
ConfigFileError = -67,
WarningPasswordForProtectionError = 68,
BufferOverflow = - 69,
NotAllowed = -70,
WarningNoExternalCommunication = 71
InstanceAlreadyRunning = -72
OutOfMemory = -73
}
```

10.7.2 EArea

Description

This enumeration contains all PLC areas that contain the available PLC tags.

Table 10-79 EArea - Native C++

Syntax	enum EArea
	{
	SRA_INVALID_AREA = 0,
	SRA_INPUT = 1,
	SRA_MARKER = 2,
	SRA_OUTPUT = 3,
	SRA_COUNTER = 4,
	SRA_TIMER = 5,
	SRA_DATABLOCK = 6,
	SRA_ENUMERATION_SIZE = 7
	};

Table 10-80 EArea - .NET (C#)

Syntax	public enum EArea
	{
	InvalidArea = 0,
	Input = 1,
	Marker = 2,
	Output = 3,
	Counter = 4,
	Timer = 5,
	DataBlock = 6,
	}

10.7.3 EOperatingState

Description

This enumeration contains the operating states of a simulated PLC instance.

Table 10-81 EOperatingState - Native C++

Syntax	enum EOperatingState
	{
	SROS_INVALID_OPERATING_STATE = 0,
	SROS_OFF = 1,
	SROS_BOOTING = 2,
	SROS_STOP = 3,
	SROS_STARTUP = 4,
	SROS_RUN = 5,
	SROS_FREEZE = 6,
	SROS_SHUTTING_DOWN = 7,
	SROS_HOLD = 8,
	
	SROS_ENUMERATION_SIZE = 9
	};

Table 10-82 EOperatingState - .NET (C#)

Syntax	`enum EOperatingState`
	`{`
	`InvalidOperatingState = 0,`
	`Off = 1,`
	`Booting = 2,`
	`Stop = 3,`
	`Startup = 4,`
	`Run = 5,`
	`Freeze = 6,`
	`ShuttingDown = 7,`
	`Hold = 8`
	`}`

10.7.4 EOperatingMode

Description

This enumeration contains the operating modes of a simulated PLC instance.

Table 10-83 EOperatingMode - Native C++

Syntax	`enum EOperatingMode`
	`{`
	`SROM_DEFAULT = 0,`
	`SROM_SINGLE_STEP_C = 1,`
	`SROM_SINGLE_STEP_CT = 2,`
	`SROM_TIMESPAN_SYNCHNRONIZED_C = 3,`
	`SROM_SINGLE_STEP_P = 4,`
	`SROM_TIMESPAN_SYNCHNRONIZED_P = 5,`
	`SROM_SINGLE_STEP_CP = 6,`
	`SROM_SINGLE_STEP_CPT = 7,`
	`SROM_TIMESPAN_SYNCHNRONIZED_CP = 8`
	`SROM_SINGLE_STEP_BUS = 9`
	`};`

Table 10-84 EOperatingMode - .NET (C#)

Syntax	`public enum EOperatingMode`
	`{`
	`Default = 0,`
	`SingleStep_C = 1,`
	`SingleStep_CT = 2,`
	`TimespanSynchronized_C = 3,`
	`SingleStep_P = 4,`
	`TimespanSynchronized_P = 5,`
	`SingleStep_CP = 6,`
	`SingleStep_CPT = 7,`
	`TimespanSynchronized_CP = 8`
	`SingleStep_Bus = 9`
	`}`

10.7.5 ECPUType

Description

This enumeration contains all CPU types that can be loaded in a simulated PLC instance.

Table 10-85 ECPUType - Native C++

Syntax	enum ECPUType { SRCT_1500_Unspecified = 0x000005DC, SRCT_1511 = 0x000005E7, SRCT_1513 = 0x000005E9, SRCT_1515 = 0x000005EB, SRCT_1516 = 0x000005EC, SRCT_1517 = 0x000005ED, SRCT_1518 = 0x000005EE, SRCT_1511C = 0x000405E7, SRCT_1512C = 0x000405E8, SRCT_1511F = 0x000105E7, SRCT_1513F = 0x000105E9, SRCT_1515F = 0x000105EB, SRCT_1516F = 0x000105EC, SRCT_1517F = 0x000105ED, SRCT_1518F = 0x000105EE, SRCT_1511T = 0x000805E7, SRCT_1515T = 0x000805EB, SRCT_1516T = 0x000805EC, SRCT_1517T = 0x000805ED, SRCT_1511TF = 0x000905E7, SRCT_1515TF = 0x000905EB, SRCT_1516TF = 0x000905EC, SRCT_1517TF = 0x000905ED, SRCT_1518ODK = 0x001005EE, SRCT_1518FODK = 0x001105EE, SRCT_1518MFP = 0x004005EE, SRCT_1518FMFP = 0x004105EE, SRCT_1518T = 0x000805EE, SRCT_1518TF = 0x000905EE, SRCT_1504DTF = 0x000905E0, SRCT_1507DTF = 0x000905E3, SRCT_ET200SP_Unspecified = 0x000205DC, SRCT_1510SP = 0x000205E6, SRCT_1512SP = 0x000205E8, SRCT_1510SPF = 0x000305E6, SRCT_1512SPF = 0x000305E8, SRCT_1514SP = 0x000205E7, SRCT_1514SPT = 0x000205E9, SRCT_1514SPF = 0x000305E7, SRCT_1514SPTF = 0x000305E9, SRCT_1514PA = 0x000305EB, SRCT_1500_RH_Unspecified = 0x008005DC, SRCT_1513R = 0x008005E9, SRCT_1515R = 0x008005EB, SRCT_1517H = 0x008005ED, SRCT_1518HF = 0x008105EE, SRCT_ET200PRO_Unspecified = 0x002005DC, SRCT_1513PRO = 0x002005E9, SRCT_1513PROF = 0x002105E9, SRCT_1516PRO = 0x002005EC, SRCT_1516PROF = 0x002105EC, SRCT_1500_SW_OC_Unspecified = 0x010005DC, SRCT_1507S = 0x010005E3, SRCT_1507SF = 0x010105E3, SRCT_1508S = 0x010005E4, SRCT_1508SF = 0x010105E4, SRCT_1508ST = 0x010805E4, SRCT_1508STF = 0x010905E4, SRCT_1505SP = 0x010005E1,

```
SRCT_1505SPF = 0x010105E1,
SRCT_1505SPT = 0x010805E1,
SRCT_1505SPTF = 0x010905E1
}
```

Table 10-86 ECPUType - .NET (C#)

```
Syntax
enum ECPUType
{
CPU1500_Unspecified = 0x000005DC,
CPU1511 = 0x000005E7,
CPU1513 = 0x000005E9,
CPU1515 = 0x000005EB,
CPU1516 = 0x000005EC,
CPU1517 = 0x000005ED,
CPU1518 = 0x000005EE,
CPU1511C = 0x000405E7,
CPU1512C = 0x000405E8,
CPU1511F = 0x000105E7,
CPU1513F = 0x000105E9,
CPU1515F = 0x000105EB,
CPU1516F = 0x000105EC,
CPU1517F = 0x000105ED,
CPU1518F = 0x000105EE,
CPU1511T = 0x000805E7,
CPU1515T = 0x000805EB,
CPU1516T = 0x000805EC,
CPU1517T = 0x000805ED,
CPU1511TF = 0x000905E7,
CPU1515TF = 0x000905EB,
CPU1516TF = 0x000905EC,
CPU1517TF = 0x000905ED,
CPU1518ODK = 0x001005EE,
CPU1518FODK = 0x001105EE,
CPU1518MFP = 0x004005EE,
CPU1518FMFP = 0x004105EE,
CPU1518T = 0x000805EE,
CPU1518TF = 0x000905EE,
CPU1504DTF = 0x000905E0,
CPU1507DTF = 0x000905E3,
CPUET200SP_Unspecified = 0x000205DC,
CPU1510SP = 0x000205E6,
CPU1512SP = 0x000205E8,
CPU1510SPF = 0x000305E6,
CPU1512SPF = 0x000305E8,
CPU1514SP = 0x000205E7,
CPU1514SPT = 0x000205E9,
CPU1514SPF = 0x000305E7,
CPU1514SPTF = 0x000305E9,
SRCT_1514PA = 0x000305EB,
CPU1500_RH_Unspecified = 0x008005DC,
CPU1513R = 0x008005E9,
CPU1515R = 0x008005EB,
CPU1517H = 0x008005ED,
CPU1518HF = 0x008105EE,
CPUET200PRO_Unspecified = 0x002005DC,
CPU1513PRO = 0x002005E9,
CPU1513PROF = 0x002105E9,
CPU1516PRO = 0x002005EC,
CPU1516PROF = 0x002105EC,
SRCT_1500_SW_OC_Unspecified = 0x010005DC,
SRCT_1507S = 0x010005E3,
SRCT_1507SF = 0x010105E3,
SRCT_1508S = 0x010005E4,
SRCT_1508SF = 0x010105E4,
SRCT_1508ST = 0x010805E4,
SRCT_1508STF = 0x010905E4,
SRCT_1505SP = 0x010005E1,
SRCT_1505SPF = 0x010105E1,
SRCT_1505SPT = 0x010805E1,
```

	SRCT_1505SPTF = 0x010905E1 }

10.7.6 ECommunicationInterface

Description

This enumeration contains the available communication interfaces of a simulated controller.

Table 10-87 ECommunicationInterface - Native C++

Syntax	enum ECommunicationInterface
	{
	SRCI_NONE = 0,
	SRCI_SOFTBUS = 1,
	SRCI_TCPIP = 2,
	SRCI_ENUMERATION_SIZE = 3
	};

Table 10-88 ECommunicationInterface - .NET (C#)

Syntax	enum ECommunicationInterface
	{
	None = 0,
	Softbus = 1,
	TCPIP = 2,
	}

10.7.7 ENetworkMode

Description

This enumeration contains the communication modes of a simulated PLC instance.

Table 10-89 ENetworkMode - Native C++

Syntax	enum class ENetworkMode
	{
	STATE_UNKNOWN = -1,
	TCPIP_MULTIPLE_ADAPTER = 0,
	TCPIP_SINGLE_ADAPTER = 1,
	SOFTBUS = 2
	};

Table 10-90 ENetworkMode - .NET (C#)

Syntax	public enum class ENetworkMode
	{
	UnknownState
	TCPIPMultipleAdapter
	TCPIPSingleAdapter
	Softbus
	};

10.7.8 EPLCInterface

Description

This enumeration includes the available interfaces of a simulated PLC instance.

Table 10-91 EPLCInterface - Native C++

Syntax	`enum class EPLCInterface`
	`{`
	`IE1 = 0,`
	`IE2,`
	`IE3`
	`};`

10.7.9 ELEDType

Description

This enumeration contains the types of LEDs for a simulated PLC instance.

Table 10-92 ELEDType - Native C++

Syntax	`enum ELEDType`
	`{`
	`SRLT_STOP = 0,`
	`SRLT_RUN = 1,`
	`SRLT_ERROR = 2,`
	`SRLT_MAINT = 3,`
	`SRLT_REDUND = 4,`
	`SRLT_FORCE = 5,`
	`SRLT_BUSF1 = 6,`
	`SRLT_BUSF2 = 7,`
	`SRLT_BUSF3 = 8,`
	`SRLT_BUSF4 = 9,`
	`SRLT_ENUMERATION_SIZE = 10`
	`};`

Table 10-93 ELEDType - .NET (C#)

Syntax	`enum ELEDType`
	`{`
	`Stop = 0,`
	`Run = 1,`
	`Error = 2`
	`Maint = 3,`
	`Redund = 4,`
	`Force = 5,`
	`Busf1 = 6,`
	`Busf2 = 7,`
	`Busf3 = 8,`
	`Busf4 = 9`
	`}`

10.7.10 ELEDMode

Description

This enumeration contains the LEDs of a simulated PLC instance.

Table 10-94 ELEDMode - Native C++

Syntax	`enum ELEDMode`
	`{`
	`SRLM_OFF = 0,`
	`SRLM_ON = 1,`
	`SRLM_FLASH_FAST = 2,`
	`SRLM_FLASH_SLOW = 3,`
	`SRLM_INVALID = 4`
	`};`

Table 10-95 ELEDMode - .NET (C#)

Syntax	`enum ELEDMode`
	`{`
	`Off = 0,`
	`On = 1,`
	`FlashFast = 2,`
	`FlashSlow = 3,`
	`Invalid = 4`
	`}`

10.7.11 EPrimitiveDataType

Description

This list contains all the primitive data types that are used by the I/O access functions.

Table 10-96 EPrimitiveDataType - Native C++

Syntax	`enum EPrimitiveDataType`
	`{`
	`SRPDT_UNSPECIFIC = 0,`
	`SRPDT_STRUCT = 1,`
	`SRPDT_BOOL = 2,`
	`SRPDT_INT8 = 3,`
	`SRPDT_INT16 = 4,`
	`SRPDT_INT32 = 5,`
	`SRPDT_INT64 = 6,`
	`SRPDT_UINT8 = 7,`
	`SRPDT_UINT16 = 8,`
	`SRPDT_UINT32 = 9,`
	`SRPDT_UINT64 = 10,`
	`SRPDT_FLOAT = 11,`
	`SRPDT_DOUBLE = 12,`
	`SRPDT_CHAR = 13,`
	`SRPDT_WCHAR = 14`
	`};`

Table 10-97 EPrimitiveDataType - .NET (C#)

Syntax	`enum EPrimitiveDataType`
	`{`
	`Unspecific = 0,`
	`Struct = 1,`
	`Bool = 2,`
	`Int8 = 3,`
	`Int16 = 4,`
	`Int32 = 5,`
	`Int64 = 6,`
	`UInt8 = 7,`
	`UInt16 = 8,`
	`UInt32 = 9,`
	`UInt64 = 10,`
	`Float = 11,`
	`Double = 12,`
	`Char = 13,`
	`WChar = 14`
	`}`

Compatible primitive data types

The following tables shows the primitive data types of the user interface (API) and the data types of the S7-PLCSIM Advanced instance that are configured in the stored tag list. The compatible data types are marked with "X".

Table 10-98 Compatible primitive data types - Reading

API	**S7-PLCSIM Advanced instance**											
	Bool	INT				UINT				Float	Double	Char	WChar
		8	16	32	64	8	16	32	64				
Bool	X												
INT8		X											
INT16		X	X			X							
INT32		X	X	X		X	X						
INT64		X	X	X	X	X	X	X					
UINT8						X							
UINT16						X	X						
UINT32						X	X	X					
UINT64						X	X	X	X				
Float										X			
Double											X		
Char												X	
WChar													X

Table 10-99 Compatible primitive data types - Write

API	S7-PLCSIM Advanced instance											
	Bool	INT				UINT				Float	Double	Char	WChar
		8	16	32	64	8	16	32	64				
Bool	X												
INT8		X	X	X	X								
INT16			X	X	X								
INT32				X	X								
INT64					X								
UINT8			X	X	X	X	X	X	X				
UINT16				X	X		X	X	X				
UINT32					X			X	X				
UINT64									X				
Float										X			
Double											X		
Char												X	
WChar													X

10.7.12 EDataType

Description

This enumeration contains all the PLC data types (STEP 7).

Table 10-100 EDataType - Native C++

Syntax	`enum EDataType`
	`{`
	`SRDT_UNKNOWN = 0,`
	`SRDT_BOOL = 1,`
	`SRDT_BYTE = 2,`
	`SRDT_CHAR = 3,`
	`SRDT_WORD = 4,`
	`SRDT_INT = 5,`
	`SRDT_DWORD = 6,`
	`SRDT_DINT = 7,`
	`SRDT_REAL = 8,`
	`SRDT_DATE = 9,`
	`SRDT_TIME_OF_DAY = 10,`
	`SRDT_TIME = 11,`
	`SRDT_S5TIME = 12,`
	`SRDT_DATE_AND_TIME = 14,`
	`SRDT_STRUCT = 17,`
	`SRDT_STRING = 19,`
	`SRDT_COUNTER = 28,`
	`SRDT_TIMER = 29,`
	`SRDT_IEC_Counter = 30,`
	`SRDT_IEC_Timer = 31,`
	`SRDT_LREAL = 48,`
	`SRDT_ULINT = 49,`
	`SRDT_LINT = 50,`
	`SRDT_LWORD = 51,`
	`SRDT_USINT = 52,`
	`SRDT_UINT = 53,`
	`SRDT_UDINT = 54,`

```
SRDT_SINT = 55,
SRDT_WCHAR = 61,
SRDT_WSTRING = 62,
SRDT_LTIME = 64,
SRDT_LTIME_OF_DAY = 65,
SRDT_LDT = 66,
SRDT_DTL = 67,
SRDT_IEC_LTimer = 68,
SRDT_IEC_SCounter = 69,
SRDT_IEC_DCounter = 70,
SRDT_IEC_LCounter = 71,
SRDT_IEC_UCounter = 72,
SRDT_IEC_USCounter = 73,
SRDT_IEC_UDCounter = 74,
SRDT_IEC_ULCounter = 75,
SRDT_ERROR_STRUCT = 97,
SRDT_NREF = 98,
SRDT_CREF = 101,
SRDT_AOM_IDENT = 128,
SRDT_EVENT_ANY = 129,
SRDT_EVENT_ATT = 130,
SRDT_EVENT_HWINT = 131,
SRDT_HW_ANY = 144,
SRDT_HW_IOSYSTEM = 145,
SRDT_HW_DPMASTER = 146,
SRDT_HW_DEVICE = 147,
SRDT_HW_DPSLAVE = 148,
SRDT_HW_IO = 149,
SRDT_HW_MODULE = 150,
SRDT_HW_SUBMODULE = 151,
SRDT_HW_HSC = 152,
SRDT_HW_PWM = 153,
SRDT_HW_PTO = 154,
SRDT_HW_INTERFACE = 155,
SRDT_HW_IEPORT = 156,
SRDT_OB_ANY = 160,
SRDT_OB_DELAY = 161,
SRDT_OB_TOD = 162,
SRDT_OB_CYCLIC = 163,
SRDT_OB_ATT = 164,
SRDT_CONN_ANY = 168,
SRDT_CONN_PRG = 169,
SRDT_CONN_OUC = 170,
SRDT_CONN_R_ID = 171,
SRDT_PORT = 173,
SRDT_RTM = 174,
SRDT_PIP = 175,
SRDT_OB_PCYCLE = 192,
SRDT_OB_HWINT = 193,
SRDT_OB_DIAG = 195,
SRDT_OB_TIMEERROR = 196,
SRDT_OB_STARTUP = 197,
SRDT_DB_ANY = 208,
SRDT_DB_WWW = 209,
SRDT_DB_DYN = 210,
SRDT_DB = 257
};
```

Table 10-101 EDataType - .NET (C#)

Syntax	

```
public enum EDataType
{
Unknown = 0,
Bool = 1,
Byte = 2,
Char = 3,
Word = 4,
Int = 5,
DWord = 6,
DInt = 7,
```

```
Real = 8,
Date = 9,
TimeOfDay = 10,
Time = 11,
S5Time = 12,
DateAndTime = 14,
Struct = 17,
String = 19,
Counter = 28,
Timer = 29,
IEC_Counter = 30,
IEC_Timer = 31,
LReal = 48,
ULInt = 49,
LInt = 50,
LWord = 51,
USInt = 52,
UInt = 53,
UDInt = 54,
SInt = 55,
WChar = 61,
WString = 62,
LTime = 64,
LTimeOfDay = 65,
LDT = 66,
DTL = 67,
IEC_LTimer = 68,
IEC_SCounter = 69,
IEC_DCounter = 70,
IEC_LCounter = 71,
IEC_UCounter = 72,
IEC_USCounter = 73,
IEC_UDCounte = 74,
IEC_ULCounter = 75,
ErrorStruct = 97,
NREF = 98,
CREF = 101,
Aom_Ident = 128,
Event_Any = 129,
Event_Att = 130,
Event_HwInt = 131,
Hw_Any = 144,
Hw_IoSystem = 145,
Hw_DpMaster = 146,
Hw_Device = 147,
Hw_DpSlave = 148,
Hw_Io = 149,
Hw_Module = 150,
Hw_SubModule = 151,
Hw_Hsc = 152,
Hw_Pwm = 153,
Hw_Pto = 154,
Hw_Interface = 155,
Hw_IEPort = 156,
OB_Any = 160,
OB_Delay = 161,
OB_Tod = 162,
OB_Cyclic = 163,
OB_Att = 164,
Conn_Any = 168,
Conn_Prg = 169,
Conn_Ouc = 170,
Conn_R_ID = 171,
Port = 173,
Rtm = 174,
Pip = 175,
OB_PCycle = 192,
OB_HwInt = 193,
OB_Diag = 195,
OB_TimeError = 196,
```

	OB_Startup = 197, DB_Any = 208, DB_WWW = 209, DB_Dyn = 210, DB = 257 }

10.7.13 ETagListDetails

Description

This list contains all PLC areas that can be used as a filter to update the tag table.

Table 10-102 ETagListDetails - Native C++

Syntax	enum ETagListDetails
	{
	SRTLD_NONE = 0,
	SRTLD_IO = 1,
	SRTLD_M = 2,
	SRTLD_IOM = 3,
	SRTLD_CT = 4,
	SRTLD_IOCT = 5,
	SRTLD_MCT = 6,
	SRTLD_IOMCT = 7,
	SRTLD_DB = 8,
	SRTLD_IODB = 9,
	SRTLD_MDB = 10,
	SRTLD_IOMDB = 11,
	SRTLD_CTDB = 12,
	SRTLD_IOCTDB = 13,
	SRTLD_MCTDB = 14,
	SRTLD_IOMCTDB = 15
	};

Table 10-103 ETagListDetails - .NET (C#)

Syntax	enum ETagListDetails
	{
	None = 0,
	IO = 1,
	M = 2,
	IOM = 3,
	CT = 4,
	IOCT = 5,
	MCT = 6,
	IOMCT = 7,
	DB = 8,
	IODB = 9,
	MDB = 10,
	IOMDB = 11,
	CTDB = 12,
	IOCTDB = 13,
	MCTDB = 14,
	IOMCTDB = 15
	}

10.7.14 ERuntimeConfigChanged

Description

This list contains all possible causes of an OnSoftwareConfigurationChanged event that the Runtime Manager sends.

Table 10-104 ERuntimeConfigChanged - Native C++

Syntax	`enum ERuntimeConfigChanged`
	`{`
	`SRCC_INSTANCE_REGISTERED = 0,`
	`SRCC_INSTANCE_UNREGISTERED = 1`
	`SRCC_CONNECTION_OPENED = 2,`
	`SRCC_CONNECTION_CLOSED = 3,`
	`SRCC_PORT_OPENED = 4,`
	`SRCC_PORT_CLOSED = 5`
	`};`

Table 10-105 ERuntimeConfigChanged - .NET (C#)

Syntax	`enum ERuntimeConfigChanged`
	`{`
	`InstanceRegistered = 0,`
	`InstanceUnregistered = 1,`
	`ConnectionOpened = 2,`
	`ConnectionClosed = 3,`
	`PortOpened = 4,`
	`PortClosed = 5`
	`}`

10.7.15 EInstanceConfigChanged

Description

This enumeration contains all possible causes for an OnSoftwareConfigurationChanged event sent from a simulated PLC instance.

Table 10-106 EInstanceConfigChanged - Native C++

Syntax	`enum EInstanceConfigChanged`
	`{`
	`SRICC_HARDWARE_SOFTWARE_CHANGED = 0,`
	`SRICC_IP_CHANGED = 1`
	`};`

Table 10-107 EInstanceConfigChanged - .NET (C#)

Syntax	`enum EInstanceConfigChanged`
	`{`
	`HardwareSoftwareChanged = 0,`
	`IPChanged = 1`
	`}`

10.7.16 EPullOrPlugEventType

Description

This enumeration contains predefined types of pull/plug events for S7 modules.

Table 10-108 EPullOrPlugEventType - Native C++

Syntax	`enum EPullOrPlugEventType`
	`{`
	`SR_PPE_UNDEFINED = 0,`
	`SR_PPE_PULL_EVENT = 1,`
	`SR_PPE_PLUG_EVENT = 2,`
	`SR_PPE_PLUG_EVENT_ERROR_REMAINS = 3,`
	`SR_PPE_PLUG_WRONG_MODULE_EVENT = 4`
	`};`

Table 10-109 EPullOrPlugEventType - .NET (C#)

Syntax	`enum EPullOrPlugEventType`
	`{`
	`Undefined = 0,`
	`Pull = 1,`
	`Plug = 2,`
	`PlugErrorRemains = 3,`
	`PlugWrongModule = 4`
	`}`

10.7.17 EProcessEventType

Description

This enumeration contains predefined types of process events for S7 modules.

Table 10-110 EProcessEventType - Native C++

Syntax	`enum EProcessEventType`
	`{`
	`SR_PET_UNDEFINED = 0,`
	`SR_PET_RISING_EDGE = 1,`
	`SR_PET_FALLING_EDGE = 2,`
	`SR_PET_LIMIT1_UNDERRUN = 3,`
	`SR_PET_LIMIT1_OVERRUN = 4,`
	`SR_PET_LIMIT2_UNDERRUN = 5,`
	`SR_PET_LIMIT2_OVERRUN = 6`
	`};`

Table 10-111 EProcessEventType - .NET (C#)

Syntax	`enum EProcessEventType`
	`{`
	`Undefined = 0,`
	`RisingEdge = 1,`
	`FallingEdge = 2,`
	`Limit_1_Underrun = 3,`
	`Limit_1_Overrun = 4,`
	`Limit_2_Underrun = 5,`
	`Limit_2_Overrun = 6`
	`}`

10.7.18 ESoftwareConfigChanged

Description

This enumeration contains download case indicators for an OnSoftwareConfigurationChanged event.

Table 10-112 ESoftwareConfigChanged - Native C++

Syntax	`enum ESoftwareConfigChanged`
	`{`
	`SRSCC_SOFTWARE_CHANGED_IN_STOP,`
	`SRSCC_SOFTWARE_CHANGED_IN_RUN`
	`};`

10.7.19 EDirection

Description

This enumeration contains properties of the diagnostic alarm.

Table 10-113 EDirection - Native C++

Syntax	`enum EDirection`
	`{`
	`SRD_DIRECTION_INPUT = 0,`
	`SRD_DIRECTION_OUTPUT = 1`
	`};`

Table 10-114 EDirection - .NET (C#)

Syntax	`enum EDirection`
	`{`
	`Input = 0,`
	`Output = 1`
	`}`

10.7.20 EDiagProperty

Description

This enumeration contains the incoming/outgoing information of the diagnostic alarm.

Table 10-115 EDiagProperty - Native C++

Syntax	`enum EDiagProperty`
	`{`
	`SRP_DIAG_APPEAR = 1,`
	`SRP_DIAG_DISAPPEAR = 2`
	`};`

Table 10-116 EDiagProperty - .NET (C#)

Syntax	`enum EDiagProperty`
	`{`
	`Appear = 1,`
	`Disappear = 2`
	`}`

10.7.21 EDiagSeverity

Description

This enumeration contains the severity of the diagnostic alarm (error, maintenance demanded, maintenance required).

Table 10-117 EDiagSeverity - Native C++

Syntax	`enum EDiagSeverity`
	`{`
	`SRDS_SEVERITY_FAILURE = 0,`
	`SRDS_SEVERITY_MAINTENANCE_DEMANDED = 1,`
	`SRDS_SEVERITY_MAINTENANCE_REQUIRED = 2`
	`};`

Table 10-118 EDiagSeverity - .NET (C#)

Syntax	`enum EDiagSeverity`
	`{`
	`Failure = 0,`
	`MaintDemanded = 1,`
	`MaintRequired = 2`
	`}`

10.7.22 ERackOrStationFaultType

Description

This enumeration contains the types of the RackOrStationFault event.

Table 10-119 ERackOrStationFaultType - Native C++

Syntax	`enum ERackOrStationFaultType`
	`{`
	`SR_RSF_FAULT = 0,`
	`SR_RSF_RETURN = 1`
	`};`

Table 10-120 ERackOrStationFaultType - .NET (C#)

Syntax	`enum ERackOrStationFaultType`
	`{`
	`Fault = 0,`
	`Return = 1`
	`}`

10.7.23 ECycleTimeMonitoringMode

Description

This enumeration contains the sources of the timer for the maximum cycle time monitoring.

Table 10-121 ECycleTimeMonitoringMode - Native C++

Syntax	`enum ECycleTimeMonitoringMode`
	`{`
	`SRCTMM_DOWNLOADED = 0,`
	`SRCTMM_IGNORED = 1,`
	`SRCTMM_SPECIFIED = 2`
	`};`

Table 10-122 ECycleTimeMonitoringMode - .NET (C#)

Syntax	`enum ECycleTimeMonitoringMode`
	`{`
	`Downloaded = 0,`
	`Ignored = 1,`
	`Specified = 2`
	`}`

10.7.24 EAutodiscoverType

Description

This enumeration is used in the Autodiscover Callback function.

Table 10-123 EAutodiscoverType - Native C++

Syntax	`enum EAutodiscoverType`
	`{`
	`SRRSI_DISCOVER_STARTED = 0,`
	`SRRSI_DISCOVER_DATA = 1,`
	`SRRSI_DISCOVER_FINISHED = 2`
	`};`

Table 10-124 EAutodiscoverType - .NET (C#)

Syntax	`public enum EAutodiscoverType`
	`{`
	`AutodiscoverStarted = 0,`
	`AutodiscoverData = 1,`
	`AutodiscoverFinished = 2`
	`}`

Examples/applications

11

11.1 Example: Communication via TCP/IP in Single Adapter Network Mode (promiscuous mode)

In Single Adapter Network Mode (promiscuous mode), the network adapter reads all incoming telegrams. The network adapter also reads telegrams that are not intended for the network adapter. The network adapter then forwards the data to the operating system for processing.

However, on virtualization platforms or corporate networks, it is often not permitted to run the network interface of the PC in Single Adapter Network Mode (promiscuous mode) for security reasons. For an example on Multi-Adapter Mode (non-prmiscuous mode), see the next topic.

The following figure provides an overview of the possibilities and differences between the following communication options:

- Local TCP/IP communication
- Single Adapter Network Mode (promiscuous mode)
- Multi-Adapter Network Mode (non-promiscuous mode)

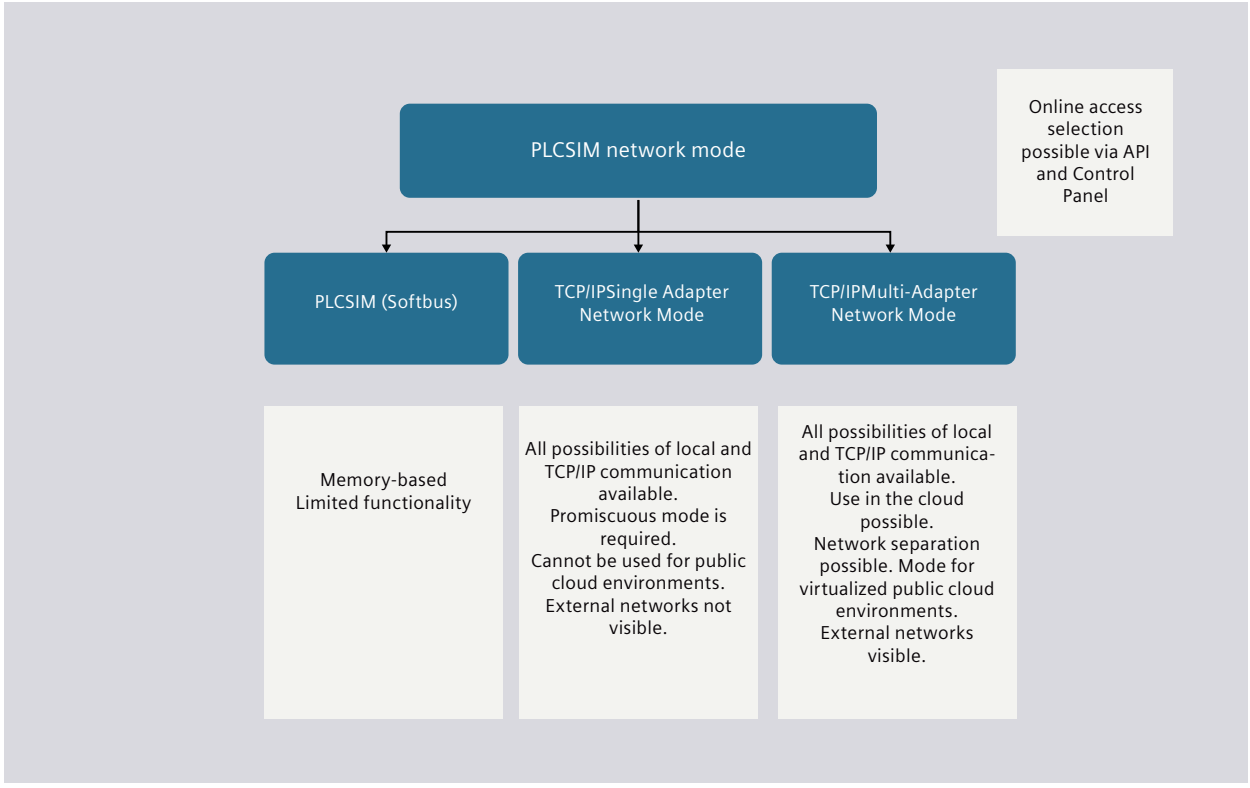


Figure 11-1 PLCSIM_networking

Example code

In this example, the physical interface is set to promiscuous mode and receives all packets; PCAP is filtering traffic. Each PLC interface has its own unique MAC address. Only two network adapters can be connected to the PLCSIM virtual switch.

```
#include "SimulationRuntimeApi.h"


ISimulationRuntimeManager* rtm = nullptr;
IInstance* instance0 = nullptr;
IInstance* instance1 = nullptr;
ERuntimeErrorCode result = SREC_INVALID_ERROR_CODE;


//initialize connection
InitializeApi(&rtm);


//set TCPIP single adapter mode (softbus mode is default)
rtm->SetNetworkMode(ENetworkMode::TCPIP_SINGLE_ADAPTER);


rtm->RegisterInstance(&instance0);
rtm->RegisterInstance(&instance1);
instance0->PowerOn(60000);
instance1->PowerOn(60000);


ShutdownAndFreeApi(rtm);
```

11.2 Example: Communication via TCP/IP in Multi-Adapter Network Mode (non-promiscuous mode)

In Multi-Adapter Network Mode (non-promiscuous mode), the evaluation of MAC addresses ensures that only data intended for the network adapter reaches the operating system.

Overview

The following figure provides an overview of the possibilities and differences between the following communication options:

- Local TCP/IP communication
- Single Adapter Network Mode (promiscuous mode)
- Multi-Adapter Network Mode (non-promiscuous mode)

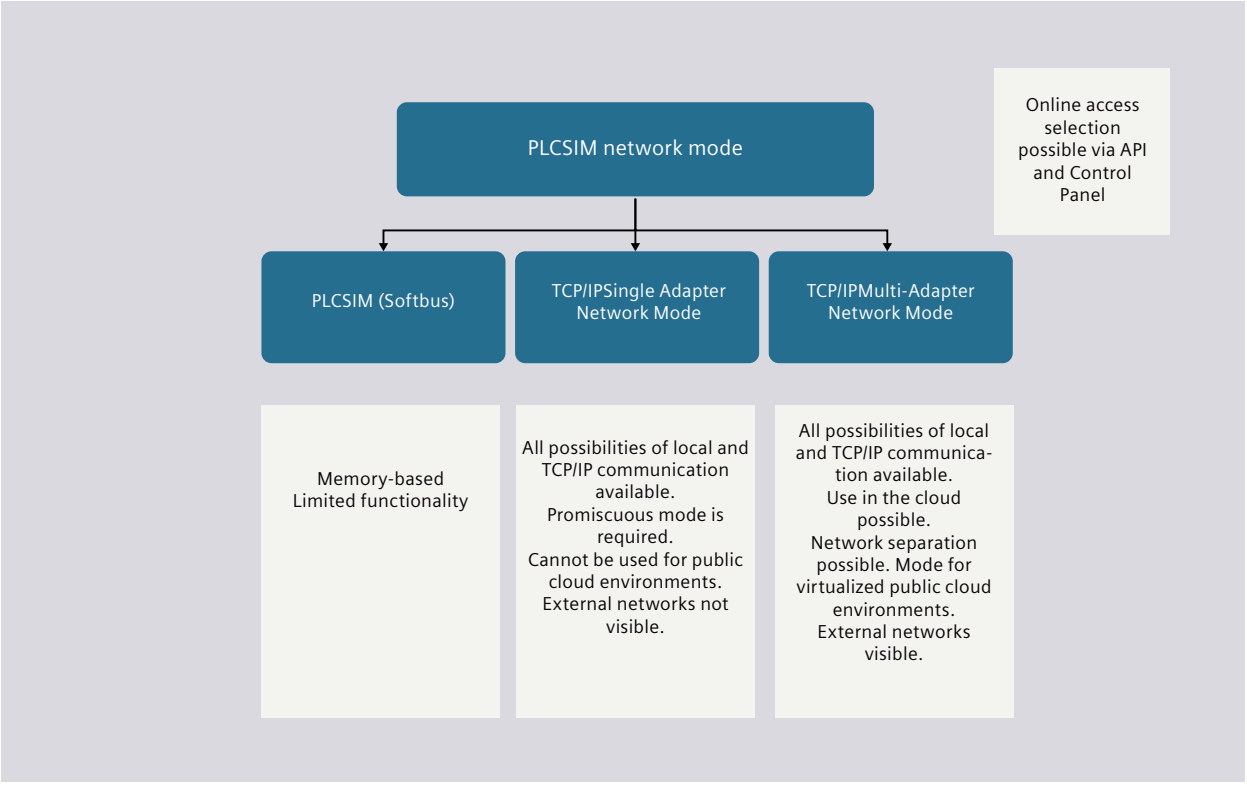


Figure 11-2 PLCSIM networking

Applications

The following applications describe how you can also communicate in Multi-Adapter Network Mode (non-promiscuous mode). You communicate in this mode by performing a one-to-one mapping of the MAC addresses between the CPU network interfaces and the network adapters of the PC.

The following applications assume that you are working in an environment where security restrictions apply and you have therefore switched to Multi-Adapter Network Mode (non-promiscuous mode).

Application 1:

You start multiple instances. Each CPU interface is assigned to the MAC address of the respective physical or virtual PC adapter.

Precondition: Each interface of a running instance is assigned to its own physical or virtual PC interface (one-to-one mapping).

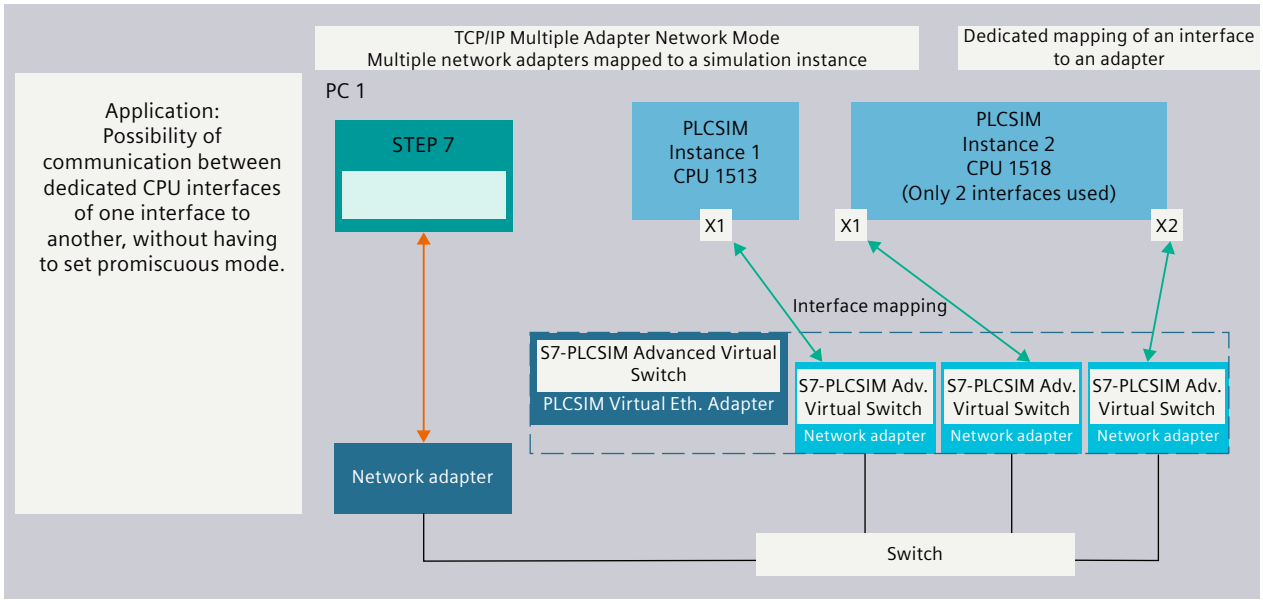


Figure 11-3 Application 1

Application 2:

The following application shows the behavior with multiple instances that have the same MAC addresses. Each CPU interface is assigned to the corresponding physical or virtual PC adapter.

The scenario is only valid if only one instance is running at a time.

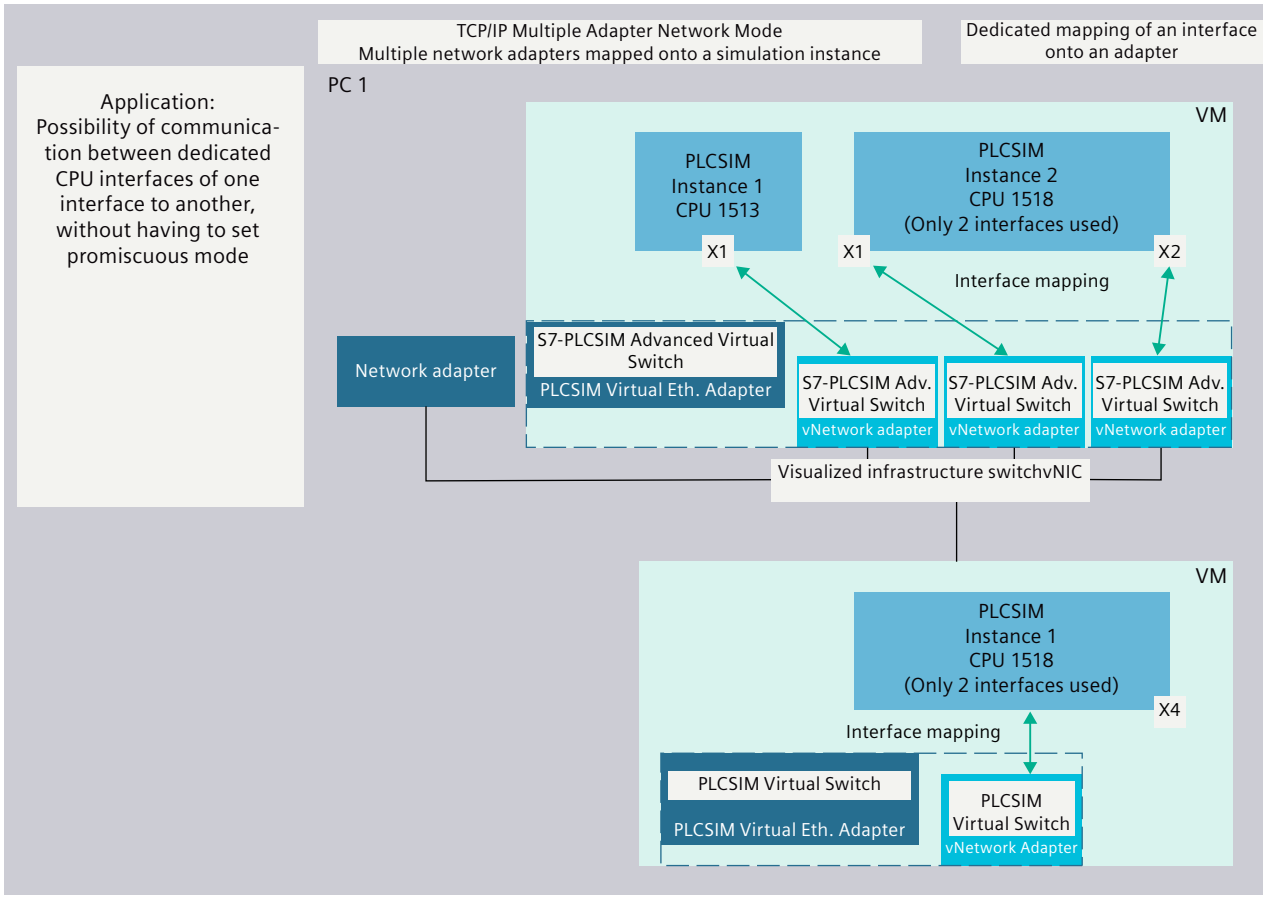


Figure 11-4 Application 2

Procedure

For external communication in Multi-Adapter Network Mode (non-promiscuous mode), the following steps are necessary depending on your application.

1. Use the following instruction to switch to Multi-Adapter Network Mode (non-promiscuous mode):
   `SetNetworkMode(SRCM_VSWITCH_NON_PROMISCUOUS)`
2. To register a new instance, call `RegisterInstance()`.
3. To select which PC interface is assignable to which CPU interface by means of a list of PC interfaces, call `GetNetInterfaces()`.
4. To execute an interface mapping between CPU and PC interface, call `SetNetInterfaceMapping(0, 23)`.
5. To check whether the configuration is correct, call `CheckNetInterfaceMappig()`.
6. To start the instance, call `PowerOn()`.

   Before starting, the instance performs an internal check.

NOTE

Setting a storage path for a new instance

If you use the `SetStoragePath()` function to set a storage path for a new instance, call this function before the MAC address is set. Otherwise, the MAC address is discarded.

This procedure is also valid for the stored CPU type and CPU serial number.

The `CleanupStoragePath()` function deletes not only the storage path but also the assignments of the network interfaces.

Examples in C++

The following examples show possible network configurations in C++.

For scenarios with more than one adapter, make sure to register the instance with a CPU that has the required number of interfaces. In the following examples we use three interfaces. Therefore we register the instance with a CPU with 3 PROFINET interfaces, such as the CPU 1518-4 PN/DP.

Example 1: Simplified settings

```
#include "SimulationRuntimeApi.h"

ISimulationRuntimeManager* rtm = nullptr;
IInstance* instance0 = nullptr;
IInstance* instance1 = nullptr;
ERuntimeErrorCode result = SREC_INVALID_ERROR_CODE;

//initialize connection
InitializeApi(&rtm);

//set NON-P mode
rtm->SetNetworkMode(ENetworkMode::TCPIP_MULTIPLE_ADAPTER);

rtm->RegisterInstance(ECPUType::SRCT_1518, &instance0);
instance0->SetNetInterfaceMapping(EPLCInterface::IE1, L"Ethernet");
instance0->SetNetInterfaceMapping(EPLCInterface::IE2, L"Ethernet
2");
instance0->SetNetInterfaceMapping(EPLCInterface::IE3, L"Ethernet
3");
instance0->PowerOn(60000);

rtm->RegisterInstance(ECPUType::SRCT_1518, &instance1);
instance1->SetNetInterfaceMapping(EPLCInterface::IE1, L"Ethernet
4");
instance1->SetNetInterfaceMapping(EPLCInterface::IE2, L"Ethernet
5");
```

```
instance1->SetNetInterfaceMapping(EPLCInterface::IE3, L"Ethernet
6");
instance1->PowerOn(60000);


ShutdownAndFreeApi(rtm);
```

Example 2: Complete settings

```
#include <iostream>
#include "SimulationRuntimeApi.h"

ISimulationRuntimeManager* rtm = nullptr;
IInstance* instance0 = nullptr;
IInstance* instance1 = nullptr;
ERuntimeErrorCode result = SREC_INVALID_ERROR_CODE;

// initialize connection
InitializeApi(&rtm);

// User can get the current network mode using GetNetworkMode. The
default mode is softbus.
ENetworkMode netMode;
rtm->GetNetworkMode(&netMode);

// If TCPIP_MULTIPLE_ADAPTER is set, the SetNetworkMode will not set
it again.
rtm->SetNetworkMode(ENetworkMode::TCPIP_MULTIPLE_ADAPTER);

rtm->RegisterInstance(ECPUType::SRCT_1518, &instance0);
rtm->RegisterInstance(ECPUType::SRCT_1518, &instance1);

// gather network interfaces in computer
// get count of network interfaces
UINT count;
result = rtm->GetNetInterfaces(nullptr, &count);

// get network interfaces
// GetNetInterfaces will return adapters with disabled
vSwitch and/or not connected as well
std::unique_ptr<SNetInterfaceInfo[]> netInterfaces =
std::make_unique<SNetInterfaceInfo[]>(count);
result = rtm->GetNetInterfaces(netInterfaces.get(), &count);

// map desired interface on instance interface
```

```
// each instance interface must have unique or no mapping
// we can map interface by these three overloads
//
// interfaceIndex
result = instance0->SetNetInterfaceMapping(EPLCInterface::IE1,
netInterfaces.get()[0].interfaceIndex);
// friendly name
result = instance0->SetNetInterfaceMapping(EPLCInterface::IE2,
netInterfaces.get()[1].interfaceName);
// MAC address
result = instance0->SetNetInterfaceMapping(EPLCInterface::IE3,
&netInterfaces.get()[2].MACAddress);


// set mapping on second instance
result = instance1->SetNetInterfaceMapping(EPLCInterface::IE1,
&netInterfaces.get()[3].MACAddress);
result = instance1->SetNetInterfaceMapping(EPLCInterface::IE2,
&netInterfaces.get()[4].MACAddress);
result = instance1->SetNetInterfaceMapping(EPLCInterface::IE3,
&netInterfaces.get()[5].MACAddress);


// we can also check what is mapped on the interface
INT32 mappedInterfaceIndex = instance0-
>GetNetInterfaceMapping(EPLCInterface::IE1);
if (mappedInterfaceIndex == 0)
{
std::cout << "No mapping is set";
}


if (mappedInterfaceIndex == netInterfaces.get()[0].interfaceIndex)
{
std::cout << netInterfaces.get()[0].interfaceName << "is mapped on
EPLCInterface::IE1";
}


// set S7-PLCSIM Advanced Virtual Switch according to set mapping
// !! requires elevated (admin) mode !!
rtm->SetNetInterfaceBindings();


// check whether setting is valid - optional -
CheckNetInterfaceMapping() function is also called inside PowerOn()
result = instance0->CheckNetInterfaceMapping();
if (result != SREC_OK)
{
return result;
}
```

```
instance0->PowerOn(60000);
instance1->PowerOn(60000);

ShutdownAndFreeApi(rtm);
```