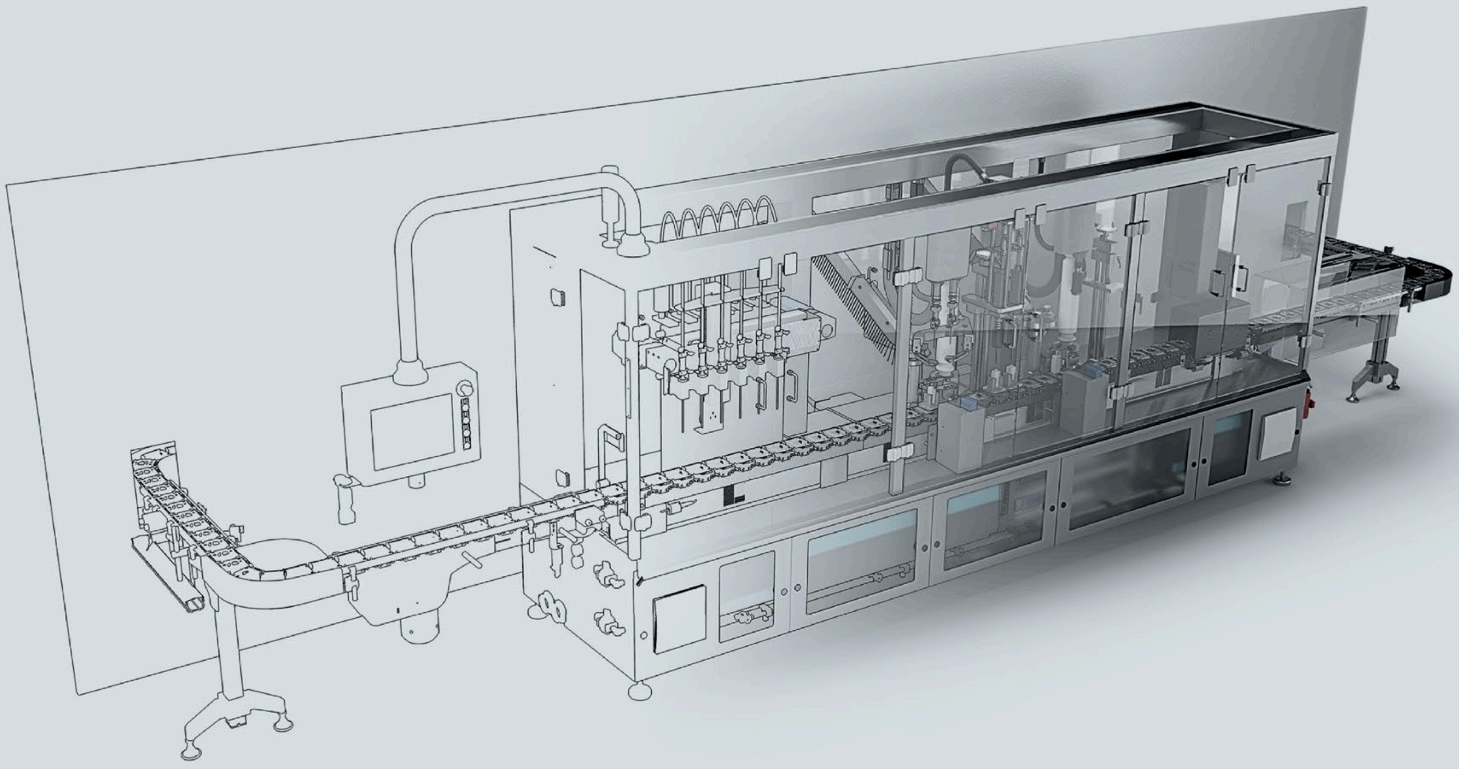
SIEMENS



Function manual

SIMATIC

S7-1500

S7-PLCSIM Advanced

Edition 05/2021

support.industry.siemens.com

SIEMENS

SIMATIC

S7-1500
S7-PLCSIM Advanced

Function Manual

Preface

Guide 1

Product overview 2

Installing 3

Communication paths 4

Simulation 5

Virtual time response 6

User interfaces (API) 7

Restrictions, messages and solution 8

List of abbreviations A

05/2021

A5E37039512-AD

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 AG. 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 AG
Digital Industries
Postfach 48 48
90026 NÜRNBERG
GERMANY

A5E37039512-AD
Ⓟ 03/2021 Subject to change

Copyright © Siemens AG 2016 - 2021.
All rights reserved

Preface

Purpose of the documentation

This function manual describes the simulation software, SIMATIC S7-PLCSIM Advanced V4.0.
You can use this software to simulate and test your SIMATIC STEP 7 programs on a virtual controller.

Scope

This function manual is valid for the following order versions

- 6ES7823-1FA03-0YA5 SIMATIC S7-PLCSIM Advanced V4.0
- 6ES7823-1FE03-0YA5 SIMATIC S7-PLCSIM Advanced V4.0 (Download)
- 6ES7823-1FA03-0YE5 SIMATIC S7-PLCSIM Advanced V4.0 Upgrade V1.0 to V4.0
- 6ES7823-1FE03-0YE5 SIMATIC S7-PLCSIM Advanced V4.0 Upgrade V1.0 to V4.0 (Download)

The articles each contain one license for two instances.

Basic knowledge required

The software 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

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

We also abbreviate SIMATIC S7-PLCSIM Advanced V4.0 as "PLCSIM Advanced".

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.

Special information

Note

Readme

You can obtain updates to the function manual as downloads on the Internet (https://support.industry.siemens.com/cs/us/en/view/109773483).

Application examples

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

- SIMATIC S7-PLCSIM Advanced: Co-Simulation via API (1 (https://support.industry.siemens.com/cs/ww/de/view/109739660/en))
- Digitalization with TIA Portal: Virtual commissioning with SIMATIC and Simulink (2 (https://support.industry.siemens.com/cs/ww/en/document/109749187))

Security information

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

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

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

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

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

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

Table of contents

Preface 3

1 Guide 12

1.1 S7-PLCSIM products 14

2 Product overview 16

2.1 What is S7-PLCSIM Advanced? 16

2.2 Compatibility during upgrade 17

2.3 Security for S7-PLCSIM Advanced 18

2.4 Simulations support 19

2.5 Supported CPUs 20

2.6 Differences between a simulated and a real CPU 21

2.6.1 Restrictions for all supported CPUs 22

2.6.2 Notes 23

2.7 Password to protect confidential configuration data 25

3 Installing 27

3.1 Introduction 27

3.1.1 System requirements 27

3.1.2 Restrictions due to antivirus programs 29

3.1.3 Licenses 30

3.1.4 Trial License 30

3.1.5 Installation log 32

3.2 S7-PLCSIM Advanced 33

3.3 Installing S7-PLCSIM Advanced 34

3.4 Changing S7-PLCSIM Advanced 36

3.5 Repairing S7-PLCSIM Advanced 37

3.6 Uninstalling S7-PLCSIM Advanced 37

4 Communication paths 39

4.1 Local communication 41

4.2 Communication via TCP / IP 42

4.3 Enable distributed communication 45

5 Simulation ... 47

5.1 Simulate CPU ... 47
5.1.1 Basic procedure for the simulation ... 47
5.1.2 Control Panel - User interface ... 48
5.1.2.1 S7-PLCSIM Advanced Symbol ... 48
5.1.2.2 Graphical interface ... 49
5.1.2.3 S7-PLCSIM Advanced Control Panel ... 50
5.1.2.4 Importing instances ... 53
5.1.3 Download ... 55
5.1.4 Network addresses in the simulation ... 57
5.1.4.1 Siemens PLCSIM Virtual Ethernet Adapter ... 57
5.1.4.2 PLCSIM Advanced instances ... 58
5.1.5 Simulate peripheral I/O ... 59
5.1.6 Simulate communication ... 60
5.1.6.1 Communication services that can be simulated ... 60
5.1.6.2 Communication between instances ... 61
5.1.7 Provide project data offline for simulation ... 62

5.2 Simulate CPU with ODK functionality ... 64
5.2.1 Special features of ODK ... 64
5.2.2 Loading functions ... 67
5.2.3 Calling functions ... 68
5.2.4 Unloading functions ... 68

5.3 Simulating Motion Control ... 69

5.4 Simulating the SIMATIC Drive Controller ... 71

5.5 Simulating a redundant S7-1500R/H system ... 73

6 Virtual time response ... 75

6.1 Speed up and slow down simulation ... 77

6.2 Stop simulation ... 78

6.3 Synchronize simulation partner ... 80
6.3.1 Synchronize simulation partner cycle-controlled ... 80
6.3.2 Synchronize simulation partner time-controlled ... 82

7 User interfaces (API) ... 84

7.1 Introduction ... 84
7.1.1 Access to instances ... 86
7.1.2 User interfaces (API) ... 87
7.1.3 Overview of user interfaces for native C++ ... 88
7.1.4 Overview of user interfaces for managed code ... 93
7.1.5 Overview of data types for native C++ ... 96
7.1.6 Overview of data types for managed code ... 98

7.2 Initialize API ... 99
7.2.1 Load API library ... 99
7.2.2 Native C++ ... 100
7.2.2.1 InitializeApi() ... 100
7.2.2.2 RuntimeApiEntry_Initialize ... 102
7.2.3 .NET (C#) ... 104
7.2.3.1 Initialize ... 104

7.3 Shut down API 104
7.3.1 Native C++ 104
7.3.1.1 DestroyInterface() 105
7.3.1.2 RuntimeApiEntry_DestroyInterface 106
7.3.1.3 FreeApi() 107
7.3.1.4 ShutdownAndFreeApi() 108
7.3.2 .NET (C#) 109
7.3.2.1 Shut down API 109

7.4 Global functions (Native C++) 109

7.5 API ISimulationRuntimeManager 114
7.5.1 Interfaces - Information and settings 114
7.5.2 Simulation Runtime instances 118
7.5.3 Remote connections 124
7.5.3.1 RunAutodiscover() 130
7.5.4 Events for ISimulationRuntimeManager 131
7.5.4.1 OnConfigurationChanged events 132
7.5.4.2 OnRuntimeManagerLost events 134
7.5.4.3 OnAutodiscoverData events 137

7.6 API IInstances 138
7.6.1 Interfaces - Information and settings 138
7.6.2 Controller - Information and settings 144
7.6.3 Operating state 156
7.6.4 Tag list 167
7.6.5 I/O access 174
7.6.5.1 Synchronizing inputs and outputs 174
7.6.5.2 I/O access via address - Reading 174
7.6.5.3 I/O access via address - Writing 183
7.6.5.4 I/O access via tag name - Reading 189
7.6.5.5 I/O access via tag name - Writing 220
7.6.6 Settings for the virtual time 249
7.6.7 Cycle control 252
7.6.8 Acyclic services 261
7.6.8.1 Overview 261
7.6.8.2 ReadRecordDone / WriteRecordDone 263
7.6.8.3 AlarmNotification 266
7.6.8.4 ProcessEvent 269
7.6.8.5 PullOrPlugEvent 271
7.6.8.6 StatusEvent 273
7.6.8.7 ProfileEvent 275
7.6.8.8 UpdateEvent 277
7.6.8.9 GetConfiguredProcessEvent 278
7.6.8.10 RackOrStationFaultEvent 279
7.6.9 Events for IInstances 281
7.6.9.1 Events for operating state and cycle control 281
7.6.9.2 Events for acyclic services 295

7.7 API IRemoteRuntimeManager 304
7.7.1 Interfaces - Information and settings 304
7.7.2 Simulation Runtime instances 309
7.7.2.1 Simulation Runtime instances (remote) 309
7.7.3 Events for IRemoteRuntimeManager 317
7.7.3.1 OnConnectionLost events 317

7.8 Data types 320
7.8.1 DLL import functions (Native C++) 321
7.8.1.1 ApiEntry_Initialize 321
7.8.1.2 ApiEntry_DestroyInterface 321
7.8.2 Event callback functions (Native C++) 322
7.8.2.1 EventCallback_VOID 322
7.8.2.2 EventCallback_SRCC_UINT32_UINT32_INT32 322
7.8.2.3 EventCallback_SRRSI_AD 323
7.8.2.4 EventCallback_IRRTM 323
7.8.2.5 EventCallback_II_SREC_ST_SROS_SROS 324
7.8.2.6 EventCallback_II_SREC_ST_UINT32_INT64_INT64_UINT32 325
7.8.2.7 EventCallback_II_SREC_ST 326
7.8.2.8 EventCallback_II_SREC_ST_SRICC_UINT32_UINT32_UINT32_UINT32 327
7.8.2.9 EventCallback_II_SREC_ST_SRLT_SRLM 328
7.8.2.10 EventCallback_II_SREC_ST_SDRI 329
7.8.2.11 EventCallback_II_SREC_ST_SDRI_BYTE 330
7.8.2.12 EventCallback_II_SREC_ST_UINT32_UINT32 331
7.8.2.13 EventCallback_II_SREC_ST_UINT32_UINT32_EPET_UINT32 332
7.8.2.14 EventCallback_II_SREC_ST_UINT32_EPPET_UINT32 333
7.8.2.15 EventCallback_II_SREC_ST_UINT32_ERSFET 334
7.8.2.16 EventCallback_II_SREC_ST_UINT32 335
7.8.3 Delegate definitions (managed code) 335
7.8.3.1 Delegate_Void 335
7.8.3.2 Delegate_SRCC_UINT32_UINT32_INT32 336
7.8.3.3 Delegate_SRRSI_AD 337
7.8.3.4 Delegate_II_EREC_DT 337
7.8.3.5 Delegate_II_EREC_DT_EOS_EOS 338
7.8.3.6 Delegate_II_EREC_DT_ELT_ELM 339
7.8.3.7 Delegate_II_EREC_DT_UINT32_INT64_INT64_UINT32 340
7.8.3.8 Delegate_IRRTM 341
7.8.3.9 Delegate_II_EREC_DT_SRICC_UINT32_UINT32_UINT32_UINT32 341
7.8.3.10 Delegate_II_EREC_DT_SDRI 342
7.8.3.11 Delegate_II_EREC_DT_SDR 343
7.8.3.12 Delegate_SREC_ST_UINT32_EPPET_UINT32 344
7.8.3.13 Delegate_SREC_ST_UINT32_UINT32_EPET_UINT32 345
7.8.3.14 Delegate_SREC_ST_UINT32 346
7.8.3.15 Delegate_SREC_ST_UINT32_UINT32 347
7.8.3.16 Delegate_SREC_ST_UINT32_ERSFET 348
7.8.4 Definitions and constants 349
7.8.5 Unions (Native C++) 350
7.8.5.1 UIP 350
7.8.5.2 UDataValue 351
7.8.6 Structures 353
7.8.6.1 SDataValue 353
7.8.6.2 SDVBNI 356
7.8.6.3 SDataValueByAddress 356

7.8.6.4 SDataValueByAddressWithCheck 357
7.8.6.5 SDataValueByName 358
7.8.6.6 SDataValueByNameWithCheck 359
7.8.6.7 SConnectionInfo 359
7.8.6.8 SInstanceInfo 360
7.8.6.9 SDimension 360
7.8.6.10 STagInfo 361
7.8.6.11 SIP 364
7.8.6.12 SIPSuite4 364
7.8.6.13 SOnSyncPointReachedResult 366
7.8.6.14 SDataRecordInfo 368
7.8.6.15 SDataRecord 369
7.8.6.16 SConfiguredProcessEvents 369
7.8.6.17 SDiagExtChannelDescription 371
7.8.6.18 SAutodiscoverData 373
7.8.7 Enumerations 374
7.8.7.1 ERuntimeErrorCode 375
7.8.7.2 EArea 379
7.8.7.3 EOperatingState 380
7.8.7.4 EOperatingMode 381
7.8.7.5 ECPUType 382
7.8.7.6 ECommunicationInterface 384
7.8.7.7 ELEDType 384
7.8.7.8 ELEDMode 385
7.8.7.9 EPrimitiveDataType 385
7.8.7.10 EDataType 388
7.8.7.11 ETagListDetails 392
7.8.7.12 ERuntimeConfigChanged 393
7.8.7.13 EInstanceConfigChanged 393
7.8.7.14 EPullOrPlugEventType 394
7.8.7.15 EProcessEventType 394
7.8.7.16 EDirection 395
7.8.7.17 EDiagProperty 396
7.8.7.18 EDiagSeverity 396
7.8.7.19 ERackOrStationFaultType 397
7.8.7.20 ECycleTimeMonitoringMode 397
7.8.7.21 EAutodiscoverType 398

8 Restrictions, messages and solution 399

8.1 Overview 399

8.2 OPC UA server 400

8.3 Web server 400

8.4 Backing up and restoring the configuration of a PLCSIM Advanced instance 401

8.5 Restrictions for file paths 402

8.6 Restrictions for communications services 402

8.7 Restrictions for instructions 402

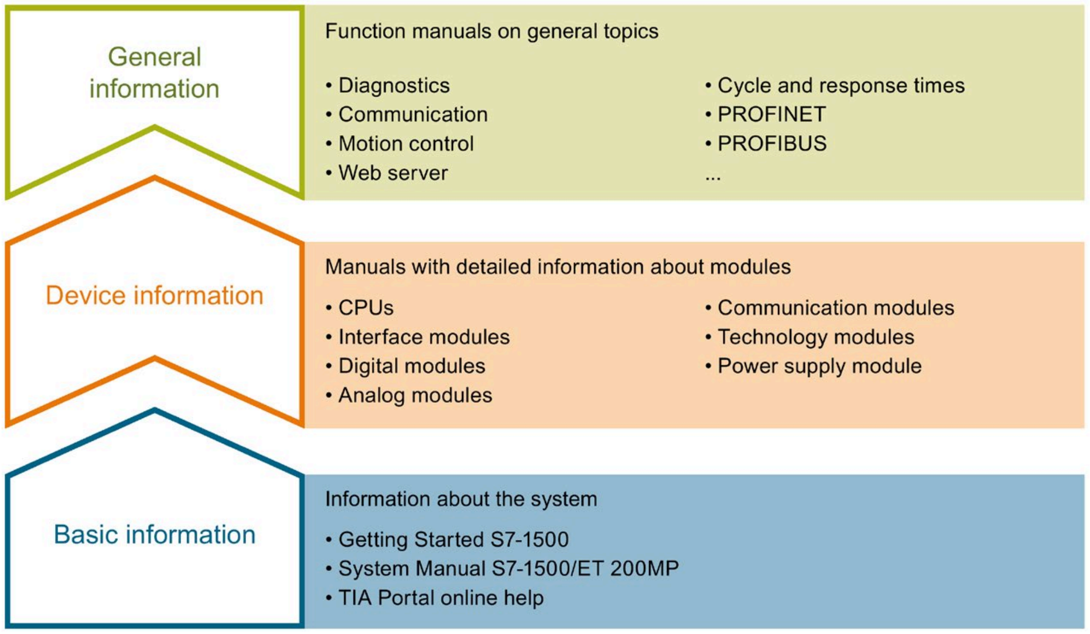
8.8 Restrictions to local communication via Softbus 403

8.9 Messages for communication via TCP/IP 404

8.10 Restrictions of security with VMware vSphere Hypervisor (ESXi) 405
8.11 Restrictions for Hyper-V 407
8.12 Monitoring overflow 407
8.13 Deviating I/O values in the STEP 7 user program 408
8.14 Multiple simulations and possible collision of IP addresses 408
8.15 Lacking access to an IP address 408
8.16 Simulation in standby mode 408
8.17 ET 200SP CPUs: Use of BusAdapters with fiber-optic interface 409
8.18 Installing SIMATIC NET 409
A List of abbreviations 410

Guide 1

The documentation for the SIMATIC S7-1500 automation system and the SIMATIC ET 200MP distributed I/O system is arranged into three areas.
This arrangement enables you to access the specific content you require.



Basic information

The System Manual and Getting Started describe in detail the configuration, installation, wiring and commissioning of the SIMATIC S7-1500 and ET 200MP systems. The STEP 7 online help supports you in the configuration and programming.

Device information

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

General information

The function manuals contain detailed descriptions on general topics regarding the SIMATIC S7-1500 and ET 200MP systems, e.g. diagnostics, communication, motion control, Web server, OPC UA.

You can download the documentation free of charge from the Internet (https://support.industry.siemens.com/cs/ww/en/view/109742691).

Changes and supplements to the manuals are documented in a Product Information.

You can download the product information free of charge from the Internet (https://support.industry.siemens.com/cs/us/en/view/68052815).

Manual Collection S7-1500/ET 200MP

The Manual Collection contains the complete documentation on the SIMATIC S7-1500 automation system and the ET 200MP distributed I/O system gathered together in one file.

You can find the Manual Collection on the Internet (https://support.industry.siemens.com/cs/ww/en/view/86140384).

SIMATIC S7-1500 comparison list for programming languages

The comparison list contains an overview of which instructions and functions you can use for which controller families.

You can find the comparison list on the Internet (https://support.industry.siemens.com/cs/ww/en/view/86630375).

"mySupport"

With "mySupport", your personal workspace, you make the best out of your Industry Online Support.

In "mySupport", you can save filters, favorites and tags, request CAx data and compile your personal library in the Documentation area. In addition, your data is already filled out in support requests and you can get an overview of your current requests at any time.

You must register once to use the full functionality of "mySupport".

You can find "mySupport" on the Internet (https://support.industry.siemens.com/My/ww/en).

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 will find the application examples on the Internet (https://support.industry.siemens.com/cs/ww/en/ps/ae).

1.1 S7-PLCSIM products

PLCSIM Advanced V4.0, PLCSIM V16 and PLCSIM V5.x

Table 1- 1 Comparison of S7-PLCSIM products

Function	PLCSIM Advanced V4.0	PLCSIM V17	PLCSIM V5.x
Runtime	Independent	Programming with STEP 7	Programming with STEP 7
User interface	Control Panel	Look&Feel of TIA Portal	Look&Feel of STEP 7 V5.x
Communication	Softbus, TCP/IP	Softbus	Softbus
Supported CPU families	S7-1500 (C, T, F), S7-1500R/H, SIMATIC Drive Controller, ET 200SP, ET 200SP F, ET 200pro, ET 200pro F, SIPLUS CPUs (S7-1500, S7-1500R/H and ET 200SP)	S7-1200 (F), S7-1500 (C, T, F), S7-1500R/H, SIMATIC Drive Controller, ET 200SP, ET 200SP F	S7-300, S7-300F, S7-400, S7-400F
API for co-simulation[1]	✓	-	-
Web server	✓, via TCP/IP	-	-
ODK	✓	-	-
OPC UA	✓, via TCP/IP	-	-
Process diagnostics	✓	✓	-
S7 communication	✓	Softbus	Softbus
Open user communication	✓, UDP via TCP/IP	Softbus	-
Secure Communication	✓, via TCP/IP	-	-
Traces[2]	✓	(✓)	-
Motion[3]	✓	(✓)	-
Protected blocks (KHP)	✓	✓, for S7-1500 CPUs only	-
Multiple instances	Up to 16	Up to 2	-
Support of distributed in-stances	✓, via TCP/IP	-	-
Virtual time	✓	-	-
Connection of real CPUs/HMIs	✓, via TCP/IP	-	-
DNS usage	✓	-	-
Virtual memory card	✓	-	-

Function	PLCSIM Advanced V4.0	PLCSIM V17	PLCSIM V5.x
Communication between the instances	-	PLCSIM as of V12 and PLCSIM V5.x can be installed and operated on the same PC or the same virtual machine.	
	-	Instances of PLCSIM as of V12 can communicate via Softbus with PLCSIM V5.x.	
	PLCSIM Advanced V3.0 and higher and PLCSIM V15 and higher can be installed and operated on the same PC or the same virtual machine. The communication between the two applications **cannot** be simulated.		-
	PLCSIM V5.4 SP8 is automatically installed with PLCSIM Advanced. The communication between the two applications can be simulated. Instances of PLCSIM Advanced can communicate via Softbus with PLCSIM V5.4 SP8.		

1 Via C++ and C# programs and simulation software

2 Can be monitored with PLCSIM V16 and higher in the TIA Portal; can also be monitored with PLCSIM Advanced V3.0 and higher on the Web server.

3 With PLCSIM V16 and higher, the axes are always in simulation mode irrespective of the axis configuration.
With PLCSIM Advanced V3.0 and higher, the axes can also be operated in "Real" mode over the API.

Product overview 2

2.1 What is S7-PLCSIM Advanced?

Using S7-PLCSIM Advanced, you can simulate your CPU programs on a virtual controller. You do not need any real controllers for this. You can configure your CPU with STEP 7 in the TIA Portal, program your application logic and then load the hardware configuration and the program into the virtual controller. From there you can run your program logic, observe the effects of simulated inputs and outputs and adapt your programs.

In addition to communicating via Softbus, S7-PLCSIM Advanced provides a full Ethernet connection and can thus also communicate distributed.

S7-PLCSIM Advanced enables interaction with native C++/C# programs or simulation software over the **user interface (API)**.

Application areas

- Verification of the user program (TIA Portal)
- Automatic 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.

Advantages

The use of S7-PLCSIM Advanced offers numerous advantages:

- Improve quality of automation projects by early error detection
- Avoid costs for hardware in simulation environments
- Reduced response times
- Reduce risk for commissioning
- Earlier training of operator is possible
- Increase production efficiency by optimizing program components
- Increase efficiency during replacement of machine components
- Increase efficiency during expansion of existing plants

2.2 Compatibility during upgrade

Compatibility of API and Runtime Manager versions

S7-PLCSIM Advanced V4.0 contains the Runtime Manager version V4.0 and the API versions V1.0 (SP1) to V4.0.

The installation of S7-PLCSIM Advanced V4.0 leads to an upgrade of an existing earlier version. The Runtime Manager of S7-PLCSIM Advanced V4.0 is compatible with projects that were created with earlier API versions. You can therefore continue to use already created projects.

Note

An API with a higher version number (e.g. V4.0) cannot connect to an earlier Runtime Manager version (e.g. V1.0).

Introductory information on the components of the Runtime simulation can be found in the section User interfaces (API) (Page 84).

Compatibility to TIA Portal and to CPU firmware versions

The firmware used in S7-PLCSIM Advanced V4.0 corresponds to that of a CPU S7-15xx V2.9.

The firmware is compatible to the TIA Portal versions V14 to V17.

Table 2- 1 Compatibility with CPU firmware versions

S7-PLCSIM Advanced	Supported CPU firmware version
V1.0 SP1	V1.8, V2.0
V2.0	V1.8 to V2.5
V2.0 SP1	V1.8 to V2.6
V3.0	V1.8 to V2.8
V4.0	V1.8 to V2.9

2.3 Security for S7-PLCSIM Advanced

Restrictions for security

Note the following restrictions when using S7-PLCSIM Advanced:

Authentication

- The user interfaces (API) do not have options for authentication and authorization. There is no protection using user accounts and passwords.
- The Runtime Manager communication is not protected by authentication.

Communication

- The multi-computer simulation communication is not encrypted.
- A TCP/IP port is opened on the PC for cross-network communication.
- The installed Npcap program library provides access to TCP/IP network communication.

Note

For cross-computer communication, it is recommended to use a closed simulation network that is not connected to a production network.

Know-how protection

Note

Know-how protected blocks

If know-how-protected blocks for the simulation support are enabled, the know-how protection is limited.

Note

CPU function libraries for ODK

The SO files for ODK are not know-how-protected. The customer is responsible for the SO files and its know-how protection.

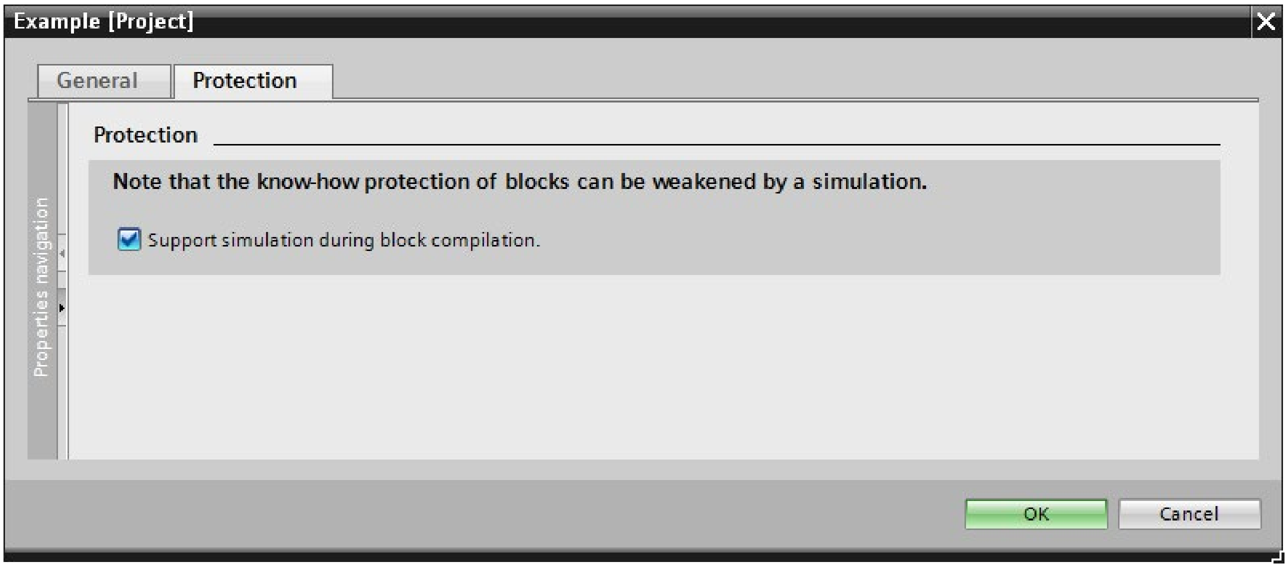
2.4 Simulations support

Requirement for simulation

Note

Enable simulation capability

To use a STEP 7 project with simulation, you must select the "Support simulation during block compilation" option in the "Protection" tab in the properties of the project and confirm with OK.



Protection

Note that the know-how protection of blocks can be weakened by a simulation.

☑ Support simulation during block compilation.

Figure 2-1 Enable simulation capability

Know-how protection

If a know-how-protected block is to be used for the simulation, it must be unlocked by entering a password. Only through unlocking can the "Simulation with SIMATIC S7-PLCSIM Advanced" option in the tab "General > Compilation" in the properties of the block be activated. Additional information can be found on the Internet (https://support.industry.siemens.com/cs/ww/en/view/109754928).

Global libraries

You cannot use know-how protection with global libraries, because the libraries are write-protected.

The "Simulation with SIMATIC S7-PLCSIM Advanced" option must be set when generating the blocks (source of the blocks).

2.5 Supported CPUs

Supported CPUs from the S7-1500 family

S7-PLCSIM Advanced V4.0 supports the simulation of the following CPUs:

Table 2- 2 Supported CPUs

Type	**Firmware version V1.8 to V2.9**	
Standard CPUs / Fail-safe CPUs[1]	CPU 1511-1 PN CPU 1513-1 PN CPU 1515-2 PN CPU 1516-3 PN/DP CPU 1517-3 PN/DP CPU 1518-4 PN/DP CPU 1518-4 PN/DP ODK CPU 1518-4 PN/DP MFP	CPU 1511F-1 PN CPU 1513F-1 PN CPU 1515F-2 PN CPU 1516F-3 PN/DP CPU 1517F-3 PN/DP CPU 1518F-4 PN/DP CPU 1518F-4 PN/DP ODK CPU 1518F-4 PN/DP MFP
Compact CPUs[2]	CPU 1511C-1 PN CPU 1512C-1 PN	-
ET 200SP CPUs[1]	CPU 1510SP-1 PN CPU 1512SP-1 PN	CPU 1510SP F-1 PN CPU 1512SP F-1 PN
Technology CPUs	CPU 1511T-1 PN CPU 1515T-2 PN CPU 1516T-3 PN/DP CPU 1517T-3 PN/DP CPU 1518T-4 PN/DP	CPU 1511TF-1 PN CPU 1515TF-2 PN CPU 1516TF-3 PN/DP CPU 1517TF-3 PN/DP CPU 1518TF-4 PN/DP
R/H CPUs[1]	CPU 1513R-1 PN CPU 1515R-2 PN CPU 1517H-3 PN	CPU 1518HF-4 PN
ET 200pro CPUs	CPU 1513pro-2 PN CPU 1516pro-2 PN	CPU 1513pro F-2 PN CPU 1516pro F-2 PN
SIMATIC Drive Controller	-	CPU 1504D TF CPU 1507D TF

[1] SIPLUS CPUs are supported. They are of the same design as the standard and fail-safe CPUs listed here with their own article numbers.

[2] The on-board I/O of the CPUs is not simulated. The simulation interface corresponds to the process image.

Unsupported CPUs

S7-PLCSIM Advanced does not support the simulation of the following CPUs:

- S7-1200 CPUs
- ET 200SP Open Controller CPU 1515SP PC
- Software Controller

2.6 Differences between a simulated and a real CPU

The virtual controller cannot fully simulate a real CPU down to the individual details. Even if a program is downloaded without errors to the CPU and running successfully, this does not necessarily mean that the virtual controller in the simulation behaves exactly like a real CPU.

Deterministic

S7-PLCSIM Advanced runs on a PC with the Windows operating system. Therefore, the scan cycle time and the exact time of actions in S7-PLCSIM Advanced are not the same as when these actions run on physical hardware. This is because that several programs share the processing resources on your PC.

To provide the best possible deterministic behavior under these conditions, S7-PLCSIM Advanced as of V2.0 requires one free Core (CPU core) per instance. Information on the minimum requirements for the computer hardware or a virtual machine can be found in the section System Requirements (Page 27).

If your program depends heavily on the time required to execute actions, then make sure that you do not evaluate your program based only on the results of the simulation time.

Know-how protection

Projects with know-how protection for blocks can only be simulated if they are enabled for simulation. You need the block password for this purpose.

Instructions

Instructions are simulated with a few exceptions, see Restrictions for instructions (Page 402).

Programs that are based on the instructions behave different than real CPUs in the simulation.

Display of the quantity structure

In STEP 7 the maximum quantity structure that is based on the CPU 1518-4 PN/DP is shown in the project navigation under "Program info" for all the CPUs.

The maximum quantity structure of the currently simulated CPU is displayed under "Online & Diagnose".

S7-1500R/H

For communication with other devices, you may have configured system IP addresses in STEP 7 for a redundant S7-1500R/H system. System IP addresses are not supported by S7-PLCSIM Advanced.

2.6.1 Restrictions for all supported CPUs

Bus systems

S7-PLCSIM Advanced does not simulate bus systems (PROFINET IO, PROFIBUS DP, backplane bus).

Intelligent IO devices (I-devices)

You can load a hardware configuration with configured I-devices on the PROFINET IO into the virtual controller. However, the I-device functionality is not supported by S7-PLCSIM Advanced.

I/O

S7-PLCSIM Advanced simulates the real CPU, but not configured I/O modules and the on-board I/O of the compact CPUs.

Communication modules

S7-PLCSIM Advanced does not support communication modules and the associated features such as "Access to PLC via communication module".

Diagnostics / diagnostic alarms

With S7-PLCSIM Advanced, simple diagnostics buffer entries can be simulated according to the PROFINET standard.

PROFIBUS-specific diagnostics (e.g. via DS0, DS1) and user-specific text lists are not supported.

Online and diagnostic functions

Some online and diagnostic functions are not very useful in the simulation and are therefore not supported. This includes the "Firmware update" function, for example.

Data logging

S7-PLCSIM Advanced does not simulate data logging.

Recipes

S7-PLCSIM Advanced does not simulate the use of recipes.

Copy protection

S7-PLCSIM Advanced does not simulate copy protection.

Limited support

S7-PLCSIM Advanced simulates some functions to a limited extent. You can find an overview in the section Restrictions, messages and solution (Page 399).

2.6.2 Notes

Password transfer when module is replaced (S7-1500)

Depending on the firmware version of the CPUs affected (the CPU to be replaced and the replacement CPU), you are either offered an update to the latest algorithm or prompted to assign a new password because the replacement CPU cannot use the existing password configuration.

If the CPU to be replaced and the replacement CPU are identical in terms of the algorithm used, no action is required: the password configuration and the other parameter settings are transferred.

S7-PLCSIM Advanced does not support any password encryption for CPU versions with firmware less than V2.0.

In order to use protection levels, the Web server and the access protection of the F-CPU in the simulation, click on the "Update password encryption" button. The button is located in the CPU properties in the "Protection & Security" tab under "Access level".

HMI devices and CPU protection levels

- S7-PLCSIM Advanced supports HMI devices as of version 14. Connections to HMI devices prior to V14 are not supported.
- S7-PLCSIM Advanced supports protection levels if the virtual S7-1500 controller is configured with a firmware version V2.0 or higher.
- It is possible to connect HMI devices as of V14 to virtual S7-1500 controllers that are configured with a firmware version V2.0 or higher, with or without protection levels.
- It is possible to connect HMI devices as of V14 to virtual S7-1500 controllers which are configured with a firmware version lower than V2.0 without protection levels.

Solution

To establish a connection to the HMI device V13 or earlier, you have to update this HMI device to version V14.

To establish a connection from the virtual controller that is configured with a firmware version lower than V2.0 to the HMI device, you have to remove existing protection levels from the project.

Safety system version V1.6 or V2.0 for fail-safe I/O

To successfully simulate and test a project with fail-safe input and output modules, you need to use safety system version V1.6 or V2.0 for the project. Simulation of the fail-safe input and output modules does not work correctly with an older version.

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.

Technology module TM Count - Error message of instruction High_Speed_Counter

When you are using S7-PLCSIM Advanced for the simulation of a high-speed counter in a TM Count technology module, the instruction High_Speed_Counter signals an error 16#80C7.

The instruction High_Speed_Counter expects that the module has set a bit for "Status ready" (STS_READY). Because S7-PLCSIM Advanced does not simulate the module behavior, the instruction signals an error.

The STS_READY bit is located in the input area of the module at offset 13.4. When the input area of your TM Count module starts at %I32, for example, the STS_READY bit is located at %I45.4.

To prevent this error message of the High_Speed_Counter instruction, set the STS_READY bit accordingly.

2.7 Password to protect confidential configuration data

As of STEP 7 V17, you have the option of assigning a password to protect confidential configuration data of the respective CPU as of FW version V2.9. This refers to data such as private keys that are required for the proper functioning of certificate-based protocols.

Assign a password to protect confidential configuration data

You assign the password in STEP 7, in the CPU properties, in the area "Protection & Security > Protection of the PLC configuration data".

You can use the same password for a PLCSIM Advanced instance as for the real CPU. This makes it easier for you to assign it uniquely.

PLCSIM Advanced stores the password encrypted in a file on the virtual memory card. The handling of the password is the same as with the real CPU.

Only the current, active Windows user and no other user is allowed to read the password for the protection of confidential configuration data on the computer.

Note

Note that your Windows password protects the password used to protect confidential configuration data. Therefore, you are not allowed to share the Windows password with other, untrustworthy users.

Additional information

Detailed information on the protection of confidential configuration data and on secure communication can be found in the Communication function manual (https://support.industry.siemens.com/cs/ww/en/view/59192925).

Special use cases

Move the virtual SIMATIC memory card to another virtual machine (e.g. SIMIT)

If you have not set a password to protect confidential configuration data, there are no restrictions on move operations.

If you have specified the password to protect confidential configuration data, the following restriction applies:

When you move the virtual memory card from one computer to another, you cannot start the PLCSIM Advanced instance on the new system.

Note

To ensure maximum security on your systems, the password for protecting confidential configuration data is not available on a new system. You must reset the password again in the new system.

Remedy:

The computer is part of an Active Directory.

The password encryption is linked to the Windows user. When you use the same Active Directory user in your domain on another computer, you can start PLCSIM Advanced instances there.

SIMIT archives the virtual SIMATIC memory card as a ZIP file. and another user/computer is trying to restart this simulation

Remedy:

1. Remove the password for protecting confidential configuration data from the STEP 7 project.
2. Load the project to the virtual controller.

You can move the virtual memory card to other computers without any restrictions.

Switching on the PLCSIM Advanced instance without valid password

When you switch on an instance and the password for the protection of confidential configuration data is incorrect, the instance restarts automatically and switches to the STOP operating state. ERROR LED flashes red.

Remedy:

1. Enter the password to protect the confidential configuration data in STEP 7 or via the SIMATIC Automation Tool.
2. Restart the instance.

Installing 3

3.1 Introduction

3.1.1 System requirements

You should preferably install PLCSIM Advanced on a SIMATIC Field PG M5 Advanced or comparable PC.

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

Table 3- 1 System requirements

	Hardware	**Virtual machine**
Processor	• At least one Intel® Core™ i7 6th Generation core per started instance • 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 (64-bit versions)

PLCSIM Advanced V4.0 supports the following operating systems:

- Windows 10 Home Version 1909
- Windows 10 Home Version 2004
- Windows 10 Home Version 2009/20H2
- Windows 10 Professional Version 1909
- Windows 10 Professional Version 2004
- Windows 10 Professional Version 2009/20H2
- Windows 10 Enterprise Version 1909
- Windows 10 Enterprise Version 2004
- Windows 10 Enterprise Version 2009/20H2
- Windows 10 Enterprise 2016 LTSB
- Windows 10 Enterprise 2019 LTSC
- Windows Server 2016 Standard (full installation)
- Windows Server 2019 Standard (full installation)

For more detailed information on operating systems, refer to the help on Microsoft Windows or the Microsoft homepage.

Some protocols might also support additional Windows versions. You can find more information in the product-specific requirements or check the compatibility with the compatibility tool. The compatibility tool is available on the Internet (https://support.industry.siemens.com/kompatool/pages/main/index.jsf?).

Note

Make sure that the Windows operating system you are using is up to date.

Virtualization platforms

You can install STEP 7 and PLCSIM Advanced on a virtual machine. For this purpose, use one of the following virtualization platforms in the specified version or a newer version:

- VMware vSphere Hypervisor (ESXi) 6.7
- VMware Workstation Pro 15.5.0
- VMware Player 15.5.0
- Microsoft Hyper-V Server 2019

The information that you need to install STEP 7 (TIA Portal) on a virtual machine is available on the Internet (https://support.industry.siemens.com/cs/ww/en/view/78788417).

3.1.2 Restrictions due to antivirus programs

NOTICE
Restrictions due to antivirus programs
Virus scanners that monitor the behavior of processes and communication can have a significant impact on the performance of the runtime and communication of PLCSIM Advanced.

Remedy

You can decrease restrictions during installation and runtime of PLCSIM Advanced. Also define exceptions for the virus scanner for secure files and directories.

Procedure

Add the following directories 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 directories to the exceptions. The procedure is described in the instructions of the respective manufacturer.

Note

Readme

You can obtain updates to the topic as downloads on the Internet (https://support.industry.siemens.com/cs/us/en/view/109773483).

Supported antivirus programs

PLCSIM Advanced supports Trend Micro Office Scan 12.0.

Known problems and limitations

Kaspersky

When using the Anti-Virus virus scanner from Kaspersky, the network settings may not be set correctly during the installation of PLCSIM Advanced. The result is that communication via TCP/IP cannot be used (error code -50).

Remedy

Check your network settings as described in the section Activating distributed communication (Page 45).

3.1.3 Licenses

Floating license

PLCSIM Advanced is supplied with a floating type license which is version and/or time-dependent. The license can be stored locally and shared for a network.

Note

Validity

A license is valid for two instances within a PLCSIM Advanced installation.

PLCSIM Advanced V4.0 can only be used with a V4.0 license.

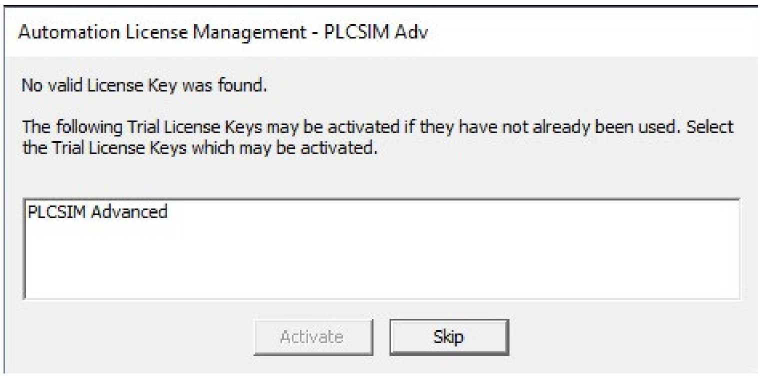
Handling of licenses is described in the Help for SIMATIC Automation License Manager (ALM).

3.1.4 Trial License

A license is available for the limited period of 21 days for S7-PLCSIM Advanced V4.0. After this Trial License has elapsed, the instance is no longer started.

Activating the Trial License

As soon as you start an instance in the Control Panel, the Automation License Manager (ALM) searches the network for a valid license. If a Floating License is available for S7-PLCSIM Advanced, the ALM offers the Trial License for activation.


Automation License Management - PLCSIM Adv

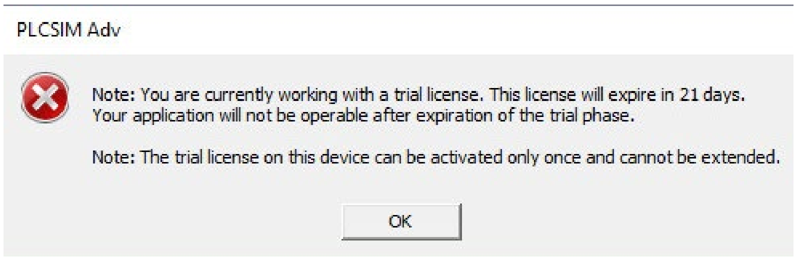
No valid License Key was found.

The following Trial License Keys may be activated if they have not already been used. Select the Trial License Keys which may be activated.

PLCSIM Advanced

Figure 3-1 Activating the Trial License

A message at the start of the instances shows the remaining number of days.


PLCSIM Adv

Note: You are currently working with a trial license. This license will expire in 21 days.
Your application will not be operable after expiration of the trial phase.

Note: The trial license on this device can be activated only once and cannot be extended.

Figure 3-2 Trial License message

Note

Remote access

With remote access, the message must be confirmed on the PC on which the instance was started.

Timeout alarm

If you do not confirm the message for the license in a certain amount of time, the instance is not started and the following message appears:

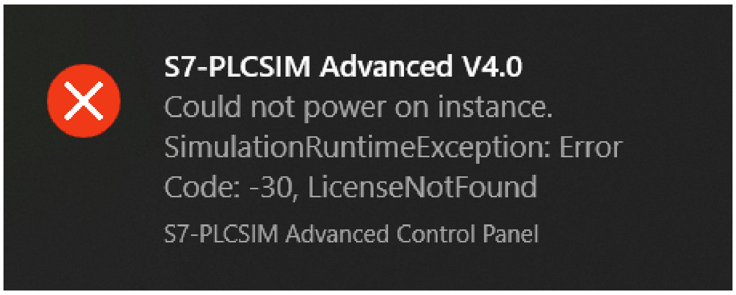

S7-PLCSIM Advanced V4.0
Could not power on instance.
SimulationRuntimeException: Error Code: -30, LicenseNotFound
S7-PLCSIM Advanced Control Panel

Figure 3-3 Timeout alarm

Solution

Start the instance again and confirm the message for the license.

API functions for licenses

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`, no valid license for PLCSIM Advanced is available, but a "TIA Portal Test Suite" license.
    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`, if no valid license for 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.

3.1.5 Installation log

The log files contains automatically recorded information on the following installation processes:

- Installation of S7-PLCSIM Advanced
- Change or update of installation of S7-PLCSIM Advanced
- Repair of an existing installation of S7-PLCSIM Advanced
- Uninstallation of S7-PLCSIM Advanced

You can evaluate installation errors and warnings using the log files. You can troubleshoot the installation yourself or contact Siemens Technical Support. Product Support personnel need information from the installation log to analyze the problem. Send the folder with the log files as a ZIP file to Support.

Memory location of the installation log

The memory location of the log file depends on the operating system. To open the folder with the log files, enter the environment variable "%autinstlog%" in the address bar in Windows Explorer. Alternatively, you reach the appropriate directory by entering "cd %autinstlog%" in the command line.

The log files are named as follows:

- "SIA_S7-PLCSIM_Advanced_V04@<DATE_TIME>.log"
- "SIA_S7-PLCSIM_Advanced_V04@<DATE_TIME>_summary.log"

Setup_Report (CAB file)

The installation log and other required files are stored in a log file. This file can be found at "%autinstlog%\Reports\Setup_report.cab".

A separate CAB file with a date ID is saved for each installation.

If you need help during installation, send this CAB file to Siemens Technical Support for troubleshooting.

3.2 S7-PLCSIM Advanced

The S7-PLCSIM Advanced package contains the following software:

- S7-PLCSIM Advanced
- Automation License Manager
- S7-PLCSIM V5.4
- .NET Framework
- Npcap

The package is available as download and on DVD.

- SIMATIC S7-PLCSIM Advanced V4.0 Floating License
- Upgrade SIMATIC S7-PLCSIM Advanced V3.0 → V4.0

After installing PLCSIM Advanced, keep the DVD in a secure, easily accessible place.

Setup program

You can use the Setup program to change, repair or uninstall your installation, if necessary.

3.3 Installing S7-PLCSIM Advanced

Installation requirements

The Setup program starts automatically with a double-click on the download package or when you insert the DVD in the drive. Make sure that the following conditions are met before you begin the installation process:

- The hardware and software of the computer meet the system requirements.
- You have administrator rights on the installation computer.
- No other programs are active. This also applies to the Siemens Automation License Manager and other Siemens applications.
- All S7-PLCSIM versions prior or equal to V14 are uninstalled.

Note

Security settings

For licensing via the ALM, you must agree during installation that port 4410 for TCP can be entered as an exception in the Windows Firewall (procedure step 5).

Note

Use of virus scanners

Note the information provided in section Restrictions due to antivirus programs (Page 29).

Installing S7-PLCSIM Advanced

To install, follow these steps:

1. Double-click the download package or insert the installation medium into the DVD drive of your computer. The setup program starts up automatically, provided you have not disabled the Autostart function on the computer. If the setup program does not start up automatically, start it manually by double-clicking the "Start.exe" file. The "General settings" window is displayed.
2. Click the "Read installation notes" button. After you have read the notes, close the file.
3. Click the "Read product information" button. After you have read the information, close the file.
4. Click "Next". The window with the products to be installed is displayed.
5. Select the products to be installed.
6. Click the "Browse" button if you want to change the default installation path. The installation path must not exceed 89 characters. The path name must not contain any UNICODE characters. If you select a different installation path than the default installation path, the desktop icon may not be displayed correctly.

7. Click "Next". The window with the security settings is displayed. To continue the installation, select the check box at the bottom of the screen to accept changes to the security and permissions settings of your system.
8. Click "Next". The window with the installation settings is displayed. You can save or print a report of the settings by clicking "Save report" or "Print report". Check the settings for correctness. If you want to make any changes, click "Back" until you reach the point in the installation process where you want to make changes. Once you have completed your changes, click "Next".
9. The overview screen shows your installation details. Click the "Install" button. The installation then starts.
10. After completion of the setup program, you must restart your computer. Select "Yes, I want to restart the computer now" to restart the computer immediately or select "No, I will restart computer later" to restart the computer later.
11. Click "Restart". If the computer is not restarted, click "Finish".

Error during installation of S7-PLCSIM Advanced

When PLCSIM Advanced is installed, any existing installation of S7-PLCSIM is displayed.

A requirement for installation of S7-PLCSIM Advanced is that no other S7-PLCSIM installation prior or equal to V14 is located on the same computer.

Even though no installation of S7-PLCSIM is displayed in the "Programs and Features" list, it is still possible that the computer has an existing installation.

Remedy

Run the setup for S7-PLCSIM prior or equal to V14 and uninstall the program.

When the setup is not available, download the setup files for S7-PLCSIM via Siemens Mall (https://support.industry.siemens.com/cs/ww/en/view/65601780).

3.4 Changing S7-PLCSIM Advanced

Requirements

The following conditions must be met before you can start changing the installation:

- The hardware and software of the computer meet the system requirements.
- You have administrator rights on the installation computer.
- No other programs are active.

Procedure

To change your S7-PLCSIM Advanced installation, follow these steps:

1. Double-click the download package or insert the installation medium into the drive. The setup program starts up automatically, provided you have not disabled the Autostart function on the computer. If the setup program does not start up automatically, start it manually by double-clicking the "Start.exe" file.
2. Follow the prompts until you reach the "Configuration" window.
3. Select the "Change upgrade" check box.
4. Follow the remaining prompts to change your installation.
5. Complete the installation operation by restarting your computer.

Note

Target directory

You cannot change the target directory because you are changing an existing installation.

3.5 Repairing S7-PLCSIM Advanced

Requirements

The following conditions must be met before you can start repairing the installation:

- The hardware and software meet the system requirements.
- You have administrator rights on the installation computer.
- No other programs are active.

Procedure

To repair your installation, follow these steps:

1. Double-click the download package or insert the installation medium into the drive. The setup program starts up automatically, provided you have not disabled the Autostart function on the computer. If the setup program does not start up automatically, start it manually by double-clicking the "Start.exe" file.
2. Follow the prompts until you reach the "Configuration" window. Select the "Repair" check box.
3. Follow the remaining prompts to repair your installation.
4. Complete the repair operation by restarting your computer.

3.6 Uninstalling S7-PLCSIM Advanced

You have two options for uninstalling S7-PLCSIM Advanced:

- You uninstall the program using the Windows Control Panel.
- You uninstall the entire product using the Setup program.

Uninstalling S7-PLCSIM Advanced using the Windows Control Panel

Proceed as follows:

1. Double-click the "Programs and Features" option in the Windows Control Panel.
2. Right-click on "SIMATIC S7-PLCSIM Advanced V4.0" and select "Uninstall".
3. Follow the prompts for uninstallation.
4. Complete the uninstallation operation by restarting your computer.

   If you do not perform a restart, the Runtime Manager continues running.

If problems occur when uninstalling PLCSIM Advanced using the Windows Control Panel, use the installation medium for uninstalling.

Uninstalling S7-PLCSIM Advanced using the Setup program

Proceed as follows:

1. Double-click the download package or insert the installation medium into the drive. The setup program starts up automatically, provided you have not disabled the Autostart function on the computer. If the setup program does not start up automatically, start it manually by double-clicking the "Start.exe" file.

   If you do not perform a restart, the Runtime Manager continues running.
2. Follow the prompts until you reach the "Configuration" window. Your previous installation is detected. Select the "Uninstall" check box.
3. Follow the prompts for uninstallation.
4. Complete the uninstallation operation by restarting your computer.

   If you do not perform a restart, the Runtime Manager continues running.

Uninstalling additional software

During uninstalling the following software from the S7-PLCSIM Advanced package remains installed:

- Automation License Manager
- S7-PLCSIM V5.4
- .NET Framework

If you also want to uninstall this software, use the Windows Control Panel.

Communication paths 4

Local and distributed communication

The following paths are open for communication between STEP 7 V15 or higher and the instances of PLCSIM Advanced user interfaces:

Table 4- 1 Local and distributed communication

Communication paths	**Local**	**Local**	**Distributed**
Protocol	Softbus	TCP/IP	TCP/IP
Communication interface in PLCSIM Advanced	PLCSIM	PLCSIM Virtual Ethernet Adapter	PLCSIM Virtual Ethernet Adapter
STEP 7 and instances	On a PC / VM	On a PC / VM	Distributed
Communication...			
between STEP 7 and instances	Yes	Yes	Yes
among instances	Yes	Yes	Yes
via OPC UA server and Web server	No	Yes	Yes
between an instance and a real hardware CPU	No	No	Yes
between an instance and a real HMI V14 and higher	No	No	Yes
between an instance and a simulated HMI V14 and higher	Yes	Yes	Yes
Secure communication...			
via Secure Open User Communication (secure TCP communication) V17 and higher	No	Yes	Yes
via OPC UA Server V17 and higher	No	Yes	Yes
via HTTPS connections to the Web server V17 and higher	No	Yes	Yes

Softbus

Softbus is a communication path via a virtual software interface.

The communication is limited to a local PC or a virtual machine. The advantage here is that no data can be accidentally downloaded to a hardware CPU or communicate with real hardware.

Selecting a communication interface

You program the communication interface via the user interfaces (API) or select them in the Control Panel under "Online Access". The setting is valid for all generated instances. The default setting is the communication via "PLCSIM" (Softbus).

Additional network settings are necessary for the distributed communication via the "PLCSIM Virtual Ethernet Adapter" (TCP/IP), see Network addresses in the simulation (Page 57).

API functions for selecting the communication interface

- GetCommunicationInterface() (Page 141)
- SetCommunicationInterface() (Page 141)
- CommunicationInterface { get; set; } (Page 142)

See also

Interfaces - Information and settings (Page 138)

4.1 Local communication

Local communication can be performed via the Softbus protocol or TCP/IP.

For local communication, the PLCSIM Advanced instance is on the same PC or on the same virtualization platform such as STEP 7 or another communication partner.

Local communication via Softbus

Local communication is performed via Softbus in PLCSIM Advanced by default.

This ensures that no data can be accidentally downloaded to a hardware CPU or that there is communication with real hardware.

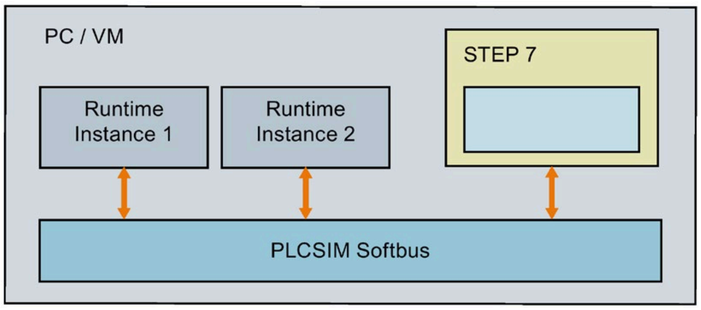


Figure 4-1 Local communication via Softbus

Local communication via TCP/IP

Communication is performed via the PLCSIM Virtual Ethernet Adapter, a virtual network interface that behaves like a real network interface.

Note

Local communication via TCP/IP

Make sure that communication is only local and cannot be downloaded to real hardware. For this, there must be no other adapters of your Windows PC configured in the physical network and in the subnet protocol of the PLCSIM Virtual Ethernet adapter. Microsoft KB 175767 provides background.

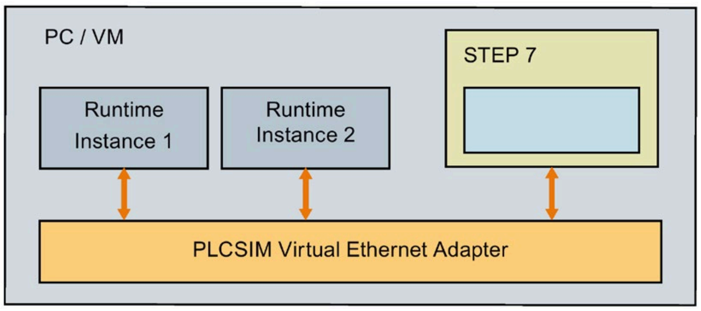


Figure 4-2 Local communication via TCP/IP

Additional information

See error code SREC_COMMUNICATION_INTERFACE_NOT_AVAILABLE for the function `PowerOn()` in the section Operating state (Page 156).

4.2 Communication via TCP / IP

Distributed communication

Distributed communication via TCP/IP means that the PLCSIM Advanced instances communicate with the other devices via the Virtual Switch . Communication is possible with real or simulated CPUs, real or simulated HMIs.

The PLCSIM Virtual Switch must be activated on the PLCSIM Virtual Ethernet Adapter for instances on the network to be visible.

Each CPU interface can be reached from the PLCSIM Virtual Ethernet Adapter and requires a unique IP address.

The PLCSIM Virtual Ehternet Adapter must be in the same IP number range (subnet mask) as the IP address of the controller.

The IP address of the controller must be unique throughout the entire, accessible network.

Example 1: Distributed communication

In the following example, STEP 7 is on a PC and the PLCSIM Advanced instances are on another PC or a virtual machine. The PCs are connected via their real Ethernet adapter.

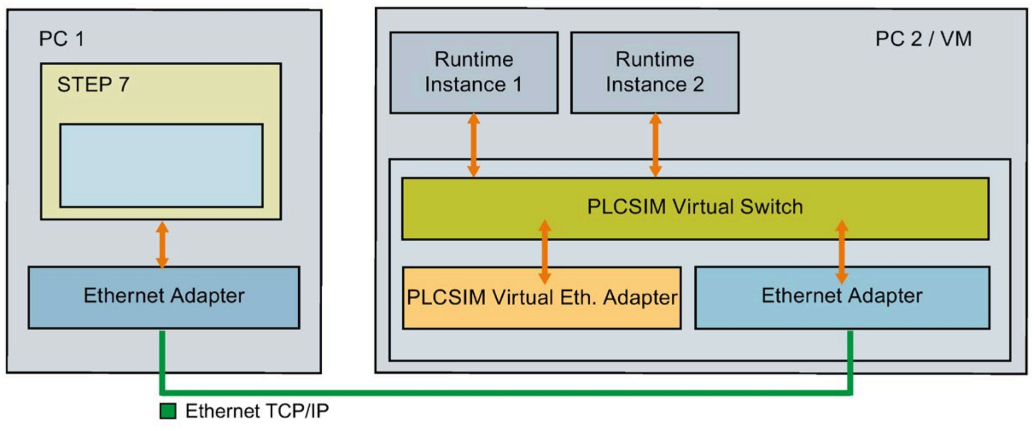


Figure 4-3   Distributed communication via Ethernet

Example 2: Distributed communication on a PC

In the following example, STEP 7 is on a PC and the PLCSIM Advanced instances are on a virtual machine on the same PC. PC and virtual machine are connected via the (virtual) network adapters.

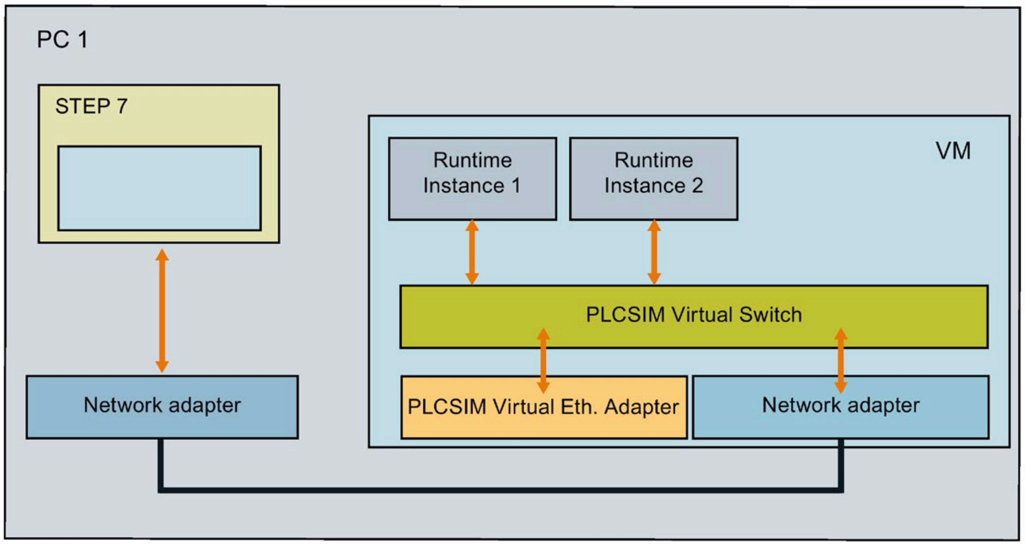


Figure 4-4 Distributed communication via network adapters

Required settings in the "Virtual Machine Settings" dialog using the VMware visualization platform as an example

If you have opened STEP 7 (TIA Portal) and your project within the virtual machine, enable the following options for your online connection as follows:

1. Right-click on the VM and select "Settings" or select the menu "VM > Settings".
2. Open the "Virtual Machine Settings" dialog via the menu command "Player > Manage > Virtual Machine Settings".
3. Then click "Network Adapter" in the "Hardware" tab and activate the following options in the right window:
   – Connected
   – Connect at power on
   – Bridged: Connected directly to the physical network
   – Replicate physical network connection state
4. Click the "Configure Adapters" button and activate your network connection, for example "Intel(R)82574L LM Gigabit Network Connection".
5. Confirm the setting with OK and exit the "Virtual Machine Settings" dialog with OK.

Example 3: Distributed communication

The following example shows a structure with PCs on which distributed STEP 7, PLCSIM Advanced instances and virtual machines with PLCSIM Advanced instances are running.

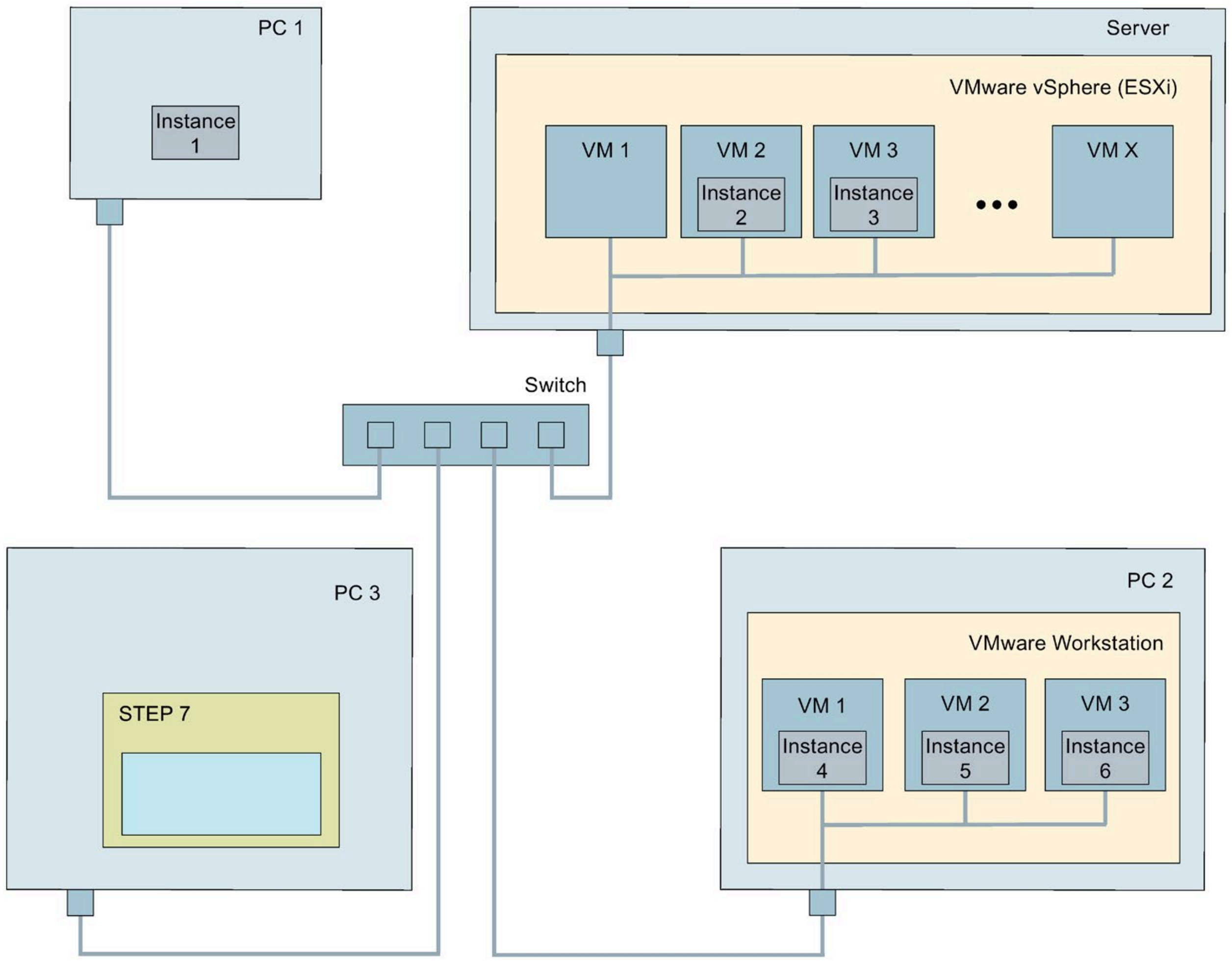


Figure 4-5 Distributed communications with PCs and virtual machines

4.3 Enable distributed communication

By default, the PLCSIM Virtual Switch can only communicate locally. For a distributed, i.e. multi-computer, communication to be possible, you must activate the PLCSIM Virtual Switch for a real network adapter.

Note

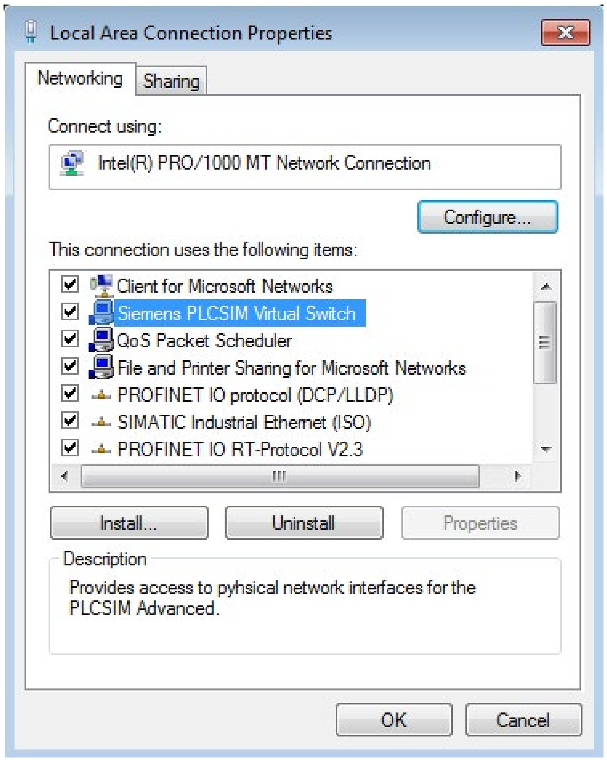
Network adapter

Make sure that only one network adapter of the PLCSIM Virtual Switch is activated. The Control Panel of PLCSIM Advanced checks the activation and may report an incorrect configuration (error code -50).

Activate PLCSIM Virtual Switch

To make the PLCSIM instances visible on the network and to reach other devices, activate the PLCSIM Virtual Switch in the Control Panel of PLCSIM Advanced or under Windows:

1. To do this, open the "Network and Sharing Center" in the Windows Control Panel.
2. Open the properties of the desired network adapter, for example, for the "Local Area Connection".
3. Select the check box for the "Siemens PLCSIM Virtual Switch" and confirm with OK.



Connect using:
Intel(R) PRO/1000 MT Network Connection

This connection uses the following items:
- [✓] Client for Microsoft Networks
- [✓] Siemens PLCSIM Virtual Switch
- [✓] QoS Packet Scheduler
- [✓] File and Printer Sharing for Microsoft Networks
- [✓] PROFINET IO protocol (DCP/LLDP)
- [✓] SIMATIC Industrial Ethernet (ISO)
- [✓] PROFINET IO RT-Protocol V2.3

Description
Provides access to pyhsical network interfaces for the PLCSIM Advanced.

Figure 4-6 Activate PLCSIM Virtual Switch

Accessible devices

When the PLCSIM Virtual Switch is activated, STEP 7 shows the devices available on the Virtual Ethernet Adapter in the project tree.

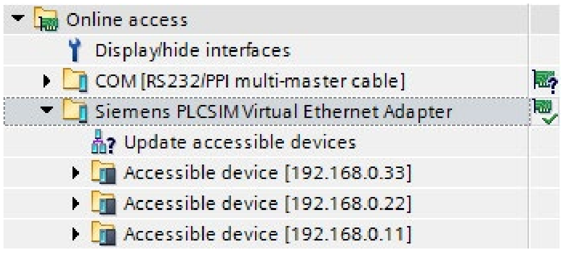

Online access
Display/hide interfaces
COM [RS232/PPI multi-master cable]
Siemens PLCSIM Virtual Ethernet Adapter
Update accessible devices
Accessible device [192.168.0.33]
Accessible device [192.168.0.22]
Accessible device [192.168.0.11]

Figure 4-7 Accessible devices on the Virtual Ethernet Adapter

Distributed communication via WLAN

When using distributed communication via WLAN, it may happen that the NPcap program library installed by PLCSIM Advanced does not work with the integrated WLAN adapter of the PC. In this case, no WLAN connection can be established.

Remedy

Use the wired network adapter of the PC/notebook and connect a WLAN adapter upstream.

Simulation 5

5.1 Simulate CPU

5.1.1 Basic procedure for the simulation

The following overview shows the basic steps to perform simulation with an instance of a virtual controller.

Requirements

The following requirements must be met for starting simulation via local communication:

- STEP 7 as of V14 and S7-PLCSIM Advanced V4.0 are installed on the same PC.
- The CPU hardware is configured in STEP 7.

Note

Enable simulation support

In the "Protection" tab in the properties of the project in STEP 7, select the check box "Support simulation during block compilation"; see Simulations support (Page 19).

Create and activate an instance via the Control Panel

- Open PLCSIM Advanced Control Panel (see section Control Panel - User interface (Page 48))
- Open the "Start Virtual S7-1500 PLC" options
- Enter a name for an instance
- Selecting the CPU family
- Create an instance using the "Start" button

In STEP 7, perform the download and start the simulation

- Download the program to the virtual controller (see section Download (Page 55))
- Switch the virtual controller to RUN to start the simulation
- Perform diagnostics

5.1.2 Control Panel - User interface

5.1.2.1 S7-PLCSIM Advanced Symbol

After installing PLCSIM Advanced, the following icons are on the Windows desktop:

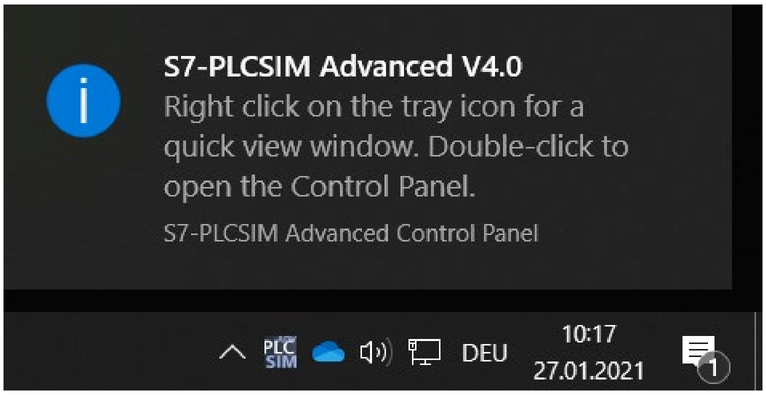
Figure 5-1 PLCSIM Advanced Symbol

A double-click on the symbol opens the Control Panel for PLCSIM Advanced. If the Control Panel is in the background, it is moved to the foreground with another double-click.

You can use Windows functions to permanently display the icon in the system tray of the taskbar.

Opening a graphical interface

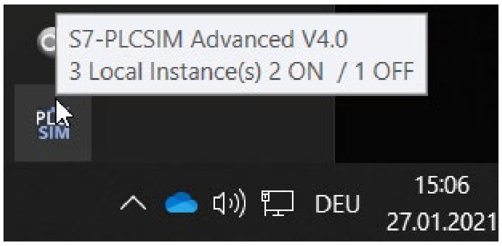
Right-clicking the icon in the taskbar opens the Control Panel with the quick view. Double-click to start the Control Panel as a window.



S7-PLCSIM Advanced V4.0
Right click on the tray icon for a quick view window. Double-click to open the Control Panel.
S7-PLCSIM Advanced Control Panel

Figure 5-2 Opening a graphical interface

You can use the mouse-over function to display messages about the current status of the instances.



S7-PLCSIM Advanced V4.0
3 Local Instance(s) 2 ON / 1 OFF

Figure 5-3 Example: Message in the taskbar

5.1.2.2 Graphical interface

The graphical interfaces synchronize by means of API commands. They are optional and are not needed to operate PLCSIM Advanced via the API.

S7-PLCSIM Advanced V4.0 provides the Control Panel with two views.

- **Control Panel as quick view**

  Right-clicking on the icon in the taskbar opens the quick view.

  Clicking on an empty area on the desktop minimizes the quick view. The instances are not affected.

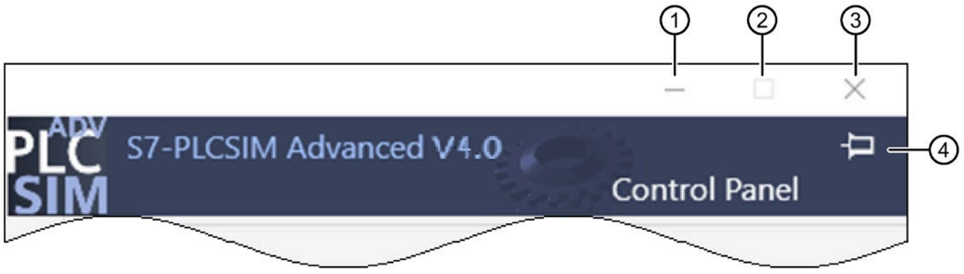
- **Control Panel as window**

  Double-clicking the icon on the desktop or in the taskbar opens the Control Panel as a window.

Control Panel as window

Unlike the quick view, you can operate the Control Panel with the buttons in the title bar.

You can close this window without exiting the simulation Runtime process.



① Stores the Control Panel as icon in the taskbar.

② No function. The window size cannot be changed.

③ Closes the Control Panel and stores it in the system tray of the taskbar.
The instances and the simulation Runtime process remain active.
This function therefore differs from the Exit function ✖.
The Exit function switches off the local instances, logs them off and closes the Control Panel.

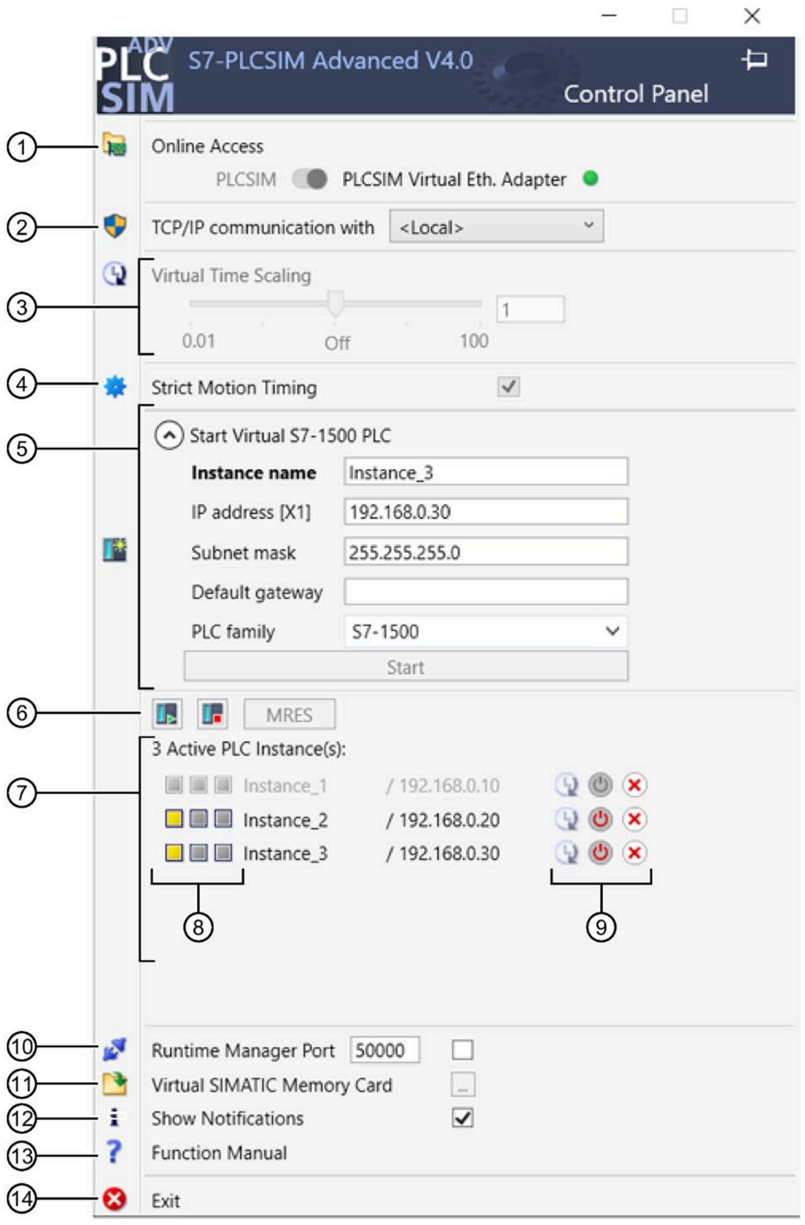
④ Pins the Control Panel on the screen so that it remains in the foreground.

Figure 5-4 Control Panel: Title bar

5.1.2.3 S7-PLCSIM Advanced Control Panel

The Control Panel is available in English in version V4.0.

Design



①	Online access	Switch to select the communication interface
②	TCP/IP communication	Selection of network adapter for distributed communication
③	Virtual time	Slider to adjust the scaling factor
④	Strict Motion Timing	Here you disable the overrun detection for Motion Control (OB MC-Servo [OB91]). When the check box is selected (default), overruns are detected. However, you can only change the setting of the overrun detection in the Control Panel for as long as no instance has been registered yet. The set behavior then applies to all subsequent registered instances. You can adapt the behavior for individual instances via the API.
⑤	**Start Virtual S7-1500 PLC**	Opens and closes the input boxes for creating the instance (virtual controller).
	• Name of the instance	Here you enter a unique name for the instance. Enter a minimum of 3, a maximum of 64 characters. If the name is unique in the network, the button "Start" is enabled.
	• IP address • Subnet mask • Standard gateway	The input boxes are visible when you switch the communication interface to "PLCSIM Virtual Ethernet Adapter". The IP address is entered automatically.
	• CPU family	Here you select the CPU family to be simulated.
	• "Start" button	Create with the button and start the instance.
⑥	Buttons	Buttons for operating the selected instances.
⑦	Instance list	The list shows the available **local** instances. The instances can be resorted using the mouse cursor.
⑧	LED displays	The meaning of the LED is displayed when you move the mouse over it.
⑨	Icons	Icons for operating the instance
⑩	Runtime Manager Port	Here you open a port on the local PC.
⑪	Virtual SIMATIC Memory Card	You open the storage location of the virtual memory card with a double-click to the left of "...". You can change the path to the virtual memory card with a double-click on "...".
⑫	Display messages	Here you disable the PLCSIM Advanced messages in the Windows task bar for the duration of the operation.
⑬	Function manual	This is where you open the S7-PLCSIM Advanced Function Manual in a standard PDF viewer.
⑭	Exit	Exit logs off all instances and closes the Control Panel.

Figure 5-5 Control Panel V4.0

Switch for communication interface

Use the switch to select the communication interface for all instances to be created:

- "PLCSIM" corresponds to the local communication via softbus (default).
- "PLCSIM Virtual Ethernet Adapter" corresponds to the communication via TCP/IP.

The setting applies to all other instances. The selected communication interface for starting an instance is maintained until all instances are shut down.

When an instance is already started, it sets "its" communication interface as the default for other instances.

To change the communication interface, switch off all instances and enable the other interface.

TCP/IP communication

You can select a real network adapter from the drop-down list during operation. You thus activate the PLCSIM Virtual Switch and establish TCP/IP communication between the instances and the real network.

The <Local> setting disables the PLCSIM Virtual Switch and disconnects the instances from the real network. Only local TCP/IP communication over virtual adapter is possible in this case.

Virtual time

You must enable the virtual time for each instance using the icon . Use the slider or the mouse wheel to select the scaling factor for the virtual time.

The selected scaling factor applies to the instances for which the virtual time is enabled.

Clicking on "Off" restores the default (1) again. For further information see Virtual and Real Time (Page 75).

Creating an instance (locally) and starting it

To create an instance, enter a unique name under "Instance Name". If the name already exists in the directory of the Virtual SIMATIC Memory Card, the existing instance is started.

In the "PLC family" drop-down list, you select a CPU family:

- S7-1500
- S7-1500R/H
- ET 200SP
- ET 200pro

Create the instance with the "Start" button and start this instance.

The instance/virtual controller is initialized with the first download from the TIA Portal.

Instance list

The list contains the instances that are available locally on the PC or virtualization platform. Instances that have already been started on the runtime API are detected and displayed in the list.

Select the operating mode of the instance with the "RUN" and "STOP" buttons. Select one or more instances for this purpose. Perform a memory reset with the "MRES" button.

The LED displays show the status of the instance that corresponds to those of the hardware CPU.
RUN and STOP are displayed depending on the current operating state of the instance.

You can "operate" the instance with icons:

Enable virtual time, apply scaling factor for the virtual time, disable virtual time,

Switch on instance ("`PowerOn`"), Switch off instance ("`PowerOff`"),

Switch off instance and log off from Runtime Manager ("`Unregister`")

Runtime Manager Port

A remote connection can be established to another Runtime Manager via the specified port. The value must be greater than 1024.

If you select the check box, the port remains stored. You can use the remote connection without having to make this setting every time you start the Control Panel. To use this functionality, the Control Panel must be started and running in the background.

Virtual SIMATIC Memory Card

The user program, the hardware configuration and the retentive data is stored on the Virtual SIMATIC Memory Card. Use the buttons to adapt the path to the virtual memory card or open the previously saved path in an Explorer window.

Display messages

Each time the Panel starts, help information and messages relating to the Control Panel are displayed, for example, when changing the IP address or when a license is missing. Disable the display if you do not need the messages.

Exit - Log off all instances

- The command switches off all local instances on the PC or the VM and logs them off from the Runtime Manager and closes the Control Panel.
- This command closes the Runtime Manager if there are no remote connections to other Runtime Managers.
- If the Runtime Manager has remote connections to instances on additional PCs, these instances and the Runtime Manager continue to run.

5.1.2.4 Importing instances

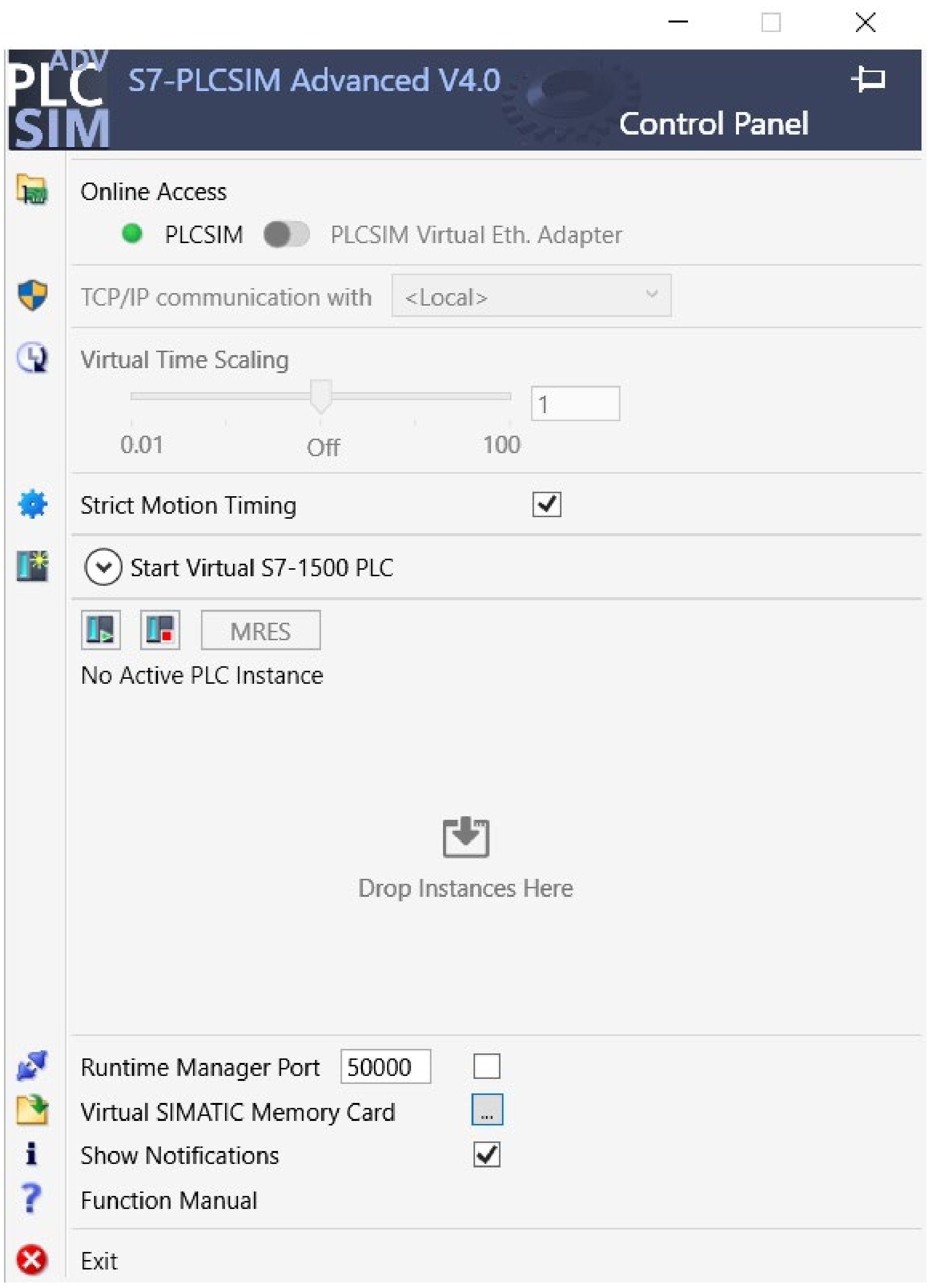
Requirement

This function is only available if you do not start the Control Panel with admin rights.

Importing instances

You can use the drag-and-drop function to import instances from a folder directly into the instance list of the Control Panel.

1. Open a folder with instances, for example, using the "Virtual SIMATIC Memory Card" button.
2. Select one or more instances and drag them into the highlighted area.



Online Access
● PLCSIM PLCSIM Virtual Eth. Adapter
TCP/IP communication with <Local>
Virtual Time Scaling 1
0.01 Off 100
Strict Motion Timing ✓
Start Virtual S7-1500 PLC
MRES
No Active PLC Instance
Drop Instances Here
Runtime Manager Port 50000 ☐
Virtual SIMATIC Memory Card ...
Show Notifications ✓
Function Manual
Exit

Figure 5-6 Control Panel: Importing instances

5.1.3 Download

Requirements

You can download the STEP 7 project to the virtual controller when the following conditions are met:

- The instance is created via the Control Panel.
- The check box "Support simulation during block compilation" is selected.

Selecting the communication interface

In the Download dialog box, select the PG/PC interface:

- "PLCSIM" for download via Softbus
- "Siemens PLCSIM Virtual Ethernet Adapter" for download via TCP/IP
- For distributed communication the real adapter that is connected to the network

Display in the download dialog

The dialog in STEP 7 at the first download of the CPU shows the compatible PLCSIM Advanced instances.

If the instance has not yet been configured after the first download only **one** interface is visible and it appears with the device type "CPU-1500 Simulation".

If the instance has been configured, the number of interfaces visible is determined my the number the CPU type has.

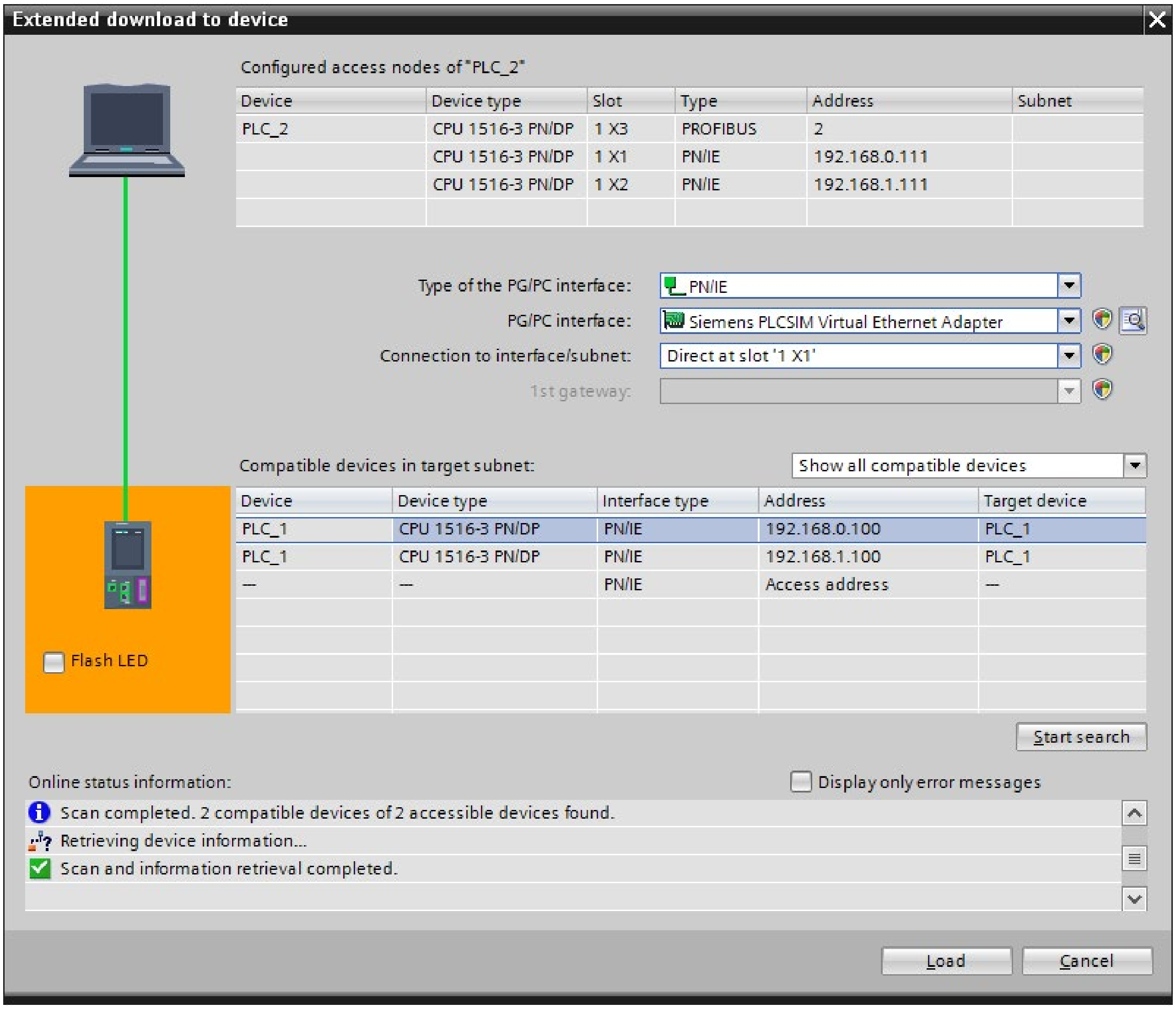
The lifelist shows the interfaces of an instance with their IP addresses.

Perform download

1. Select the PG/PC interface.
2. Click "Download".

   → In the "Load preview" window, STEP 7 shows the message "The downloads are performed on a simulated CPU".

   → After the first download, the PLCSIM Advanced instance displays the CPU type.



Configured access nodes of "PLC_2"

Device	Device type	Slot	Type	Address	Subnet
PLC_2	CPU 1516-3 PN/DP	1 X3	PROFIBUS	2	
	CPU 1516-3 PN/DP	1 X1	PN/IE	192.168.0.111	
	CPU 1516-3 PN/DP	1 X2	PN/IE	192.168.1.111	

Type of the PG/PC interface: PN/IE
PG/PC interface: Siemens PLCSIM Virtual Ethernet Adapter
Connection to interface/subnet: Direct at slot '1 X1'
1st gateway:

Compatible devices in target subnet: Show all compatible devices

Device	Device type	Interface type	Address	Target device
PLC_1	CPU 1516-3 PN/DP	PN/IE	192.168.0.100	PLC_1
PLC_1	CPU 1516-3 PN/DP	PN/IE	192.168.1.100	PLC_1
—	—	PN/IE	Access address	—

Online status information:

Scan completed. 2 compatible devices of 2 accessible devices found.
Retrieving device information...
Scan and information retrieval completed.

Figure 5-7 Example: Download via the "PLCSIM Virtual Ethernet Adapter" (TCP/IP) after naming

5.1.4 Network addresses in the simulation

5.1.4.1 Siemens PLCSIM Virtual Ethernet Adapter

IP address

At the PLCSIM Virtual Ethernet Adapter you assign a static IP address or obtain an IP address via DHCP (default).

MAC address

A randomly generated MAC address is assigned to the PLCSIM Virtual Ethernet Adapter during its installation.

PLCSIM Advanced only uses MAC addresses that are designated as "locally administered" (bit 2 in LSB).

The Siemens-specific prefix is: 02-1B-1B

Three bytes follow, which are determined at random.

Storage location

This MAC address is stored in the registry key "PlcsimvminiMacAddress".

You can overwrite this value.

5.1.4.2 PLCSIM Advanced instances

Detect CPUs and instances

If Ethernet interfaces of CPUs and PLCSIM Advanced instances are mixed in a network, the instances can be recognized by the "PLCSIM" suffix on the station type.

Structure of the MAC address for an instance

The following figure shows the structure of the dynamically generated, locally managed MAC address:

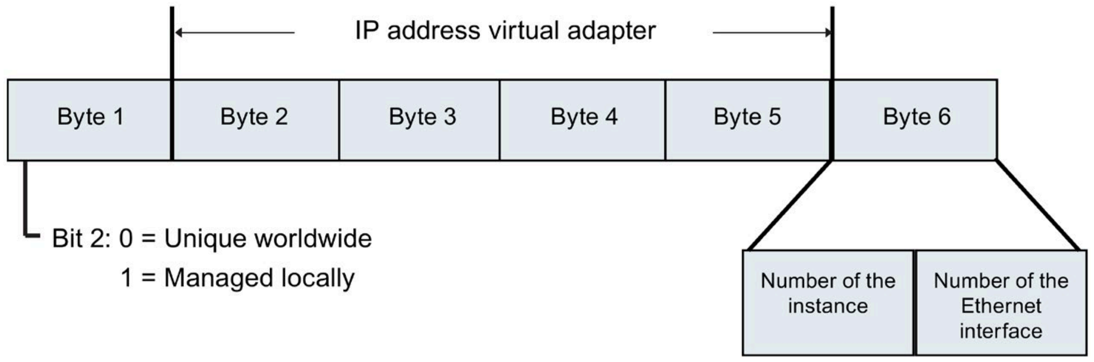

IP address virtual adapter
Byte 1
Byte 2
Byte 3
Byte 4
Byte 5
Byte 6
Bit 2: 0 = Unique worldwide
1 = Managed locally
Number of the instance
Number of the Ethernet interface

Figure 5-8 Structure of the MAC address for an instance

The MAC address tells you the PC on which a PLCSIM Advanced instance has been started.

Assignment of the Ethernet interfaces

Port configurations of the Ethernet interfaces cannot be simulated in PLCSIM Advanced V4.0. Topological interconnection is not supported. A MAC address for a port is reserved internally for each Ethernet interface.

Table 5- 1 Assignment of the Ethernet interfaces, for example, for a CPU 1518-4 PN/DP

Ethernet interface	Last digit of the MAC Address
IE 1 IE 1 / Port 1	...........0 1
IE 2 IE 2 / Port 1	...........2 3
IE 3 IE 3 / Port 1	...........4 5

Example

02-C0-A8-00-83-10 means:

02 → locally managed MAC address of a PLCSIM Advanced instance

C0-A8-00-83 → IP of the Siemens PLCSIM Virtual Ethernet adapter = 192.168.0.131

1 → Instance 1

0 → Ethernet interface IE 1

If no Virtual SIMATIC Memory Card is loaded during startup of PLCSIM Advanced, the interfaces of PLCSIM Advanced display instances with their locally managed MAC address.

5.1.5 Simulate peripheral I/O

The Runtime API writes to and reads from a memory area. This memory is synchronized with the internal process image of the virtual 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 virtual controller must be in RUN to apply changes made by the API.

Note

Dominance of the API when synchronizing

The API dominates when synchronizing. If the user program writes to the same address range as the API, the changes of the API overwrite those of the virtual controller.

See also

Deviating I/O values in the STEP 7 user program (Page 408)

5.1.6 Simulate communication

5.1.6.1 Communication services that can be simulated

PLCSIM Advanced V4.0 supports the following communication options:

Table 5- 2 Supported communication options

Communications options	**Functionality / instructions**
PG communication	On commissioning, testing, diagnostics
Open communication using TCP/IP	• TSEND_C / TRCV_C • TSEND / TRCV • TCON[1] • T_DISCON
Secure Open user communication (secure TCP communication)	• TSEND_C / TRCV_C • TCON
Open communication using ISO-on-TCP	• TSEND_C / TRCV_C • TSEND / TRCV • TCON • T_DISCON
Open communication via UDP[2]	• TUSEND / TURCV • TCON • T_DISCON
Communication via Modbus TCP[3]	• MB_CLIENT • MB_SERVER
E-mail[2, 3]	• TMAIL_C
S7 communication	• PUT / GET • BSEND / BRCV • USEND / URCV
OPC UA server[2, 3]	Secure data exchange with OPC UA clients
Web server[2, 3]	Data exchange via HTTP, HTTPS

[1] When the "PLCSIM" interface (Softbus) is set, communication is performed **internally** via ISO-on-TCP.

[2] Only via the communication interface "PLCSIM Virtual Ethernet Adapter" (TCP/IP).

[3] "Access to PLC via communication module" is not supported.

Special conditions apply when communicating with TUSEND/TURCV, see Restrictions for communications services (Page 402).

Restrictions for MODBUS communication via Softbus

For communication via Softbus, use the supported Modbus versions shown in the following table or, alternatively, communication via TCP/IP.

Table 5- 3 Modbus communication via Softbus

MODBUS TCP Lib	MB_CLIENT	MB_SERVER	SOFTBUS
V6.0	V6.0	V5.3	✓
V5.2	V5.2	V5.2	x
	V5.1	V5.1	x
	V5.0	V5.0	x
V4.2	V4.1	V4.2	x
	V4.1	V4.1	x
	V4.0	V4.0	✓

✓ = Communication possible

x = Communication not possible

TMAIL_C

When the TMAIL_C instruction is used, the mail server might not be located on the same PC as the PLCSIM Advanced instance.

Remedy

Make the mail server available via a different PC in the network.

5.1.6.2 Communication between instances

PLCSIM Advanced supports communication between instances. An instance may be a simulation in PLCSIM Advanced V2.0 or a simulation in WinCC Runtime as of V14.

You can run two instances of PLCSIM Advanced, which then communicate with each other. To enable instances to communicate with each other, they must have a unique IP address.

Each simulated CPU requires a unique IP address

If the CPUs have the same IP address, you cannot run multiple simulations. Each simulated CPU requires a unique IP address.

Make sure that the IP addresses in STEP 7 are unique before you start your simulations.

T-block instructions and UDP

PLCSIM Advanced simulates T-block connections for which the UDP protocol is configured only via the communication interface "PLCSIM Virtual Ethernet Adapter" (TCP/IP).

T-block instructions and data segmentation

PLCSIM Advanced implements T-block instructions with a data segmentation of 4 KB. A real CPU has data segmentation of 8192 bytes.

If you send more than 4 KB in a single TSEND instruction and receive data in ad hoc mode with a TRCV instruction, the TRCV instruction generates new data with only 4 KB. You must perform the TRCV instruction several times to receive additional bytes.

5.1.7 Provide project data offline for simulation

Simulations regardless of STEP 7

To perform simulations independent of STEP 7, you can save the user program and the hardware configuration in STEP 7 in a directory.

Saving retentive data securely

The retentive data is automatically saved when the virtual controllers are shut down.

To save the retentive data safely in the virtual SIMATIC Memory Card, the instances must be correctly logged off. Use one of the following functions for this:

- The `PowerOff()` API function
- In the Control Panel, the function "Shutdown instance", "Log off instance" or the Exit function "Log off all instances"

Provide project data offline

1. Create a "User-defined Card Reader" for your project data in the "Card Reader/USB storage" folder in the project tree of STEP 7 for the CPU.
2. In the "Load preview" dialog for the target device, select "PLC Simulation Advanced" as an action, click in the selection field for this.
   → The project is saved to the <Virtual Memory Card>\SIMATIC.S7S\OMSSTORE directory.
3. Save the folder "\SIMATIC.S7S" with the project data to a medium of your choice.

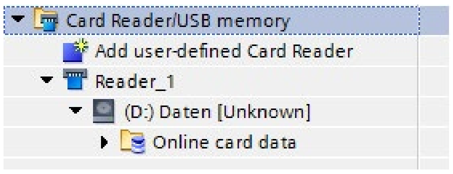
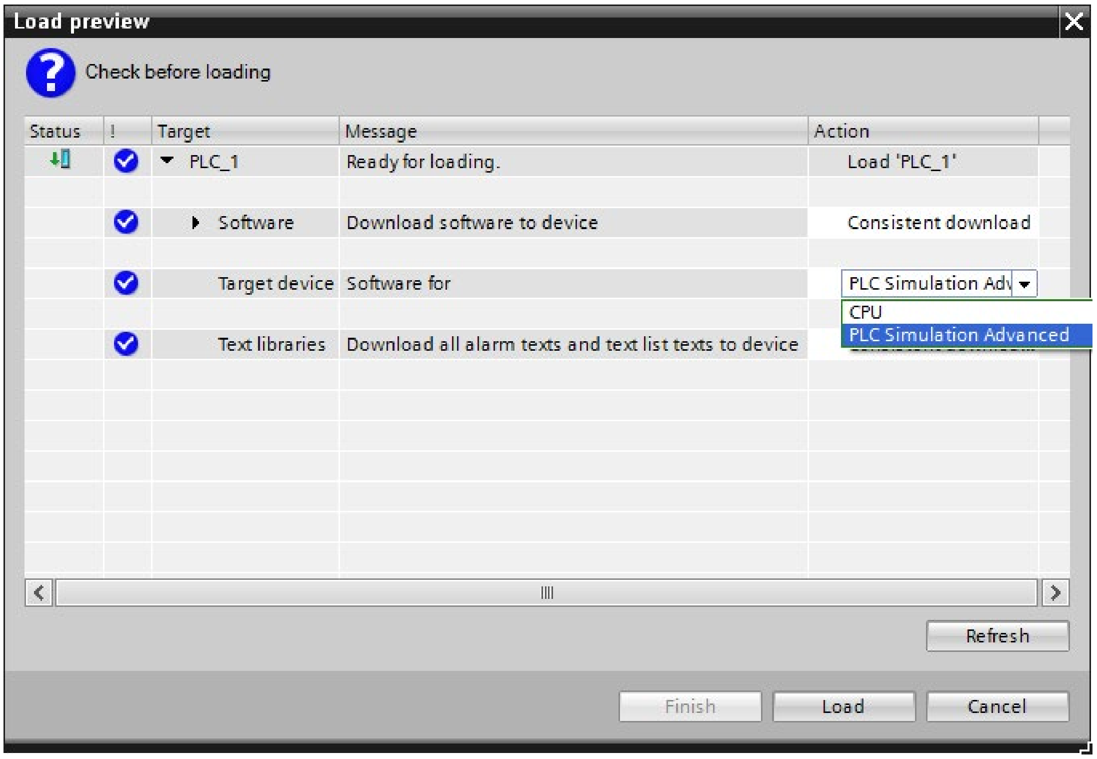


Figure 5-9 Add card reader



Status	!	Target	Message	Action
	✓	PLC_1	Ready for loading.	Load 'PLC_1'
	✓	Software	Download software to device	Consistent download
	✓	Target device	Software for	PLC Simulation Adv
	✓	Text libraries	Download all alarm texts and text list texts to device	

Figure 5-10 Preview of download dialog

Provide project data for simulation

1. On the PC on which PLCSIM Advanced is installed, create the directory "\SIMATIC_MC" in the directory in which the instance saves its data.
2. Move the "\SIMATIC.S7S" folder to the directory you have created.
   → The instances can be started with the project data.

API functions

The project data can be used for an instance via the user interface. Use of the following functions for this:

API functions

- GetStoragePath() (Page 148)
- StoragePath { get; set; } (Page 149)
- ArchiveStorage() (Page 150)
- RetrieveStorage() (Page 152)

See also

Controller - Information and settings (Page 144)

5.2 Simulate CPU with ODK functionality

Introduction

The ODK is an engineering tool that allows the creation of high-level language applications for S7-1500 CPUs. You use it to generate function libraries that are used in the STEP 7 user program.

The ODK for PLCSIM Advanced V4.0 supports the programming language C ++.

You can find the description of the ODK in the Programming and Operating Manual "S7-1500 Open Development Kit 1500S", as of V2.5 Edition 12/2017: SIMATIC STEP 7 (TIA Portal) Options ODK 1500S (https://support.industry.siemens.com/cs/ww/en/ps/13914/man)

Section 6 "Development of a CPU function library for the real-time environment" is relevant for ODK applications under PLCSIM Advanced.

Supported CPUs

PLCSIM Advanced V4.0 supports the ODK functionality of the following controllers:

- CPU 1518(F)-4 PN/DP ODK
- CPU 1518(F)-4 PN/DP MFP

5.2.1 Special features of ODK

Simulating CPU with ODK functionality with PLCSIM Advanced

The simulation of a CPU with ODK functionality requires a special start procedure.

You have the following options:

- Start the instances of a Virtual SIMATIC memory card that contains the project data for the CPU with ODK functionality.
- Before starting the instances, select the CPU type via the API, for example, "CPU1518MFP".
- After the first download, select the functions "Switch off instance" and "Shutdown instance" in the Control Panel.

Note

When you perform the first download to a CPU of the "S7-1500" family, for example, via the PLCSIM Advanced Control Panel, no ODK1500S directory is created on the virtual SIMATIC Memory Card. The CPU cannot be switched to RUN. In this case, you will find messages about missing ODK blocks (e.g. SFC 2013) in the diagnostics buffer.

Supported function libraries

PLCSIM Advanced V4.0 supports the following function libraries for the real-time environment:

- CPU function library: Original Shared Object, SO file as for the hardware CPUs
- PLCSIM Advanced function library (Windows Sync):
  - a 32-bit Windows DLL for ODK Runtime
  - a 64-bit Windows DLL for ODK Runtime

Note

Do not mix function libraries

When simulating with PLCSIM Advanced, only function libraries with the same binary format can be loaded at a time.

If you want to use function libraries with a different binary format, all others must be unloaded first.

Note

Limitations for the execution of CPU function libraries (Windows Sync) with an infinite loop in the class constructor

When the CPU function library (DLL file) contains an object of a class in whose constructor an infinite loop is programmed, the corresponding "ODK client" process gets permanently stuck in this loop when instantiating the object.

Even after reaching the timeout, the infinite loop cannot be interrupted automatically. The PLCSIM Advanced instance remains in RUN even though the entry "Time error - CPU changes to STOP mode" is displayed in the diagnostics buffer.

Note

Limitations for traces in the execution of CPU function libraries (Windows Sync)

Avoid using traces when developing a CPU function library (DLL file) in the class constructor (call of the "ODK_TRACE()" function) to prevent trace messages with faulty parameter values.

Note

No know-how protection for SO files

The SO files for ODK are not know-how-protected.

Debugging a PLCSIM Advanced function library (DLL file)

To debug a function library, attach the Visual Studio Debugger to the corresponding ODK client process which has loaded the respective function library.

PLCSIM Advanced V4.0 supports Visual Studio 2017 and 2019.

Simulation of the ODK with PLCSIM Advanced

If you have loaded the TIA project on the PLCSIM Advanced and the instruction "<*STEP7Prefix*>_Load" was called for the first time, each PLCSIM Advanced instance starts another Windows process ("ODK client") in which the ODK application is executed synchronously with the STEP 7 user program.

Which ODK client is started depends on the function library to be loaded:

- "Siemens.Simatic.PlcSim.Vplc1500.ODKClient.**so**.exe" for an original Shared Object
- "Siemens.Simatic.PlcSim.Vplc1500.ODKClient.**x86**.exe" for a 32-bit application
- "Siemens.Simatic.PlcSim.Vplc1500.ODKClient.**x64**.exe" for a 64-bit application

The executable files of these processes are in the same directory as those of the PLCSIM Advanced Instances ("Siemens.SIMATIC.Simulation.Runtime.Instance.exe").

Note

PLCSIM Advanced does not support asynchronous ODK functions.

Error codes

The same error codes as described in the Programming and Operating Manual "S7-1500 Open Development Kit 1500S" apply to the instructions in the real-time environment. Error codes are also available for PLCSIM Advanced, because the ODK client processes can be closed unexpectedly and therefore an error handling is required.

Restrictions for stack processing

Note

Limitations for stack processing in the version of CPU function libraries for real-time-environment

PLCSIM Advanced ignores the stack size for a CPU function library that is adjusted via the parameter <SyncCallStackSize>. PLCSIM Advanced always provides the maximum stack size of 1 MB.

See Programming and Operating Manual "S7-1500 Open Development Kit 1500S" V2.5, section 5.1.4 Defining the runtime properties of a CPU function library.

PLCSIM Advanced cannot catch any Exceptions of the type "Stack Overflow" while CPU function libraries for the real-time environment (SO files) are being executed.

When developing a CPU function library (SO file), make sure that the maximum stack size of 1 MB is not exceeded. An overflow of the stack leads to an undefined behavior and can lead to the termination of the ODK client process.

Note

Limitations for heap processing in the version of CPU function libraries (Windows Sync)

If a heap corruption occurs when executing a C/C++ function from a CPU function library (DLL file), then this program error is first ignored and execution of the function continues. Only after fully processing the function is the corresponding error code returned (0x8090).

When developing a CPU function library (DLL file), make sure to avoid heap corruption. This way you ensure that after fully processing a C/C++ function no error code is returned.

5.2.2 Loading functions

Loading functions - Instruction "<*STEP7Prefix*>_Load"

If you have loaded the TIA project on the PLCSIM Advanced and the instruction "<*STEP7Prefix*>_Load" was called for the first time, each PLCSIM Advanced instance starts another Windows process. The ODK client then attempts to load the function library which is specified in the SCL file. This is in the directory "<storage path of the instance> \SIMATIC_MC\ODK1500S". See `GetStoragePath()`, `SetStoragePath()` in the section Controller - Information and settings (Page 148).

The ODK client process continues until the instruction "<*STEP7Prefix*>_Unload" is called to unload the last loaded function library or until the process of the PLCSIM Advanced instance ends.

The function call is synchronous and returns after completion of the operation. The output parameter provides information on the progress status.

ODK error code for PLCSIM Advanced

The following table lists the error codes that apply in addition to the error codes that apply to the CPU specifically for ODK applications with PLCSIM Advanced:

Table 5- 4 ODK: Output parameter - Load functions

DONE	BUSY	ERROR	STATUS	Description
0	0	1	0x80A4 = -32604	• The ODK client process cannot be started. • A connection to the ODK client cannot be established or has been interrupted.
0	0	1	0x8095 = -32619	• The ODK client process that is currently running expects a function library with a different binary format.

5.2.3 Calling functions

Call functions - Instruction "*<STEP7Prefix>*SampleFunction"

When calling ODK functions, data is exchanged between the virtual controller and the function library.

The execution of a single function can be interrupted by the execution of higher prioritized OBs.

Technically, the execution of a function is an asynchronous instruction because it is executed in another process. However, the processes are synchronized via the virtual controller. This means that the function call does not return before either the function returns or the ODK client process is closed during the execution.

ODK error code for PLCSIM Advanced

The following table lists the error codes that apply in addition to the error codes that apply to the CPU specifically for ODK applications with PLCSIM Advanced:

Table 5- 5 ODK: Output parameter - Call functions

DONE	BUSY	ERROR	STATUS	Description
0	0	1	0x80A4 = -32604	• The connection to the ODK client was interrupted.

5.2.4 Unloading functions

Unload functions - Instruction "*<STEP7Prefix>*_Unload"

The CPU function library is unloaded by calling the instruction "*<STEP7Prefix>*_Unload". If no other function library is loaded or if the process of the PLCSIM Advanced instance is closed, then the ODK client process is shut down.

The function call is asynchronous, the call returns immediately. The output parameter informs about the progress status.

5.3 Simulating Motion Control

Restrictions

PLCSIM Advanced simulates the real CPU, but not configured, connected technology modules or other I/O devices.

It is possible to download a STEP 7 project with technology modules for operation of motion control. However, the built-in logic of the technology modules is not part of the simulation. Therefore, the corresponding motion control instructions are not supported.

In contrast to a real CPU, S7-PLCSIM Advanced does not support isochronous mode for centralized I/O in S7-1500 with local send clock.

OB 91 and OB 92

If you convert a Motion Control project which contains the OB 91 and OB 92 from STEP 7 V13, then you cannot load this project to a PLCSIM Advanced.

Solution

Delete OB 91 and OB 92 in the project and recompile the project.

The OBs are thus created again with the simulation support required for PLCSIM Advanced.
Compilation resets the properties of the blocks to the default values.
Restore the required settings in the properties.

Why is the "Overflow" message for the OB MC-Servo [OB91] displayed in the diagnostic buffer of the virtual controller S7-PLCSIM Advanced?

S7-PLCSIM Advanced provides you with the virtual controller of the S7-1500 hardware CPU. The virtual controller allows you to run the firmware of the S7-1500 under the Windows operating system.

The wide-ranging communication options and the functions of the API offer you integration into existing simulation landscapes or co-simulation with other tools. The virtual controller runs in the Windows environment with the following restrictions. The real hardware CPUs, on the other hand, are designed for the maximum possible performance without the compromises required by a general PC operating system.

If there are overflows of the OB MC-Servo [OB91] in the diagnostic buffer, the time for the application cycle (ms) has been exceeded, because the calculation of this application cycle could not be completed within the required time.

Solution

The overflows of OB MC-Servo [OB91] in the diagnostic buffer decrease:

- When fewer additional Windows processes are executed

  and

- When the computing power of the CPU is higher

Overflow detection is activated for S7-PLCSIM Advanced as of V3.0 for exact simulation of the technology objects. If diagnostic buffer overflows occur on your PC for OB MC-Servo [OB91] and your instance goes into the STOP mode, the following remedies are available to you:

1. Use the virtual time of S7-PLCSIM Advanced and start with the lowest possible scaling factor for the virtual clock. Increase the value step-by-step until the first overflows occur in the diagnostic buffer. Repeat this procedure until you have determined the maximum scaling factor for which you do not yet get any overflows in the diagnostic buffer.

   Information on the scaling factor can be found in section Speed up and slow down simulation (Page 77).

2. Set a longer application cycle (ms) for the OB MC-Servo [OB91] in STEP 7.

Simulation with external simulation software

Note

In a virtual S7-1500 controller, the technology objects are connected to the process image. Simulation software can thus access the process image via the user interfaces (API) of PLCSIM Advanced and simulate the behavior of the other connected axes.

Simulation mode in STEP 7

The simulation mode in STEP 7 is a standard function of the technology objects and is independent of PLCSIM Advanced.

If you want to move an axis in simulation mode, select the "Activate simulation" check box in STEP 7 under "Technology Object > Configuration> Basic Parameters > Simulation". No additional setting is required for a virtual axis.

Feedback of the axis position

The speed setpoint of the simulated drive is integrated into the actual position value with a time delay (PT1). The result of this calculation is returned to the technology object as position actual value of the axis.

Reference point approach of the axis

If you selected "Use zero mark via PROFIdrive frame" in STEP 7 for the reference point approach, PLCSIM Advanced responds immediately to any active (mode 2, 3, 8) or passive (mode 4, 5) reference point approach command (MC_Home). The actual position is predefined as the reference point.

Additional information

You can find information on the technology functions of the CPU in the S7-1500/S7-1500T Motion Control (https://support.industry.siemens.com/cs/ww/en/view/109751049) function manuals.

For more information, refer to the manuals of the supported SIMATIC controllers. (https://support.industry.siemens.com/cs/ww/en/view/109744173)

5.4 Simulating the SIMATIC Drive Controller

Introduction

The SIMATIC Drive Controller is a drive-based controller in the SIMATIC S7-1500 range.

A SIMATIC Drive Controller combines the following functionalities in a SINAMICS S120 Booksize Compact housing:

- Fail-safe SIMATIC S7-1500 technology CPU with integrated technology I/Os
- SINAMICS S120 drive control

Both components are called "CPU" and "SINAMICS Integrated" in the documentation.

The SIMATIC Drive Controller supports PROFINET and PROFIBUS DP communication.

Supported SIMATIC Drive Controllers

S7-PLCSIM Advanced V4.0 supports the SIMATIC Drive Controllers as of firmware version V2.9:

- CPU 1504D TF (6ES7615-4DF10-0AB0)
- CPU 1507D TF (6ES7615-7DF10-0AB0)

Special features

Unlike other SIMATIC S7-1500 technology CPUs, the SIMATIC Drive Controllers also have:

- Integrated inputs/outputs (onboard I/O)
- Integrated drive control SINAMICS Integrated

Restrictions

S7-PLCSIM Advanced only simulates the standard CPU functionality of the SIMATIC Drive Controller.

Not simulated are:

- the technology functions of the onboard I/O
- the SINAMICS Integrated
- PROFINET IO
- PROFIBUS DP

The integrated inputs/outputs of the X122, X132 and X142 interfaces can only be simulated as binary inputs/outputs.

Technological functions are not simulated, for example, Timer DI/DQ, Oversampling DI/DQ. Channel parameter assignments, such as signal inversion, input delay and edge detection are not possible.

The functionality of the SINAMICS Integrated is not simulated – but the SINAMICS Integrated is shown as a valid node.

Simulations are possible as with SINAMICS S120 CU320-2 based on the drive telegrams (e.g. by reading and forcing the telegram addresses).

Coupled isochronous mode

In coupled isochronous mode, the relevant clock systems use a shared system clock, for example, from PROFINET IO or the local send clock of the technology I/Os. The leading clock system provides its own system clock to the other clock systems.

Note

Leading clock system

Clock synchronization with technology I/Os X142 (local send clock) as leading clock system is not possible with S7-PLCSIM Advanced. In this case, you configure the PROFINET IO interface X150 as leading clock system.

The information provided in the section Simulation of Motion Control (Page 69) still applies.

Additional information

The SIMATIC Drive Controller system manual (https://support.industry.siemens.com/cs/ww/en/view/109766665) describes in detail the configuration, installation, wiring and commissioning of the SIMATIC Drive Controller. The STEP 7 online help supports you in the configuration and programming.

The SIMATIC Drive Controller equipment manual (https://support.industry.siemens.com/cs/ww/en/view/109766666) contains a compact description of the module-specific information, such as properties, wiring diagrams, characteristics and technical specifications.

5.5 Simulating a redundant S7-1500R/H system

Introduction

In a redundant S7-1500R/H system, the CPUs are duplicated, in other words, redundant. The two CPUs process the same project data and the same user program in parallel. The two CPUs are synchronized over two redundancy connections. If one CPU fails, the other CPU maintains control of the process.

Supported CPUs

PLCSIM Advanced V4.0 supports the R/H CPUs of the redundant S7-1500R/H system as of firmware version V2.9 with the following functional restrictions:

- CPU 1513R-1 PN (6ES7513-1RL00-0AB0)
- CPU 1515R-2 PN (6ES7515-2RM00-0AB0)
- CPU 1515R-2 PN SIPLUS RAIL (6AG2 515-2RM00-4AB0)
- CPU 1515R-2 PN SIPLUS (6AG1 515-2RM00-7AB0)
- CPU 1517H-3 PN (6ES7517-3HP00-0AB0)
- CPU 1517H-3 PN SIPLUS (6AG1 517-3HP00-4AB0)
- CPU 1518HF-4 PN (6ES7518-4JP00-0AB0)

Supported operating and system states

Like standard S7-1500 CPUs, the S7-1500R/H CPUs have the operating states STOP, STARTUP and RUN. For operation as a redundant system, one of the two CPUs can take on an additional operating state, SYNCUP, for synchronizing the two subsystems. The RUN operating state is divided into the following states for redundant systems:

- RUN
- RUN-Syncup
- RUN-Redundant

The system states of the redundant S7-1500R/H system result from the combination of the operating states of the individual CPUs as follows:

- STOP
- STARTUP
- RUN-Solo
- SYNCUP
- RUN-Redundant

Note

RUN-Solo system state is supported by PLCSIM Advanced

The simulation of a redundant S7-1500R/H system is possible in the RUN-Solo system state (RUN operating state of the CPU). In the RUN operating state, the (leading) primary CPU behaves just like an S7-1500 standard CPU. The MAINT LED on the CPU is always yellow (maintenance request) because no partner CPU was found for redundant operation.

No simulation is possible with PLCSIM Advanced in redundant system operation.

Additional information

The Redundant system S7-1500R/H System Manual (https://support.industry.siemens.com/cs/de/de/view/109754833/en) describes in detail the configuration, installation, wiring and commissioning of the redundant S7-1500R/H system. The STEP 7 online help supports you in the configuration and programming.

The equipment manuals of the R-, H- and HF-CPUs contain a compact description of the module-specific information, such as properties, wiring diagrams, characteristics and technical specifications of the CPUs.

6 Virtual time response

The virtual controller uses internally two types of clocks for 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 points is measured in virtual time.

The virtual time can be accelerated or slowed for test 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 PLCSIM Advanced runs in a Windows environment, Windows might temporarily suspend the virtual controller process. In such a case, both the virtual and the real clock stop in the virtual controller. They only continue to run when Windows resumes processing.

Virtual system time

When you start PLCSIM Advanced, the virtual system time of the virtual controller starts with the system time of Windows.

The virtual system time is based on the virtual clock, i.e. if a scaling factor is used, the system time runs correspondingly faster or slower.

All events that the virtual controller sends to the API provides 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 / winter time

---

API functions

- GetSystemTime() (Page 249)
- SetSystemTime() (Page 250)
- SystemTime { get; set; } (Page 250)

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 virtual controller for simulations.

To set the required scaling factor, enable the grayed out icon on the control panel to the right of your PLCSIM Advanced instance. The symbol then becomes active and you can use the scaling function.

- The default is 1, i.e. the course of the virtual time corresponds to the course of 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%.

API functions

- GetScaleFactor() (Page 250)
- SetScaleFactor() (Page 251)
- ScaleFactor { get; set; } (Page 252)

See also

Settings for the virtual time (Page 249)

6.1 Speed up and slow down simulation

Influence of fast forward and slow motion

Simulations can be accelerated and slowed down. Fast forward and slow motion only affects time-based components, for example, cyclic OBs. Compared to the real time, they are performed more frequently with fast forward and less frequently with slow motion.

Fast forward and slow motion do not change the execution speed of the CPU machine codes. For example, the speed at which all operations of an OB1 cycle are executed does not change. The execution speed depends on the processor of the PC on which the virtual controller running. If you change the scaling factor, more or fewer cycle control points are reached in a given period of virtual time.

Note

Performance

The performance depends on the size of your project, among other things.

If the scaling factor is too high and the cycle-time monitoring indicates that the PC was incapable of calculating the OB1 or cyclic OBs in the specified time, the virtual controller goes to STOP.

Recommendation: To avoid this, start with a small scaling factor and gradually increase it step-by-step while keeping the virtual controller in RUN.

If an overflow of events occurs, slow down the speed of the simulation. See Monitoring overflow (Page 407) and Cycle control (Page 252).

Fast forward

To speed up the virtual time, select a scaling factor greater than 1 in the Control Panel or in the API.

Slow motion

To slow down the virtual time, select a scaling factor less than 1 in the Control Panel or in the API.

API functions

- GetScaleFactor() (Page 250)
- SetScaleFactor() (Page 251)
- ScaleFactor { get; set; } (Page 252)

See also

Settings for the virtual time (Page 249)

6.2 Stop simulation

Freeze state of the virtual controller

To stop a simulation and to synchronize a simulation partner, a virtual controller can be set to a freeze state via the API. When the virtual controller has reached a synchronization point, it sends the event `OnSyncPointReached` to the API clients.

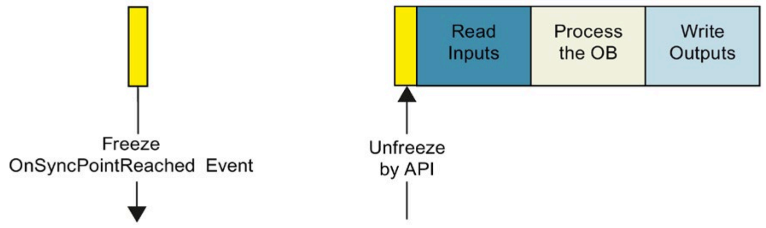


Figure 6-1 Freeze state of the virtual controller

The following occurs in the freeze state:

- The virtual time is stopped.
- No OBs and no timers are running.
- The user program is no longer executed.
- The virtual controller is still accessible from the TIA Portal.
- The input and output data of the virtual controller are in a consistent state.

Note

Freeze state during downloading

To complete a download in freeze state, the virtual controller must pass a cycle control point at the end of the download.

Note

Freeze-state ≠ operating state

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

- The LED display on the Control Panel and on the Web server accordingly shows RUN or STOP for instance.
- The instance shows the operating state `SROS_FREEZE` / `Freeze`, see EOperatingState (Page 380).

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.

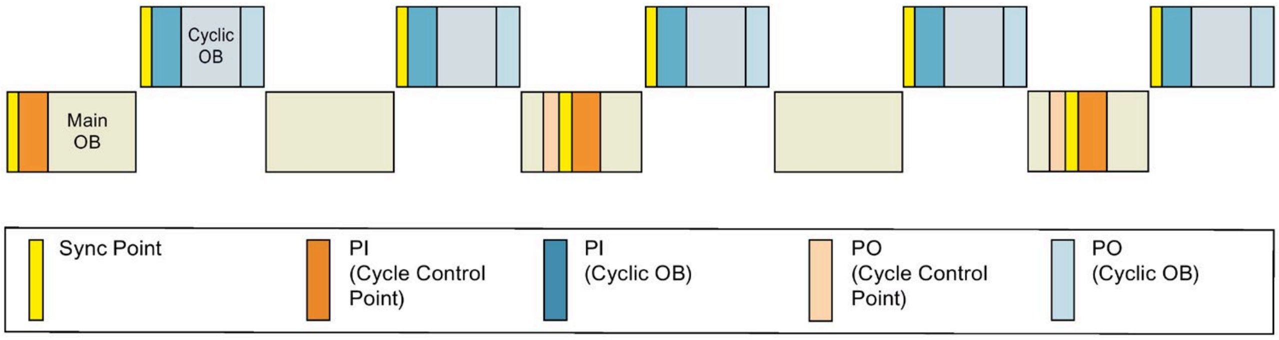


Figure 6-2 Overview of the synchronization points

Trigger freeze state

To trigger the freeze state, following modes are available for the virtual controller:

- SingleStep operating modes

  See Synchronize simulation partner cycle-controlled (Page 80).
- TimespanSynchronized operating modes

  See Synchronize simulation partner time-controlled (Page 82).

In Default operating mode, the virtual controller does not change into a freeze state.

API functions

- Settings for the cycle control (Page 252)
- GetOperatingMode() (Page 252)
- SetOperatingMode() (Page 253)
- OperatingMode { get; set; } (Page 253)
- EOperatingMode (Page 381)

6.3 Synchronize simulation partner

6.3.1 Synchronize simulation partner cycle-controlled

SingleStep operating modes

Several simulation partners (clients) are synchronized cycle-controlled with the SingleStep operating modes of the virtual controller. The operating modes define the synchronization point at which the virtual controller changes to the freeze state and sends the `OnSyncPointReached` event.

Table 6- 1 Cycle-controlled operating modes (SingleStep)

Operating mode	Synchronization point		Minimum cycle time[1]	Send clock "Bus"[2]
	Cycle control point	Before reading in the process image partition		
	"C"	"P"	"T"	
`SingleStep_C`	✓			
`SingleStep_P`		✓		
`SingleStep_CP`	✓	✓		
`SingleStep_CT`	✓		✓	
`SingleStep_CPT`	✓	✓	✓	
`SingleStep_Bus`				✓

1 In addition, the minimum cycle time of the OB 1 is overwritten in this operating mode. When you define a minimum cycle time of 200 ms via the API, the minimum distance between two cycle control points is 200 virtual milliseconds. The default setting is 100 ms.

2 Send clock that you can set in STEP 7 in the properties of the PROFINET interface of the CPU (Advanced options > Realtime settings > IO communication > Send clock)

API functions / events

- GetOverwrittenMinimalCycleTime_ns() (Page 254)
- SetOverwrittenMinimalCycleTime_ns() (Page 255)
- OverwrittenMinimalCycleTime_ns { get; set; } (Page 255)
- RunToNextSyncPoint() (Page 256)
- OnSyncPointReached (Page 292)
- EventCallback_II_SREC_ST_UINT32_INT64_INT64_UINT32 (Page 325) / Delegate_II_EREC_DT_UINT32_INT64_INT64_UINT32 (Page 340)

Terminating the freeze state

The `RunToNextSyncPoint()` function cancels the freeze state and induces the virtual controller 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 virtual controller 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.

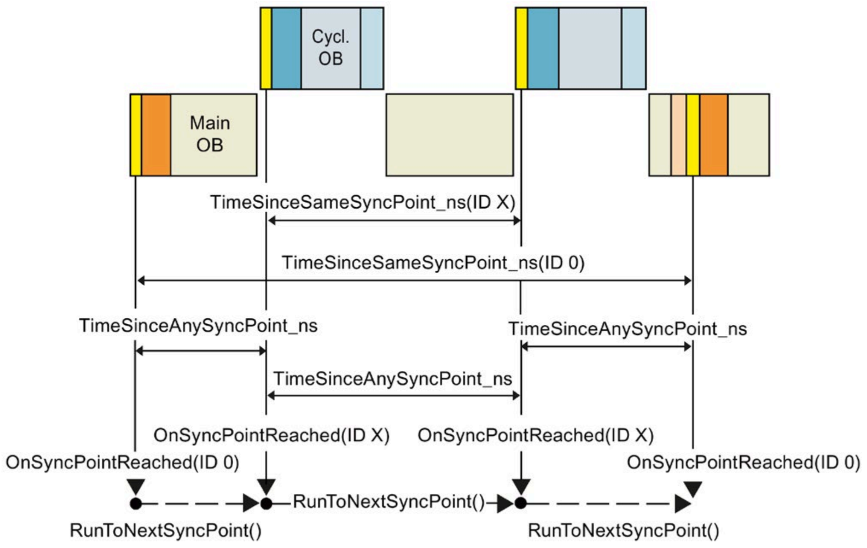


Figure 6-3 Example: Sequence in the SingleStep_CP operating mode

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.

In order to display the tag values **after** the processing, select the extended mode for the watch table and then select "Permanently, at end of scan cycle" in the "Monitor with trigger" column.

See also

Cycle control (Page 252)

6.3.2 Synchronize simulation partner time-controlled

TimespanSynchronized operating modes

Several simulation partners (clients) are synchronized time-controlled with the TimespanSynchronized operating modes of the virtual controller. The operating modes define the synchronization point at which the virtual controller changes to the freeze state and sends the `OnSyncPointReached` event.

Table 6- 2 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 252)
- StartProcessing() (Page 257)
- OnSyncPointReached (Page 292)

Terminating the freeze state

The `StartProcessing(t)` function cancels the freeze state and induces the virtual controller 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 virtual controller also sends the runtime since the last call of the `StartProcessing`(t) (`TimeSinceSameSyncPoint_ns` / `TimeSinceAnySyncPoint_ns`) function.

The `StartProcessing()` function cancels the freeze state.

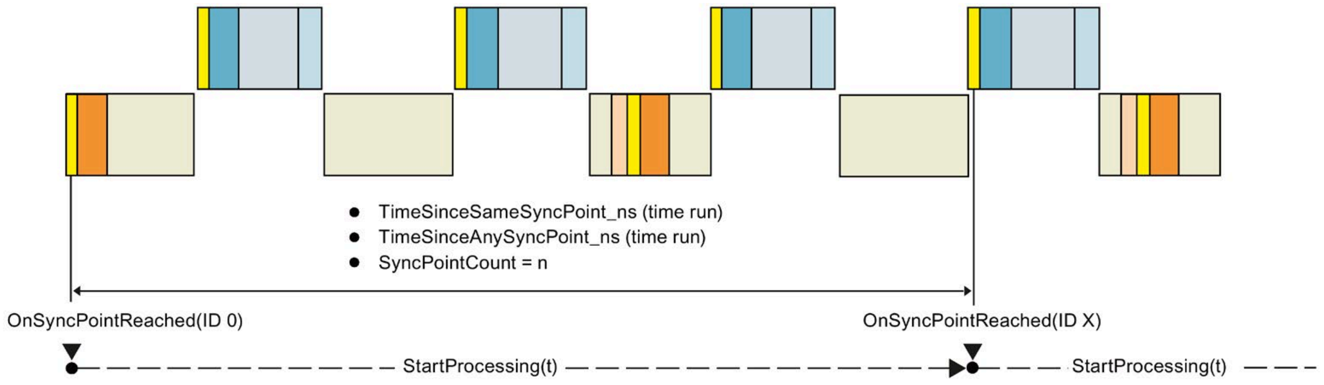


Figure 6-4 Example: Sequence in the TimespanSynchronized_CP operating mode

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 virtual controller 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 he 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 virtual controller. The API client does not receive events from other API clients. It cannot send messages to other API clients.

User interfaces (API) 7

7.1 Introduction

Components of the Simulation Runtime

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

Table 7- 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 virtual controller. Each virtual controller 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.
• "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 virtual controllers.

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

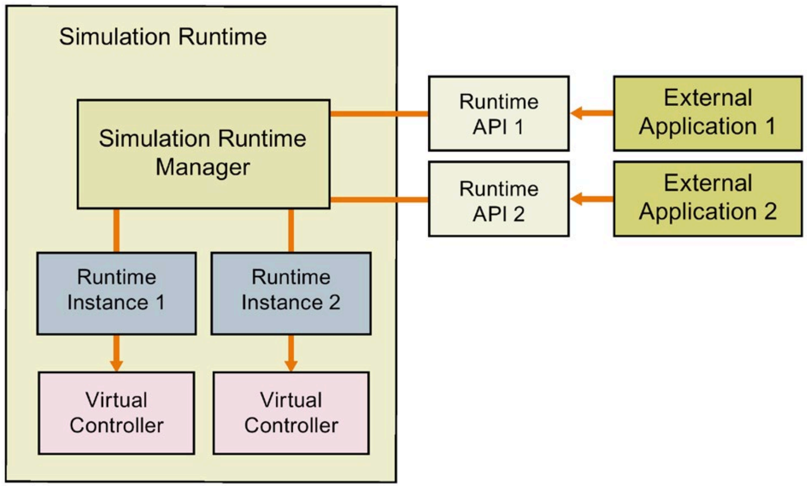
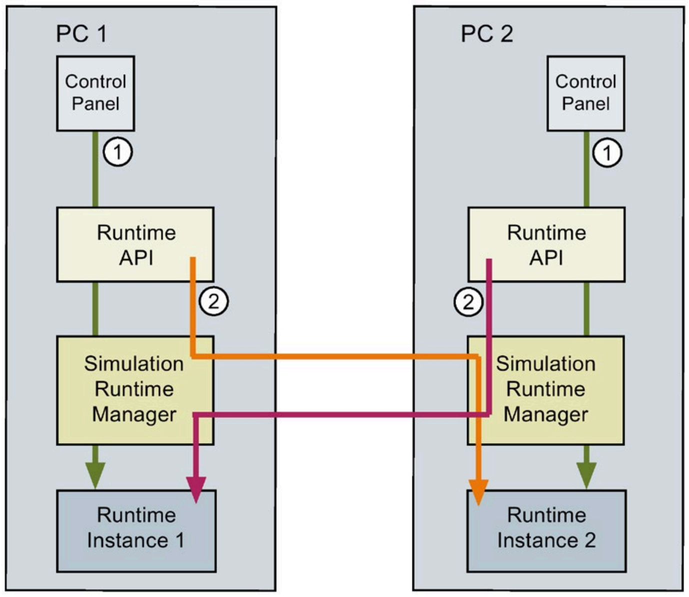


Figure 7-1 External applications and Simulation Runtime

7.1.1 Access to instances

Access via the Control Panel and the API

You can access only one instance that is available locally 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 7-2 Access to instances with distributed communication

API functions

- Table 7-6 Overview of IInstances functions - Native C++ (Page 90)
- Table 7-13 Overview of IInstances functions - .NET (C#) (Page 94)
- Table 7-8 Overview of IRemoteRuntimeManager functions - Native C++ (Page 92)
- Table 7-15 Overview of IRemoteRuntimeManager functions - .NET (C#) (Page 95)

See also

Overview of user interfaces for managed code (Page 93)

7.1.2 User interfaces (API)

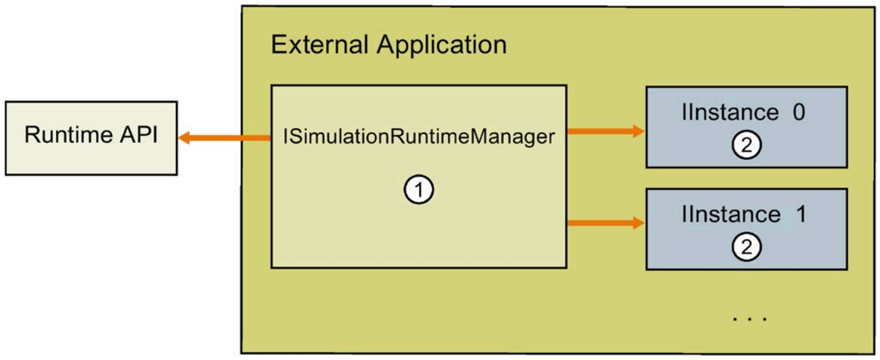
The user interfaces of Simulation Runtime include functions you use, for example, to create instances, to change the operating state of a virtual controller, or to exchange I/O data.

Simulation Runtime has the following user interfaces:

- ISimulationRuntimeManager
- IInstances
- IRemoteRuntimeManager

API and external applications

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



① **ISimulationRuntimeManager**
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 **IInstances**
Interface of a Runtime instance. It is used to change the operating state of a virtual controller and to exchange I/O data. Each instance has a unique name and an ID.

Figure 7-3 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 user interfaces for native C++ (Page 88)
- Overview of data types for native C++ (Page 96)
- Overview of user interfaces for managed code (Page 93)
- Overview of data types for managed code (Page 98)

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. Zero pointers are not permitted. Exceptions to this are the functions that return an interface of a virtual controller:

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

7.1.3 Overview of user interfaces for native C++

Initializing and shutting down API

Table 7- 2 Overview of initializing and shutting down API - Native C++

Actions	Functions
Initialize API	InitializeApi (Page 100) RuntimeApiEntry_Initialize (Page 102)

Shut down API (Page 104)	`DestroyInterface`
	`RuntimeApiEntry DestroyInterface`

Logging off API library (Page 107)	`FreeApi` `ShutdownAndFreeApi`

Global functions

Table 7- 3 Overview of global functions - Native C++

Actions	Functions

Global functions (Page 109)	`GetNameOfAreaSection()`
	`GetNameOfCPUType()`
	`GetNameOfCommunicationInterface()`
	`GetNameOfDataType()`
	`GetNameOfLEDMode()`
	`GetNameOfLEDType()`
	`GetNameOfOperatingMode()`
	`GetNameOfOperatingState()`
	`GetNameOfPrimitiveDataType()`
	`GetNameOfTagListDetails()`
	`GetNameOfErrorCode()`
	`GetNameOfRuntimeConfigChanged()`
	`GetNameOfInstanceConfigChanged()`
	`GetNameOfDiagSeverity()`
	`GetNameOfDirection()`
	`GetNameOfRackOrStationFaultType()`
	`GetNameOfProcessEventType()`
	`GetNameOfPullOrPlugEventType()`
	`GetNameOfCycleTimeMonitoringMode()`
	`GetNameOfDiagProperty()`
	`GetNameOfAutodiscoverType()`

API ISimulationRuntimeManager

Table 7- 4 Overview of API ISimulationRuntimeManager functions - Native C++

Settings	Functions

Interface (Page 114)	`GetVersion()`
	`IsInitialized()`
	`IsRuntimeManagerAvailable()`
	`Shutdown()`
	`GetStrictMotionTiming()`
	`SetStrictMotionTiming()`

Simulation Runtime instances (Page 118)	`GetRegisteredInstancesCount()`
	`GetRegisteredInstanceInfoAt()`
	`RegisterInstance()`
	`RegisterCustomInstance()`
	`CreateInterface()`

Remote connections (Page 124)	`OpenPort()`
	`ClosePort()`
	`GetPort()`
	`GetRemoteConnectionsCount()`
	`GetRemoteConnectionInfoAt()`
	`RemoteConnect()`
	`RunAutodiscover()`

Table 7- 5 Overview of API ISimulationRuntimeManager events - Native C++

Events	Functions

OnConfigurationChanged (Page 132)	`RegisterOnConfigurationChangedCallback()`
	`UnregisterOnConfigurationChangedCallback()`
	`RegisterOnConfigurationChangedEvent()`
	`UnregisterOnConfigurationChangedEvent()`
	`WaitForOnConfigurationChangedEvent()`

OnRuntimeManagerLost (Page 134)	`RegisterOnRuntimeManagerLostCallback()`
	`UnregisterOnRuntimeManagerLostCallback()`
	`RegisterOnRuntimeManagerLostEvent()`
	`UnregisterOnRuntimeManagerLostEvent()`
	`WaitForOnRuntimeManagerLostEvent()`

OnAutodiscover (Page 137)	`RegisterOnAutodiscoverCallback()`
	`UnregisterOnAutodiscoverCallback()`

API IInstances

Table 7- 6 Overview of IInstances functions - Native C++

Settings	Functions

Interface (Page 138)	`GetID()`
	`GetName()`
	`GetCPUType()`
	`SetCPUType()`
	`GetCommunicationInterface()`
	`SetCommunicationInterface()`
	`GetInfo()`
	`UnregisterInstance()`
	`GetStrictMotionTiming()`
	`SetStrictMotionTiming()`

Controller (Page 144)	`GetControllerName()`
	`GetControllerShortDesignation()`
	`GetControllerIPCount()`
	`GetControllerIP()`
	`GetControllerIPSuite4()`
	`SetIPSuite()`
	`GetStoragePath()`
	`SetStoragePath()`
	`ArchiveStorage()`
	`RetrieveStorage()`
	`CleanupStoragePath()`

Operating state (Page 156)	`PowerOn()`
	`PowerOff()`
	`Run()`
	`Stop()`
	`GetOperatingState()`
	`MemoryReset()`

Tag list (Page 167)	`UpdateTagList()`
	`GetTagListStatus()`
	`GetTagInfoCount()`
	`GetTagInfos()`
	`CreateConfigurationFile()`

I/O access via address - Reading (Page 174)	`GetAreaSize()`
	`ReadBit()`
	`ReadByte()`
	`ReadBytes()`
	`ReadSignals()`

Settings	Functions

I/O access via address - Writing (Page 183)	`WriteBit()`
	`WriteByte()`
	`WriteBytes()`
	`WriteSignals()`

I/O access via tag name - Reading (Page 189)	`Read()`
	`ReadBool()`
	`ReadChar(), ReadWChar()`
	`ReadDouble()`
	`ReadFloat()`
	`ReadInt8(), ReadInt16(), ReadInt32(), ReadInt64()`
	`ReadUInt8(), ReadUInt16(), ReadUInt32(), ReadUInt64()`
	`ReadSignals()`

I/O access via tag name - Writing (Page 220)	`Write()`
	`WriteBool()`
	`WriteChar(), WriteWChar()`
	`WriteDouble()`
	`WriteFloat()`
	`WriteInt8(), WriteInt16(), WriteInt32(), WriteInt64(),`
	`WriteUInt8(), WriteUInt16(), WriteUInt32(), WriteUInt64()`
	`WriteSignals()`

Virtual time (Page 249)	`GetSystemTime()`
	`SetSystemTime()`
	`GetScaleFactor()`
	`SetScaleFactor()`

Cycle control (Page 252)	`GetOperatingMode()`
	`SetOperatingMode()`
	`SetSendSyncEventInDefaultModeEnabled()`
	`IsSendSyncEventInDefaultModeEnabled`
	`GetOverwrittenMinimalCycleTime_ns()`
	`SetOverwrittenMinimalCycleTime_ns()`
	`RunToNextSyncPoint()`
	`StartProcessing()`
	`SetCycleTimeMonitoringMode()`
	`GetCycleTimeMonitoringMode()`

Acyclic services (Page 261)	

Table 7- 7 Overview of IInstances events - Native C++

Events	Functions
OnOperatingStateChanged (Page 281)	`RegisterOnOperatingStateChangedCallback()` `UnregisterOnOperatingStateChangedCallback()` `RegisterOnOperatingStateChangedEvent()` `UnregisterOnOperatingStateChangedEvent()` `WaitForOnOperatingStateChangedEvent()`
OnLedChanged (Page 285)	`RegisterOnLedChangedCallback()` `UnregisterOnLedChangedCallback()` `RegisterOnLedChangedEvent()` `UnregisterOnLedChangedEvent()` `WaitForOnLedChangedEvent()`
OnConfigurationChanging (Page 287)	`RegisterOnConfigurationChangingCallback()` `UnregisterOnConfigurationChangingCallback()` `RegisterOnConfigurationChangingEvent()` `UnregisterOnConfigurationChangingEvent()` `WaitForOnConfigurationChangingEvent()`
OnConfigurationChanged (Page 290)	`RegisterOnConfigurationChangedCallback()` `UnregisterOnConfigurationChangedCallback()` `RegisterOnConfigurationChangedEvent()` `UnregisterOnConfigurationChangedEvent()` `WaitForOnConfigurationChangedEvent()`
OnSyncPointReached (Page 292)	`RegisterOnSyncPoin-tReachedCallback()UnregisterOnSyncPointReachedCallback()RegisterOnSyncPointReachedEvent()UnregisterOnSyncPoin-tReachedEvent()WaitForOnSyncPointReachedEvent()`

API IRemoteRuntimeManager

Table 7- 8 Overview of IRemoteRuntimeManager functions - Native C++

Settings	Functions
Interface (Page 304)	`GetVersion()` `GetIP()` `GetPort()` `GetRemoteComputerName()` `Disconnect()` `GetStrictMotionTiming()` `SetStrictMotionTiming()`
Simulation Runtime instances (Page 309)	`GetRegisteredInstancesCount()` `GetRegisteredInstanceInfoAt()` `RegisterInstance()` `RegisterCustomInstance()` `CreateInterface()`

Table 7- 9 Overview of IRemoteRuntimeManager events - Native C++

Events	Functions
OnConnectionLost (Page 317)	`RegisterOnConnectionLostCallback()` `UnregisterOnConnectionLostCallback()` `RegisterOnConnectionLostEvent()` `UnregisterOnConnectionLostEvent()` `WaitForOnConnectionLostEvent()`

7.1.4 Overview of user interfaces for managed code

Initializing and shutting down API

Table 7- 10 Overview of initializing and shutting down API - .NET (C#)

Actions	Functions
Initialize API (Page 104)	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeManager`
Shut down API (Page 109)	

API ISimulationRuntimeManager

Table 7- 11 Overview of ISimulationRuntimeManager functions - .NET (C#)

Settings	Functions

Interface (Page 114)	`Version { get; }`
	`IsInitialized { get; }`
	`IsRuntimeManagerAvailable { get; }`
	`Shutdown()`
	`StrictMotionTiming { get; set; }`

Simulation Runtime instances (Page 118)	`RegisterInstanceInfo { get; }`
	`RegisterInstance()`
	`RegisterCustomInstance()`
	`CreateInterface()`

Remote connections (Page 124)	`OpenPort()`
	`ClosePort()`
	`Port { get; }`
	`RemoteConnectionInfo { get; }`
	`RemoteConnect()`
	`RunAutodiscover()`

Table 7- 12 Overview of ISimulationRuntimeManager events - .NET (C#)

Events	Functions

OnConfigurationChanged (Page 132)	`OnConfigurationChanged`
	`RegisterOnConfigurationChangedEvent()`
	`UnregisterOnConfigurationChangedEvent()`
	`WaitForOnConfigurationChangedEvent()`

OnRuntimeManagerLost (Page 134)	`OnRuntimeManagerLost()`
	`RegisterOnRuntimeManagerLostEvent()`
	`UnregisterOnRuntimeManagerLostEvent()`
	`WaitForOnRuntimeManagerLostEvent()`

OnAutodiscover (Page 137)	`OnAutodiscoverData`

API IInstances

Table 7- 13 Overview of IInstances functions - .NET (C#)

Settings	Functions

Interface (Page 138)	`Dispose ()`
	`ID { get; }`
	`Name { get; }`
	`CPUType { get; set; }`
	`CommunicationInterface { get; }`
	`Info { get; }`
	`UnregisterInstance()`
	`StrictMotionTiming { get; set; }`

Controller - Information and settings (Page 144)	`ControllerName { get; }`
	`ControllerShortDesignation { get; }`
	`ControllerIPSuite4 { get; }`
	`SetIPSuite()`
	`StoragePath { get; set; }`
	`ArchiveStorage()`
	`RetrieveStorage()`
	`CleanupStoragePath()`

Operating state (Page 156)	`PowerOn()`
	`PowerOff()`
	`Run()`
	`Stop()`
	`OperatingState { get; }`
	`MemoryReset()`

Tag list (Page 167)	`UpdateTagList()`
	`GetTagListStatus()`
	`TagInfos { get; }`
	`CreateConfigurationFile()`

I/O access via address - Reading (Page 174)	`InputArea | MarkerArea | OutputArea { get; }`
	`AreaSize { get; }`
	`ReadBit()`
	`ReadByte()`
	`ReadBytes()`
	`ReadSignals()`

I/O access via address - Writing (Page 183)	`WriteBit()`
	`WriteByte()`
	`WriteBytes()`
	`WriteSignals()`

I/O access via tag name - Reading (Page 189)	`Read()`
	`ReadBool()`
	`ReadChar(), ReadWChar()`
	`ReadDouble()`
	`ReadFloat()`
	`ReadInt8(), ReadInt16(), ReadInt32(), ReadInt64()`
	`ReadUInt8(), ReadUInt16(), ReadUInt32(), Rea-dUInt64()`
	`ReadSignals()`

I/O access via tag name - Writing (Page 220)	`Write()`
	`WriteBool()`
	`WriteChar(), WriteWChar()`
	`WriteDouble()`
	`WriteFloat()`
	`WriteInt8(), WriteInt16(), WriteInt32(), WriteInt64(),`
	`WriteUInt8(), WriteUInt16(), WriteUInt32(), WriteUInt64()`
	`WriteSignals()`

Settings	Functions

Virtual time (Page 249)	`SystemTime { get; set; }`
	`ScaleFactor { get; set; }`

Cycle control (Page 252)	`OperatingMode { get; set; }`
	`IsSendSyncEventInDefaultModeEnabled { get; set; }`
	`OverwrittenMinimalCycleTime_ns { get; set; }`
	`RunToNextSyncPoint`
	`StartProcessing()`
	`SetCycleTimeMonitoringMode()`
	`GetCycleTimeMonitoringMode()`

Acyclic services (Page 261)	

Table 7- 14 Overview of IInstances events - .NET (C#)

Events	Functions

OnOperatingStateChanged (Page 281)	`OnOperatingStateChanged`
	`RegisterOnOperatingStateChangedEvent()`
	`UnregisterOnOperatingStateChangedEvent()`
	`WaitForOnOperatingStateChangedEvent()`

OnLedChanged (Page 285)	`OnLedChanged`
	`RegisterOnLedChangedEvent()`
	`UnregisterOnLedChangedEvent()`
	`WaitForOnLedChangedEvent()`

OnConfigurationChanging (Page 287)	`OnConfigurationChanging`
	`RegisterOnConfigurationChangingEvent()`
	`UnregisterOnConfigurationChangingEvent()`
	`WaitForOnConfigurationChangingEvent()`

OnConfigurationChanged (Page 290)	`OnConfigurationChanged`
	`RegisterOnConfigurationChangedEvent()`
	`UnregisterOnConfigurationChangedEvent()`
	`WaitForOnConfigurationChangedEvent()`

OnSyncPointReached (Page 292)	`OnSyncPointReached`
	`RegisterOnSyncPointReachedEvent()`
	`UnregisterOnSyncPointReachedEvent()`
	`WaitForOnSyncPointReachedEvent()`

API IRemoteRuntimeManager

Table 7- 15 Overview of IRemoteRuntimeManager functions - .NET (C#)

Settings	Functions

Interface (Page 304)	`Dispose()`
	`Version { get; }`
	`IP { get; }`
	`Port { get; }`
	`RemoteComputerName { get; }`
	`Disconnect()`
	`StrictMotionTiming { get; set; }`

Simulation Runtime instances (Page 118)	`RegisterInstanceInfo { get; }`
	`RegisterInstance()`
	`RegisterCustomInstance()`
	`CreateInterface()`

Table 7- 16 Overview IRemoteRuntimeManager events - .NET (C#)

Events	Functions

OnConnectionLost() (Page 317)	`OnConnectionLost()`
	`RegisterOnConnectionLostEvent()`
	`UnregisterOnConnectionLostEvent()`
	`WaitForOnConnectionLostEvent()`

7.1.5 Overview of data types for native C++

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

Table 7- 17 Overview of data types - Native C++

Data type	
DLL import functions (Page 321)	`ApiEntry_Initialize` `ApiEntry_DestroyInterface`

Event callback functions (Page 322)	`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`
	`Event-Callback_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`

Definitions and constants (Page 349)	
Unions (Page 350)	`UIP` `UDataValue`

Data type	

Structures (Page 353)	`SDataValue`
	`SDVBNI`
	`SDataValueByAddress`
	`SDataValueByName`
	`SConnectionInfo`
	`SInstanceInfo`
	`SDimension`
	`STagInfo`
	`SIP`
	`SIPSuite4`
	`SOnSyncPointReachedResult`
	`SDataValueByAddressWithCheck`
	`SDataValueByNameWithCheck`
	`SDataRecordInfo`
	`SDataRecord`
	`SConfiguredProcessEvents`
	`SDiagExtChannelDescription`
	`SAutodiscoverData`

Enumerations (Page 375)	`ERuntimeErrorCode`
	`EArea`
	`EOperatingState`
	`EOperatingMode`
	`ECPUType`
	`ECommunicationInterface`
	`ELEDType`
	`ELEDMode`
	`EPrimitiveDataTypes`
	`EDataType`
	`ETagListDetails`
	`ERuntimeConfigChanged`
	`EInstanceConfigChanged`
	`EPullOrPlugEventType`
	`EProcessEventType`
	`EDirection`
	`EDiagProperty`
	`EDiagSeverity`
	`ERackOrStationFaultType`
	`ECycleTimeMonitoringMode`
	`EAutodiscoverType`

7.1.6 Overview of data types for managed code

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

Table 7- 18 Overview of data types - .NET (C#)

Data type	

Delegate definitions (Page 335)
- 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`
	`Dele-gate_II_EREC_DT_SRICC_UINT32_UINT32_UINT32_UINT32`
	`Delegate_II_EREC_DT_ELT_ELM`
	`Delegate_II_EREC_DT_SDRI`
	`Delegate_II_EREC_DT_SDR`
	`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 349)	

Structures (Page 353)	`SDataValue`
	`SDVBNI`
	`SDataValueByAddress`
	`SDataValueByName`
	`SConnectionInfo`
	`SInstanceInfo`
	`SDimension`
	`STagInfo`
	`SIP`
	`SIPSuite4`
	`SOnSyncPointReachedResult`
	`SDataValueByAddressWithCheck`
	`SDataValueByNameWithCheck`
	`SDataRecordInfo`
	`SDataRecord`
	`SConfiguredProcessEvents`
	`SDiagExtChannelDescription`
	`SAutodiscoverData`

Data type	

Enumerations (Page 374)	`ERuntimeErrorCode`
	`EArea`
	`EOperatingState`
	`EOperatingMode`
	`ECPUType`
	`ECommunicationInterface`
	`ELEDType`
	`ELEDMode`
	`EPrimitiveDataTypes`
	`EDataType`
	`ETagListDetails`
	`ERuntimeConfigChanged`
	`EInstanceConfigChanged`
	`EPullOrPlugEventType`
	`EProcessEventType`
	`EDirection`
	`EDiagProperty`
	`EDiagSeverity`
	`ERackOrStationFaultType`
	`ECycleTimeMonitoringMode`
	`EAutodiscoverType`

7.2 Initialize API

7.2.1 Load API library

Description

For PLCSIM Advanced, the interfaces of the API V4.0 are not compatible with the interfaces of previous API versions. However, the Runtime Manager of PLCSIM Advanced V4.0 is compatible with the API of previous PLCSIM Advanced versions.

Earlier versions of the API are also installed during the installation of PLCSIM Advanced V4.0.

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

The installation path of 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\4.0").

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 100).
- For .NET via the call of the function "System.Reflection.Assembly.LoadFile(string)" in the online documentation for MSDN.

7.2.2 Native C++

7.2.2.1 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 which is also compatible with the header file of the API ("SimulationRuntimeApi.h").

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

- In the directory to which the parameter of the function leads (`in_SimulationRuntimeApiDllPath`)
- In the directory of your application that calls this function.
- In the installation directory of PLCSIM Advanced

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

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

Table 7- 19 InitializeApi() - Native C++

Syntax	`ERuntimeErrorCode InitializeApi( ISimulationRuntimeManager** out_SimulationRuntimeManagerInterface );`
	`ERuntimeErrorCode InitializeApi( WCHAR* in_SimulationRuntimeApiDllPath, ISimulationRuntimeManager** in-out_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 320). • `WCHAR* in_SimulationRuntimeApiDllPath`: The path to the Runtime API library.	
Return values	Runtime error code	Condition
	`SREC_OK`	The function is successful.
	`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 Compatibility during upgrade (Page 17).
	`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 it.

See DestroyInterface() (Page 105).

7.2.2.2 RuntimeApiEntry_Initialize

Description

Use the function `RuntimeApiEntry_Initialize` only if the API library (DLL) is to be loaded from a different directory than the directory of your application that calls 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 instance of the virtual controller or to obtain access to an existing instance.

Table 7- 20 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 320). • `UINT32 in_InterfaceVersion:` Version of the API interface to be downloaded: `DAPI_DLL_INTERFACE_VERSION`.	
Return values	Runtime error code	Condition
	`SREC_OK`	The function is successful.
	`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 Compatibility during upgrade (Page 17).
	`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 Win32 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 it.

See DestroyInterface() (Page 105).

7.2.3 .NET (C#)

7.2.3.1 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 the first time.

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

Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationInitializationException`	
	Runtime error code	Condition
	`ERuntimeErrorCode.ConnectionError`	Unable to establish a connection to the Runtime Manager.
	`ERuntimeErrorCode.WrongVersion`	The version of the API is not compatible with Runtime. See Compatibility during upgrade (Page 17).
	`ERuntimeErrorCode.RuntimeNotAvailable`	No Runtime Manager runs in this Windows user session.
	`ERuntimeErrorCode.ConfigFileError`	Operation regarding the configuration file "UserInterfaceConfiguration.xml" has failed, for example, create, read, write.

7.3 Shut down API

7.3.1 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 Win32 API-Funktion `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 107)
- ShutdownAndFreeApi() (Page 108)

7.3.1.1 DestroyInterface()

Description

A function pointer to the `RuntimeApiEntry_DestroyInterface` function. The function pointer `DestoyInterface()` is only valid if the `InitializeApi` function has been successfully called.

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

Table 7- 22 DestroyInterface() - Native C++

Syntax	`ERuntimeErrorCode 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`.
Example C++	See code below	

```
// 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;
```

7.3.1.2 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 directory of the application that calls this function.

If the API was initialized using the `InitializeApi` function, you select the DestroyInterface() (Page 105) function.

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

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

Syntax	`__declspec(dllexport) ERuntimeErrorCode RuntimeA-`
	`piEntry_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`.

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 Win32 API)`
	`dllHandle = LoadLibraryA(DAPI_DLL_NAME_X86);`
	`if ( dllHandle != NULL )`
	`{`
	` Destroy = (ApiEntry_ DestroyInter-`
	`face)GetProcAddress(dllHandle,`
	` DAPI_ENTRY_DESTROY_INTERFACE);`
	`}`
	…
	`// Frees the memory of an IInstance interface`
	`result =` **Destroy**`(instance);`	

7.3.1.3 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` function. If the `InitializeApi` function was not called, the library must be unloaded using the Win32 API function `FreeLibrary()`.

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

Syntax	`ERuntimeErrorCode FreeApi();`	
Parameters	None	
Return values	Runtime error code	Condition
	`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**`();`	

7.3.1.4 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` function. If the `InitializeApi` function was not called, the library must be unloaded using the Win32 API-Funktion `FreeLibrary()`.

Table 7- 25 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	Condition
	`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;`	

7.3.2 .NET (C#)

7.3.2.1 Shut down API

You can terminate the .NET components of the API for the IInstance and IRemoteRuntimeManager interfaces by calling the Dispose (Page 138) function.

In addition these interfaces can also be cleared automatically by the .NET Garbage Collector.

Manually clearing the API

To manually clear the API, follow these steps:

1. Delete all interfaces. Interfaces - Information and settings (Page 138)
2. Call the Shutdown() (Page 114) function of the ISimulationRuntimeManager interface.

7.4 Global functions (Native C++)

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

GetNameOfAreaSection()

Table 7- 26 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 7- 27 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 7- 28 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 7- 29 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 7- 30 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 7- 31 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 7- 32 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 7- 33 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

GetNameOfErrorCode()

Table 7- 34 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

GetNameOfOperatingState

Table 7- 35 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 7- 36 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 7- 37 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 7- 38 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 7- 39 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 7- 40 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 7- 41 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 7- 42 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 7- 43 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 7- 44 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 7- 45 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 7- 46 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 7- 47 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

See also

EPrimitiveDataType (Page 385)

EDataType (Page 388)

Enumerations (Page 374)

7.5 API ISimulationRuntimeManager

7.5.1 Interfaces - Information and settings

GetVersion() / Version { get; }

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

Table 7- 48 GetVersion() - Native C++

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

Table 7- 49 Version { get; } - .NET (C#)

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

IsInitialized() / IsInitialized { get; }

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

Table 7- 50 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- 51 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 131).

Table 7- 52 IsRuntimeManagerAvailable() - Native C++

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

Table 7- 53 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- 54 Shutdown() - Native C++

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

Table 7- 55 Shutdown() - .NET (C#)

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

GetStrictMotionTiming() / StrictMotionTiming { get; }

Returns the current global setting for the "Strict Motion Timing" feature that has an effect on newly created instances.

Table 7- 56 GetStrictMotionTiming() - Native C++

Syntax	`ERuntimeErrorCode GetStrictMotionTiming(bool* enabled);`	
Parameters	bool* enabled: Receives the current setting. `true`: Active `false`: Inactive	
Return values	Runtime error code	Condition
	`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 UserInterfaceConfigura-tion.xml.

Table 7- 57 StrictMotionTiming { get; } - .NET (C#)

Syntax	`bool StrictMotionTiming { get; }`	
Parameters	None	
Return values	`Sie-mens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.ConfigFileError`	The setting could not be read from the configuration file UserInterfaceConfigu-ration.xml.

SetStrictMotionTiming() / StrictMotionTiming { set; }

Sets the global setting for the "Strict Motion Timing" feature that has an effect on newly created instances.

Table 7- 58 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	Condition
	`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 UserInterfaceConfigura-tion.xml.

Table 7- 59 StrictMotionTiming { set; } - .NET (C#)

Syntax	`bool StrictMotionTiming { set; }`	
Parameters	None	
Return values	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`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 UserInterfaceConfigu-ration.xml.

7.5.2 Simulation Runtime instances

GetRegisteredInstancesCount()

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

Table 7- 60 GetRegisteredInstancesCount() - Native C++

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

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 7- 61 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. See Data types (Page 349).	
Return values	Runtime error code	Condition
	`SREC_OK`	The function is successful.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_DOES_NOT_EXIST`	There is no instance information for this index.
	`SREC_INDEX OUT_OF_RANGE`	The index is greater than 15.

RegisteredInstanceInfo { get; }

Returns information about all already registered instances. Use the ID or name of this instance to create an interface of this instance, see `CreateInterface()`.

Table 7- 62 RegisteredInstanceInfo { get; } - .NET (C#)

Syntax	`SInstanceInfo[] RegisteredInstanceInfo { get; }`	
Parameters	None	
Return values	`SInstanceInfo[]`: An array of information about all registered instances.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeErrorCode.Timeout`	The function does not return on time.

RegisterInstance()

Registers a new instance of a virtual controller in Runtime Manager. Creates and returns an interface of this instance.

Table 7- 63 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`
	`);`	

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`. See Data types (Page 349). • `IInstance** out_InstanceInterface:` Pointer to a Simulation Runtime interface pointer. The pointer must be initialized with `ZERO`. The interface is created within the function.	
Return values	Runtime error code	Condition
	`SREC_OK`	The function is successful.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_WRONG_ARGUMENT`	The name or the IInstance pointer is invalid.
	`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 = ZERO;`
	`ERuntimeErrorCode result = Initialize(&api);`
	
	`// Example: How To Create And Register An Instance`
	`IInstance* psa = ZERO;`
	`if (result == SREC_OK)`
	`{`
	` result = api->RegisterInstance(&psa);`
	`}`	

Note

Native C++

If you no longer require the interface, delete it.

See DestroyInterface() (Page 105).

Table 7- 64 RegisterInstance() - .NET (C#)

Syntax	`IInstance RegisterInstance();`
	`IInstance RegisterInstance(`
	` string in_InstanceName`
	`);`
	`IInstance RegisterInstance(`
	` ECPUType in_CPUType`
	`);`
	`IInstance RegisterInstance(`
	` ECPUType in_CPUType`
	` 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 349).	
Return values	If the function is successful, an interface of a virtual controller, otherwise a null pointer.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.WrongArgument`	The name is invalid.
	`ERuntimeError-Code.LimitReached`	There are already 16 instances registered in Runtime Manager.
	`ERuntimeError-Code.AlreadyExists`	An instance with this name already exists.

RegisterCustomInstance()

Registers a new instance of a virtual controller in Runtime Manager. Creates and returns an interface of this instance.

Table 7- 65 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 virtual controller 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 349). • `IInstance** out_InstanceInterface:` Pointer to a Simulation Runtime interface pointer. The pointer must be initialized with `ZERO`. The interface is created within the function.	
Return values	Runtime error code	Condition
	`SREC_OK`	The function is successful.
	`SREC_TIMEOUT`	The function does not return on time.
	`SREC_WRONG_ARGUMENT`	The DLL name, the instance name or the IInstance pointer is invalid.
	`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 = ZERO;`
	`ERuntimeErrorCode result = Initialize(&api);`
	
	`// Example: How To Create And Register An Instance`
	`IInstance* psa = ZERO;`
	`if (result == SREC_OK)`
	`{`
	` result = api->`**RegisterCustomInstance**`(L"C:\\Temp\\vplc.dll");`
	`}`	

Note

Native C++

If you no longer require the interface, delete it.

See DestroyInterface() (Page 105).

Table 7- 66 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 virtual controller 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 349).	
Return values	If the function is successful, an interface of a virtual controller; otherwise a Null pointer.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.WrongArgument`	The name or the ID is invalid.
	`ERuntimeError-Code.LimitReached`	There are already 16 instances registered in Runtime Manager.
	`ERuntimeError-Code.AlreadyExists`	An instance with this name already exists.

CreateInterface()

Creates and returns an interface of an already registered instance of a virtual controller.

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

Table 7- 67 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 `ZERO`. The interface is created within the function.	
Return values	Runtime error code	Condition
	`SREC_OK`	The function is successful.
	`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 = ZERO;`
	`ERuntimeErrorCode result = Initialize(&api);`
	
	`IInstance* psa1 = ZERO;`
	`IInstance* psa2 = ZERO;`
	`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 = ZERO;`
	`ERuntimeErrorCode result = Initialize(&api);`
	
	`IInstance* psa = ZERO;`
	`if (result == SREC_OK)`
	`{`
	` result = api->`**CreateInterface**`(L"My SimulationRuntime Instance", &psa);`
	`}`	

Note

Native C++

If you no longer require the interface, delete it.

See DestroyInterface() (Page 105).

Table 7- 68 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 inter-face.	
Return values	If the function is successful, an interface of a virtual controller; otherwise a Null pointer.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.WrongArgument`	The name or the ID is invalid.
	`ERuntimeError-Code.DoesNotExists`	The instance is not registered in Runtime Manager.

7.5.3 Remote connections

OpenPort()

Opens a port to which another Runtime Manager can connect.

Table 7- 69 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	Condition
	`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- 70 OpenPort() - .NET (C#)

Syntax	`void OpenPort(`
	` UInt16 in_Port`
	`);`	

Parameters	• `UInt16 in_Port:` The port. The value must be greater than 1024.	
Return values	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.AlreadyExists`	A port is already open.
	`ERuntimeError-Code.WrongArgument`	The port is invalid.
	`ERuntimeError-Code.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- 71 ClosePort() - Native C++

Syntax	`ERuntimeErrorCode ClosePort();`	
Parameters	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`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- 72 ClosePort() - .NET (C#)

Syntax	`void ClosePort(`
	` UInt16 in_Port`
	`);`	

Parameters	None	
Return values	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeErrorCode.Timeout`	The function does not return on time.

GetPort() / Port { get; }

Returns the open port. If no port is open or the function fails, the return value is 0.

Table 7- 73 GetPort() - Native C++

Syntax	`UINT16 GetPort();`
Parameters	None
Return values	`UINT16`: The open port. 0, if no port is open.

Table 7- 74 Port { get; } - .NET (C#)

Syntax	`UInt16 Port { get; }`
Parameters	None
Return values	`UInt16`: The open port. 0, if no port is open.
Exceptions	None

GetRemoteConnectionsCount()

Supplies the number of open remote connections.

Table 7- 75 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- 76 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	Condition
	`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- 77 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	Condition
	`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- 78 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 `ZERO`. The interface is created in the function.

Return values	Runtime error code	Condition
	`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 Compatibility during upgrade (Page 17).

Example C++

```
ISimulationRuntimeManager* api = ZERO;
ERuntimeErrorCode result = Initialize(&api);

IRemoteRuntimeManager * client = ZERO;

if (result == SREC_OK)
{
 result = api->RemoteConnect(192,203,145,144, 4444, &client);
}
```

Note

Native C++

If you no longer require the interface, delete it.

See DestroyInterface() (Page 105).

Table 7- 79 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	Condition
	`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 Compatibility during upgrade (Page 17).

See also

Tag list (Page 167)

Data types (Page 320)

7.5.3.1 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 PLCSIM Advanced V4.0.

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- 80 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	Condition
	`SREC_OK`	The function is successful.
	`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 398).
	`SREC_TIMEOUT`	Communication errors in the local Runtime Manager.

Table 7- 81 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.	
Return values	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeErrorCode.WrongArgument`	The timeout value is outside the permissible range.
	`ERuntimeErrorCode.AutodiscoverAlreadyRunning`	A `RunAutodiscover()` call is already running in the background. Wait for the message `AutodiscoverFinished` in the callback function. See EAutodiscoverType (Page 398).
	`ERuntimeErrorCode.Timeout`	Communication errors in the local Runtime Manager.

7.5.4 Events for ISimulationRuntimeManager

Events for runtime instances and remote connections

The following events are triggered for the ISimulationRuntimeManager interface:

Table 7- 82 Events for ISimulationRuntimeManager

Event	**Cause**
OnConfigurationChanged (Page 132)	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 134)	The connection to the Runtime Manager is interrupted.
RunAutodiscover (Page 137)	The network searches for Runtime Managers which are ready to establish a remote connection.

7.5.4.1 OnConfigurationChanged events

OnConfigurationChanged

Registers or unregisters an event handler method.

Table 7- 83 OnConfigurationChanged - .NET (C#)

Syntax	`event Delegate_SRCC_UINT32_UINT32_INT32 OnConfiguration-Changed;`
Parameters	None. See Delegate_SRCC_UINT32_UINT32_INT32 (Page 336).
Return values	None
Exceptions	None
Note	The event handler method runs in a separate thread.

RegisterOnConfigurationChangedCallback()

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 7- 84 RegisterOnConfigurationChangedCallback() - Native C++

Syntax	`void RegisterOnConfigurationChangedCallback( EventCallback_SRCC_UINT32_UINT32_INT32 in_CallbackFunction );`
Parameters	• `EventCallback_SRCC_UINT32_UINT32_INT32 in_CallbackFunction`: A callback function to subscribe to an event. See EventCallback_SRCC_UINT32_UINT32_INT32 (Page 322).
Return values	None
Note	The event handler method runs in a separate thread.

RegisterOnConfigurationChangedEvent()

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- 85 RegisterOnConfigurationChangedEvent() - Native C++

Syntax	`void RegisterOnConfigurationChangedEvent();`
	`void RegisterOnConfigurationChangedEvent( 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 7- 86 RegisterOnConfigurationChangedEvent() - .NET (C#)

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

UnregisterOnConfigurationChangedCallback()

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

Table 7- 87 UnregisterOnConfigurationChangedCallback() - Native C++

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

UnregisterOnConfigurationChangedEvent()

Unregisters the event object.

Table 7- 88 UnregisterOnConfigurationChangedEvent() - Native C++

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

Table 7- 89 UnregisterOnConfigurationChangedEvent() - .NET (C#)

Syntax	`void UnregisterOnConfigurationChangedEvent();`
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 7- 90 WaitForOnConfigurationChangedEvent() - Native C++

Syntax	`bool WaitForOnConfigurationChangedEvent();`
	`bool WaitForOnConfigurationChangedEvent(`
	` 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 7- 91 WaitForOnConfigurationChangedEvent - .NET (C#)

Syntax	`bool WaitForOnConfigurationChangedEvent();`
	`bool WaitForOnConfigurationChangedEvent(`
	` 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.

7.5.4.2 OnRuntimeManagerLost events

OnRuntimeManagerLost

Registers or unregisters an event handler method.

Table 7- 92 OnRuntimeManagerLost - .NET (C#)

Syntax	`event Delegate_Void OnRuntimeManagerLost;`
Parameters	None. See Delegate_Void (Page 335).
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. 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 7- 93 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 322).
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- 94 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- 95 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- 96 UnregisterOnRuntimeManagerLostCallback() - Native C++

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

UnregisterOnRuntimeManagerLostEvent()

Unregisters the event object.

Table 7- 97 UnregisterOnRuntimeManagerLostEvent() - Native C++

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

Table 7- 98 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- 99 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- 100 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.

7.5.4.3 OnAutodiscoverData events

OnAutodiscoverData

Registers or unregisters an event handler method.

Table 7- 101 OnAutodiscoverData - .NET (C#)

Syntax	`event Delegate_SRRSI_AD OnAutodiscoverData`
Parameters	None. See Delegate_SRRSI_AD (Page 337).
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. 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 7- 102 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 323).
Return values	None

UnregisterOnAutodiscoverCallback()

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

Table 7- 103 UnregisterOnAutodiscoverCallback() - Native C++

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

7.6 API IInstances

7.6.1 Interfaces - Information and settings

Dispose()

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

Table 7- 104 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 7- 105 GetID() - Native C++

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

Table 7- 106 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 7- 107 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 349). • `UINT32 in_ArrayLength:` Field length (Wide character)	
Return values	Runtime error code	Condition
	`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++	see below	

```
ISimulationRuntimeManager * api = NULL;
ERuntimeErrorCode result = Initialize(&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 7- 108 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	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.

GetCPUType()

Returns the CPU type of the virtual controller.

Table 7- 109 GetCPUType() - Native C++

Syntax	`ECPUType GetCPUType();`
Parameters	None
Return values	An enumeration element that defines the CPU type. See ECPUType (Page 382).

SetCPUType()

Sets the CPU type of the virtual controller. A change of CPU type occurs only when the controller is restarted.

Table 7- 110 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 virtual controller. 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.

Table 7- 111 CPUType { get; set; } - .NET (C#)

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

GetCommunicationInterface()

Returns the communication interface of the virtual controller: Local communication (Softbus) or TCPIP. A change of communication interface occurs only when the controller is restarted. All instances that are started must use the same communication interface.

PowerOn is prevented if a communication interface that is not used by the started instances is selected.

Table 7- 112 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 virtual controller uses the Softbus. • `SRCI_TCPIP` Is returned if the virtual controller communicates over the virtual adapter.

SetCommunicationInterface()

Sets the communication interface of the virtual controller: Local communication (Softbus) or TCPIP. A change of communication interface occurs only when the controller is restarted. All instances that are started must use the same communication interface.

PowerOn is prevented if a communication interface that is not used by the started instances is selected.

Table 7- 113 SetCommunicationInterface() - Native C++

Syntax	`void SetCommunicationInterface(ECommunicationInterface in Value);`
Parameters	• `SRCI_NONE` Cannot be selected. • `SRCI_SOFTBUS` Is set to activate communication via Softbus. • `SRCI_TCPIP` Is set to activate communication over the virtual adapter.
Return values	None

CommunicationInterface { get; set; }

Sets or returns the communication interface of the virtual controller: Local communication (Softbus) or TCPIP. A change of communication interface occurs only when the controller is restarted. All instances that are started must use the same communication interface.

PowerOn is prevented if a communication interface that is not used by the started instances is selected.

Table 7- 114 CommunicationInterface { get; set; } - .NET (C#)

Syntax	`ECommunicationInterface CommunicationInterface { get; set; }`
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 virtual controller uses the Softbus. • `ECommunicationInterface.TCPIP` Is returned if the virtual controller communicates over the virtual adapter.
Exceptions	None

GetInfo() / Info { get; }

Returns a structure that provides information about the instance.

Table 7- 115 GetInfo() - Native C++

Syntax	`SInstanceInfo GetInfo();`
Parameters	None
Return values	`SInstanceInfo`: A structure that provides information about the instance. See SInstanceInfo (Page 360).

Table 7- 116 Info { get; } - .NET (C#)

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

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.

Table 7- 117 UnregisterInstance() - Native C++

Syntax	`ERuntimeErrorCode UnregisterInstance();`	
Parameters	None	
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.

Table 7- 118 UnregisterInstance() - .NET (C#)

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

7.6.2 Controller - Information and settings

GetControllerName() / ControllerName { get; }

Returns the downloaded name of the virtual controller.

Table 7- 119 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	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_INDEX_OUT_OF_RANGE`	The name does not fit in the storage.

Table 7- 120 ControllerName { get; } - .NET (C#)

Syntax	`string ControllerName { get; }`	
Parameters	None	
Return values	`string:` The downloaded name of the virtual controller.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeError-Code.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 virtual controller.

Table 7- 121 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	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_INDEX_OUT_OF_RANGE`	The name does not fit in the storage.

Table 7- 122 ControllerShortDesignation { get; } - .NET (C#)

Syntax	`string ControllerShortDesignation { get; }`	
Parameters	None	
Return values	`string:` The downloaded short designation of the virtual controller.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeError-Code.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 virtual controller. If the function fails, the return value is 0.

Table 7- 123 GetControllerIPCount() - Native C++

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

GetControllerIP() / ControllerIP { get; }

Returns a configured IP address of the instance.

Table 7- 124 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 virtual controller. If the function fails, the return value is 0.

Table 7- 125 ControllerIP { get; } - .NET (C#)

Syntax	`string[] ControllerIP { get; }`
Parameters	None
Return values	`string`: All downloaded IP addresses of the virtual controller. 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 7- 126 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 virtual controller. If the function fails, the return values are 0.

Table 7- 127 ControllerIPSuite4 { get; } - .NET (#)

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

SetIPSuite()

Sets the IP suite of the network interface of a virtual controller.

Table 7- 128 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 virtual controller. If the communication interface is "Softbus", this flag is ignored.	
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 virtual controller is not running.
	`SREC_DOES_NOT_EXIST`	There is no network interface with this ID.
	`SREC_INVALID_OPERATING_STATE`	The virtual controller has not yet com-pleted the boot process or is already in the shutdown phase.

Table 7- 129 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 virtual controller. If the communication interface is "Softbus", this flag is ignored.	
Return values	None	
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.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeErrorCode.DoesNotExist`	There is no network interface with this ID.
	`ERuntimeErrorCode.InvalidOperatingState`	The virtual controller has not yet completed the boot process or is already in the shutdown phase.

GetStoragePath()

Returns the full directory in which the instance stores its data.

Table 7- 130 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 320). • `UINT32 in_ArrayLength:` Length of the array (Wide character)	
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_INDEX_OUT_OF_RANGE`	The path does not fit in the storage.

SetStoragePath()

Sets the full path of the directory 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 7- 131 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`. See Data types (Page 320).	
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_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 directory 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 7- 132 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	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.IndexOutOfRange`	The length of the path exceeds the limit.
	`ERuntimeError-Code.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 instance of the virtual controller must be in OFF operating state for this.

Table 7- 133 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	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_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 directory 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. The user has to execute the SetStoragePath() function beforehand.

Table 7- 134 ArchiveStorage() - .NET (C#)

Syntax	`void ArchiveStorage( string in_FullFileName );`	
Parameters	• `string 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	Condition
	`SREC_NO_STORAGE_PATH_SET`	No storage path is set. The user has to execute the SetStoragePath() function beforehand.
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.InvalidOperatingState`	The instance is not in OFF operating state.
	`ERuntimeErrorCode.InvalidArchivePath`	The archive path is invalid.
	`ERuntimeErrorCode.CreateDirectoriesFailed`	The directory for the ZIP file could not be created.
	`ERuntimeErrorCode.ArchiveStorageNotCreated`	The ZIP file could not be created.
	`ERuntimeErrorCode.StorageTransferError`	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 virtual controller must be in OFF operating state for this.

Table 7- 135 RetrieveStorage() - Native C++

Syntax	`ERuntimeErrorCode RetrieveStorage(`
	` 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	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_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 7- 136 RetrieveStorage() - .NET (C#)

Syntax	`void RetrieveStorage( string in_FullFileName );`	
Parameters	• `string 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	Condition
	`SREC_NO_STORAGE_PATH_SET`	No storage path is set. The user has to execute the SetStoragePath() function beforehand.
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InvalidOperatingState`	The instance is not in OFF operating state.
	`ERuntimeError-Code.InvalidArchivePath`	The archive path is invalid.
	`ERuntimeError-Code.DeleteExistingStorageFailed`	The old storage cannot be deleted.
	`ERuntimeError-Code.RetrieveStorageFailure`	The ZIP file cannot be unzipped.
	`ERuntimeError-Code.StorageTransferError`	Error during network data transfer. Memory data between client and server computers do not match.

CleanupStoragePath()

The function deletes the directory 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 directory is missing, the function is considered successful.

To make sure that the correct directory is deleted, the function checks if 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 directory, only the following directories with files are also allowed:

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

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

Table 7- 137 CleanupStoragePath() - Native C++

Syntax	`ERuntimeErrorCode CleanupStoragePath( );`	
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_INVALID_OPERATING_STATE`	The instance is not in OFF operating state.
	`SREC_DELETE_EXISTING_STORAGE_FAILED`	The directory with the memory cannot be deleted.
	`SREC_INVALID_STORAGE`	The memory is invalid. It contains files or directories that are not permitted.

Table 7- 138 CleanupStoragePath() - .NET (C#)

Syntax	`void CleanupStoragePath( );`	
Return values	None	
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.InvalidOperatingState`	The instance is not in OFF operating state.
	`ERuntimeErrorCode.DeleteExistingStorageFailed`	The directory with the memory cannot be deleted.
	`SREC_INVALID_STORAGE`	The memory is invalid. It contains files or directories that are not permitted.

GetStrictMotionTiming() / StrictMotionTiming { get; }

Returns the current instance setting for the "Strict Motion Timing" feature.

Table 7- 139 GetStrictMotionTiming() - Native C++

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

Table 7- 140 StrictMotionTiming { get; } - .NET (C#)

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

SetStrictMotionTiming() / StrictMotionTiming { set; }

Sets the instance setting for the "Strict Motion Timing" feature.

Table 7- 141 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	Condition
	`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 7- 142 StrictMotionTiming { set; } - .NET (C#)

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

7.6.3 Operating state

PowerOn()

The function creates the process for the Simulation Runtime instance and starts the firmware of the virtual controller.

Table 7- 143 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 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	Condition
	`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 virtual switch is configured incorrectly.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the virtual controller is no longer running.
	`SREC_UNSUPPORTED_PCAP_DRIVER`	The PCAP driver used is not supported. 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. 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 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_AVAILABLE`	**For local communication via Softbus** PLCSIM Advanced cannot connect to the Softbus. **Solution** • Try again to establish the connection. • Close PLCSIM Advanced and the TIA Portal and restart the applications. • Reboot the PC. • Repair the 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 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 differs from the first instance that is already active. **Example:** 1. PowerOn() 1. instance with Softbus is successful. 2. PowerOn() 2. instance with TCP/IP, the error code is returned.

	`SREC_WARNING_PASSWORD_PROTECTION_ERROR`	The password protection prevented the memory from being opened. View the restrictions with activated password protection.

Table 7- 144 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 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	Runtime error code	Condition
	`ERuntimeErrorCode.OK`	The function is successful.
	`ERuntimeErrorCode.WarningAlreadyExists`	Warning: The instance is started.
	`ERuntimeErrorCode.UnsupportedPcapDriver`	The PCAP driver used is not supported. 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. 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.
	`ERuntimeErrorCode.NotEmpty`	Warning: No valid license for 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.

	`ERuntimeError-Code.CommunicationInterfaceNotAvailable`	**For local communication via Softbus** PLCSIM Advanced cannot connect to the Softbus. **Solution** • Try again to establish the connection. • Close PLCSIM Advanced and the TIA Portal and restart the applications. • Reboot the PC. • Repair the 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 PLCSIM Advanced installation.
	`ERuntimeError-Code.AccessDenied`	No write access in the Storage Directory.
	`ERuntimeError-` `Code.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".
	`ERuntimeError-Code.WrongCommunicationInterface`	You have tried to supply the second instance with the communication interface that differs from the first instance that is already active. Example: 1. PowerOn() 1. instance with Softbus is successful. 2. PowerOn() 2. instance with TCP/IP, the error code is returned.
	ERuntimeError-Code.`WarningPasswordForProtectionError`	The password protection prevented the memory from being opened. View the restrictions with activated password protection.

Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The expected operating state does not occur on time.
	`ERuntimeErrorCode.ErrorLoadingDll`	The "Siemens.Simatic.Simulation.Runtime.Instance.exe" cannot load the "Siemens.Simatic.PlcSim.Vplc1500.dll".
	`ERuntimeErrorCode.StoragePathAlreadyInUse`	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.VirtualSwitchMisconfigured`	The virtual switch is configured incorrectly.
	`ERuntimeErrorCode.InstanceNotRunning`	The process of the virtual controller is no longer running.

PowerOff()

Shuts down the Simulation Runtime and closes its process.

Table 7- 145 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 value of 60000 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	Condition
	`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 virtual controller is not running.

Table 7- 146 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 value of 60000 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	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The expected operating state does not occur on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.

Run()

Calls on the virtual controller to change to RUN operating state.

Table 7- 147 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 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	Condition
	`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 virtual controller is not running.

Table 7- 148 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 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 }`	
Return values	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The expected operating state does not occur on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.

Stop()

Calls on the virtual controller to change to STOP operating state.

Table 7- 149 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 value of 60000 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	Condition
	`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 virtual controller is not running.

Table 7- 150 Stop() - .NET (C#)

Syntax	`void Stop();`
	`void Stop(`
	` bool in_IsSynchronous`
	`);`	

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 state if this function is successful: `{ EOperatingState.Stop }`	
Return values	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The expected operating state does not occur on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.

GetOperatingState() / OperatingState { get; }

Returns the operating state of the virtual controller. When the operating state changes, the OnOperatingStateChanged() (Page 281) event is triggered. For details about the operating state, see Data types (Page 380).

Table 7- 151 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 virtual controller is not yet ready to start the user program. • `SROS_STOP:` If the virtual controller 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 virtual controller is still in the Shutdown phase.

Table 7- 152 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 virtual controller is not yet ready to start the user program. • `EOperatingState.Stop:` If the virtual controller 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 virtual controller is still in the Shutdown phase.

MemoryReset()

Shuts down the virtual controller, closes its processes and performs a restart.

Table 7- 153 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	Condition
	`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 virtual controller is not running.

Table 7- 154 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	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The expected operating state does not occur on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.

7.6.4 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 virtual controller 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 7- 155 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	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 virtual controller 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 quo- tation mark.

Table 7- 156 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	Condition
	`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 virtual controller 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 7- 157 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	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.

Table 7- 158 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`	
	Runtime error code	Condition
	`ERuntimeError-Code.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 7- 159 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 7- 160 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	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 virtual controller is not running.
	`SREC_INDEX_OUT_OF_RANGE`	The elements do not fit in the storage.

Table 7- 161 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	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.

CreateConfigurationFile()

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

Table 7- 162 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	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 file name is invalid.

Table 7- 163 CreateConfigurationFile() - .NET (C#)

Syntax	`void CreateConfigurationFile(`
	` string in_FullFileName`
	`);`	

Parameters	None	
Return values	• `string in_FullFileName:` File name of the XML file that is to be written to: <Path> + <File name> + <File extension>.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.WrongArgument`	The file name is invalid.

7.6.5 I/O access

7.6.5.1 Synchronizing inputs and outputs

Description

In PLCSIM Advanced the complete scope of the input and output area is used (see GetAreaSize/AreaSize (Page 174)). This is also possible when no IO module is configured.

Inputs and outputs that are defined via configured IO modules are synchronized to the defined update of the process image partition (PIP).

Inputs and outputs that are not assigned to an IO 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 the values via the API, but values which are written via the user program (TIA Portal) are not visible in the API.
- Outputs can be used as output and as input.

  You can write the values via the API and via the CPU / the user program (TIA Portal). If API and user program write to the same area, the values from the API will overwrite the values from the user program.

7.6.5.2 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 7- 164 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 7- 165 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 379).
Return values	`UINT32:` Size of the area in bytes. If the function was successful, the value is not equal to 0.

Table 7- 166 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 virtual controller.

Therefore, use access via the tag name (Page 189) and not via the address areas.

Table 7- 167 ReadBit() - Native C++

Syntax	`ERuntimeErrorCode ReadBit(`
	` EArea in_Area,`
	` UINT32 in_Offset,`
	` UINT8 in_Bit,`
	` bool* 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 379). • `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	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 virtual controller is not running.
	`SREC_INDEX_OUT_OF_RANGE`	Offset or bits are invaid.
	`SREC_WRONG_ARGUMENT`	The area is invalid.

Table 7- 168 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	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeError-Code.IndexOutOfRange`	Offset or bits are invaid.

ReadByte()

Reads an individual bit from the area.

Note

The function allows access to the entire storage area of the virtual controller.

Therefore, use access via the tag name and not via the address areas.

Table 7- 169 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 379). • `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	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 virtual controller is not running.
	`SREC_INDEX_OUT_OF_RANGE`	Offset is invalid.
	`SREC_WRONG_ARGUMENT`	The area is invalid.

Table 7- 170 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	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeError-Code.IndexOutOfRange`	Offset is invalid.

ReadBytes()

Reads a byte array from the area.

Note

The function allows access to the entire storage area of the virtual controller.

Therefore, use access via the tag name and not via the address areas.

Table 7- 171 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 379). • `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	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 virtual controller is not running.
	`SREC_INDEX_OUT_OF_RANGE`	The offset is outside the area size. No byte could be read.
	`SREC_WRONG_ARGUMENT`	The area is invalid.

Table 7- 172 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	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeError-Code.IndexOutOfRange`	The offset is outside the area size. 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 virtual controller.

Therefore, use access via the tag name (Page 189) and not via the address areas.

Table 7- 173 ReadSignals() - Native C++

Syntax	`ERuntimeErrorCode ReadSignals( EArea in_Area, SDataValueByAddress* inout_Signals, UINT32 in_SignalCount );`
	`ERuntimeErrorCode ReadSignals( EArea in_Area, SDataValueByAddressWithCheck* inout_Signals, UINT32 in_SignalCount, bool* out_SignalsHaveChanged );`	

Parameters	• `EArea in_Area:` The area from which you want to read. Permissible values: `{SRA_INPUT, SRA_MARKER, SRA_OUTPUT}.` See EArea (Page 379). • `SDataValueByAddress* inout_Signals:` The signal list to be read. The result is stored in the structure. • `SDataValueByAddressWithCheck* inout_Signals:` The signal list that is read. The result is stored in the structure. "`Value-HasChanged`" 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 preced-ing call.	
Signal error	Error code	Condition
	`SREC_OK`	The signal operation is successful.
	`SREC_DOES_NOT_EXIST`	The entry does not exist in the stored tag table.
	`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 385).
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 virtual controller is not running.
	`SREC_WRONG_ARGUMENT`	The area is invalid.
	`SREC_SIGNAL_CONFIGURATION_ERROR`	At least one signal error is in the list.

Table 7- 174 ReadSignals() - .NET (C#)

Syntax	`void ReadSignals(` `ref SDataValueByAddress[] inout_Signals` `);` `void ReadSignals(` `ref SDataValueByAddressWithCheck[] inout_Signals` `out bool out_SignalsHaveChanged);` `);`	
Parameters	• `ref SDataValueByAddress[] inout_Signals:` The signal list to be read. • `ref SDataValueByAddressWithCheck[] inout_Signals:` The signal list that is read. The result is stored in the structure. "`Value-HasChanged`" 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	None	
Signal error	Runtime error code	Condition
	`ERuntimeErrorCode.OK`	The signal operation is successful.
	`ERuntimeError-Code.IndexOutOfRange`	Offset or bits are invalid.
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeError-Code.SignalConfigurationError`	At least one signal error is in the list.

7.6.5.3 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 virtual controller.

Therefore, use access via the tag name (Page 220) and not via the address areas.

Table 7- 175 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 379). • `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	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 virtual controller is not running.
	`SREC_INDEX_OUT_OF_RANGE`	Offset or bits are invaid.
	`SREC_WRONG_ARGUMENT`	Area is invalid.

Table 7- 176 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	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeError-Code.IndexOutOfRange`	Offset or bits are invaid.

WriteByte()

Writes an individual byte to the area.

Note

Data can be overwritten

The function allows access to the entire storage area of the virtual controller.

Therefore, use access via the tag name (Page 220) and not via the address areas.

Table 7- 177 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 379). • `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	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 virtual controller is not running.
	`SREC_INDEX_OUT_OF_RANGE`	Offset is invalid.
	`SREC_WRONG_ARGUMENT`	Area is invalid.

Table 7- 178 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	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeError-Code.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 virtual controller.

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 220) and not via the address areas.

Table 7- 179 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 379). • `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	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 virtual controller is not running.
	`SREC_INDEX_OUT_OF_RANGE`	The offset is outside the area size. No byte could be written.
	`SREC_WRONG_ARGUMENT`	The area is invalid.

Table 7- 180 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	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeError-Code.IndexOutOfRange`	The offset is outside the area size. 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 virtual controller.

Therefore, use access via the tag name (Page 220) and not via the address areas.

Table 7- 181 WriteSignals() - Native C++

Syntax	`ERuntimeErrorCode WriteSignals(`
	` EArea in_Area,`
	` SDataValueByAddress* in_Signals,`
	` UINT32 in_SignalCount`
	`);`	

Parameters	• `EArea in_Area:` The area that is to be written. Permissible values: `{SRA_INPUT, SRA_MARKER, SRA_OUTPUT}`. See EArea (Page 379). • `SDataValueByAddress* inout_Signals:` The signal list to be written. • `UINT32 in_SignalCount:` Number of signals in the list.	
Signal error	Error code	Condition
	`SREC_OK`	The signal operation is successful.
	`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 virtual controller is not running.
	`SREC_SIGNAL_CONFIGURATION_ERROR`	At least one signal error is in the list.
	`SREC_WRONG_ARGUMENT`	The area is invalid.

Table 7- 182 WriteSignals() - .NET (C#)

Syntax	`void InputArea.WriteSignals( SDataValueByAddress[] in_Signals );` `void MarkerArea.WriteSignals( SDataValueByAddress[] in_Signals );` `void OutputArea.WriteSignals( SDataValueByAddress[] in_Signals );`	
Parameters	• `SDataValueByAddress[] in_Signals:` The signal list to be written.	
Return values	None	
Signal error	Error code	Condition
	`ERuntimeErrorCode.OK`	The signal operation is successful.
	`ERuntimeError-Code.IndexOutOfRange`	Offset or bits are invalid.
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeError-Code.SignalConfigurationError`	At least one signal error is in the list.

7.6.5.4 I/O access via tag name - Reading

Individual access to IO 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 7- 183 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	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 virtual controller 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 385).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 184 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	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeError-Code.IndexOutOfRange`	The offset lies outside the area range. No value could be read.
	`ERuntimeError-Code.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeError-Code.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeError-Code.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 386).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

ReadBool()

Reads the value of a PLC tag.

Table 7- 185 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	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 virtual controller 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 385).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 186 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	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeError-Code.IndexOutOfRange`	The offset lies outside the area range. No value could be read.
	`ERuntimeError-Code.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeError-Code.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeError-Code.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 385).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

ReadInt8()

Reads the value of a PLC tag.

Table 7- 187 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	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 virtual controller 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 385).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 188 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	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeError-Code.IndexOutOfRange`	The offset lies outside the area range. No value could be read.
	`ERuntimeError-Code.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeError-Code.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeError-Code.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 385).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

ReadInt16()

Reads the value of a PLC tag.

Table 7- 189 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	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 virtual controller 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 385).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 190 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	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeError-Code.IndexOutOfRange`	The offset lies outside the area range. No value could be read.
	`ERuntimeError-Code.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeError-Code.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeError-Code.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 385).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

ReadInt32()

Reads the value of a PLC tag.

Table 7- 191 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	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 virtual controller 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 385).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 192 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	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeError-Code.IndexOutOfRange`	The offset lies outside the area range. No value could be read.
	`ERuntimeError-Code.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeError-Code.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeError-Code.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 385).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

ReadInt64()

Reads the value of a PLC tag.

Table 7- 193 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	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 virtual controller 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.
	`ERuntimeError-Code.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeError-Code.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 385).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 194 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	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeError-Code.IndexOutOfRange`	The offset lies outside the area range. No value could be read.
	`ERuntimeError-Code.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeError-Code.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeError-Code.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 385).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

ReadUInt8()

Reads the value of a PLC tag.

Table 7- 195 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	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 virtual controller 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 385).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 196 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	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeError-Code.IndexOutOfRange`	The offset lies outside the area range. No value could be read.
	`ERuntimeError-Code.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeError-Code.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeError-Code.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 385).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

ReadUInt16()

Reads the value of a PLC tag.

Table 7- 197 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	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 virtual controller 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 385).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 198 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	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeError-Code.IndexOutOfRange`	The offset lies outside the area range. No value could be read.
	`ERuntimeError-Code.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeError-Code.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeError-Code.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 385).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

ReadUInt32()

Reads the value of a PLC tag.

Table 7- 199 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	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 virtual controller 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 385).
	SREC_NOT_UP_TO_DATE	The stored tag list must be updated.

Table 7- 200 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	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeError-Code.IndexOutOfRange`	The offset lies outside the area range. No value could be read.
	`ERuntimeError-Code.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeError-Code.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeError-Code.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 385).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

ReadUInt64()

Reads the value of a PLC tag.

Table 7- 201 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	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 virtual controller 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 385).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 202 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	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeError-Code.IndexOutOfRange`	The offset lies outside the area range. No value could be read.
	`ERuntimeError-Code.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeError-Code.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeError-Code.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 385).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

ReadFloat()

Reads the value of a PLC tag.

Table 7- 203 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	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 virtual controller is not running.
	`ERuntimeError-Code.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 385).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 204 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	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeError-Code.IndexOutOfRange`	The offset lies outside the area range. No value could be read.
	`ERuntimeError-Code.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeError-Code.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeError-Code.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 385).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

ReadDouble()

Reads the value of a PLC tag.

Table 7- 205 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	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 virtual controller is not running.
	`ERuntimeError-Code.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 385).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 206 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	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeError-Code.IndexOutOfRange`	The offset lies outside the area range. No value could be read.
	`ERuntimeError-Code.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeError-Code.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeError-Code.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 385).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

ReadChar()

Reads the value of a PLC tag.

Table 7- 207 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	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 virtual controller is not running.
	`ERuntimeError-Code.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 385).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 208 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	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeError-Code.IndexOutOfRange`	The offset lies outside the area range. No value could be read.
	`ERuntimeError-Code.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeError-Code.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeError-Code.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 385).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

ReadWChar()

Reads the value of a PLC tag.

Table 7- 209 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	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 virtual controller is not running.
	`ERuntimeError-Code.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 385).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 210 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	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeError-Code.IndexOutOfRange`	The offset lies outside the area range. No value could be read.
	`ERuntimeError-Code.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeError-Code.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeError-Code.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 385).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

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 7- 211 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	Condition
	`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 385).
	`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 virtual controller 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 7- 212 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	Condition
	`ERuntimeErrorCode.OK`	The signal operation is successful.
	`ERuntimeError-Code.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeError-Code.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeError-Code.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 385).
	`ERuntimeError-Code.IndexOutOfRange`	Offset or bits are invalid.
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

7.6.5.5 I/O access via tag name - Writing

Individual access to IO 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 7- 213 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 `WriteSig-nals()` function.	
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 virtual controller 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 385).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.
	`SREC_WRONG_ARGUMENT`	The expected type is `UNSPECIFIC`.

Table 7- 214 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.	
Return values	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeError-Code.IndexOutOfRange`	The offset lies outside the area range. No value could be written.
	`ERuntimeError-Code.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeError-Code.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeError-Code.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 385).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.
	`ERuntimeError-Code.WrongArgument`	The expected type is `UNSPECIFIC`.

WriteBool()

Writes the value of a PLC tag.

Table 7- 215 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	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 virtual controller 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 385).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 216 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.	
Return values	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeError-Code.IndexOutOfRange`	The offset lies outside the area range. No value could be written.
	`ERuntimeError-Code.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeError-Code.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeError-Code.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 387).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

WriteInt8()

Writes the value of a PLC tag.

Table 7- 217 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	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 virtual controller 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 385).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 218 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.	
Return values	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeError-Code.IndexOutOfRange`	The offset lies outside the area range. No value could be written.
	`ERuntimeError-Code.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeError-Code.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeError-Code.TypeMissmatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 385).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

WriteInt16()

Writes the value of a PLC tag.

Table 7- 219 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	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 virtual controller 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 385).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 220 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.	
Return values	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeError-Code.IndexOutOfRange`	The offset lies outside the area range. No value could be written.
	`ERuntimeError-Code.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeError-Code.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeError-Code.TypeMissmatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 385).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

WriteInt32()

Writes the value of a PLC tag.

Table 7- 221 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	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 virtual controller 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 385).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 222 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.	
Return values	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeError-Code.IndexOutOfRange`	The offset lies outside the area range. No value could be written.
	`ERuntimeError-Code.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeError-Code.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeError-Code.TypeMissmatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 385).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

WriteInt64()

Writes the value of a PLC tag.

Table 7- 223 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	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 virtual controller 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 385).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 224 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.	
Return values	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeError-Code.IndexOutOfRange`	The offset lies outside the area range. No value could be written.
	`ERuntimeError-Code.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeError-Code.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeError-Code.TypeMissmatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 385).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

WriteInt8()

Writes the value of a PLC tag.

Table 7- 225 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	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 virtual controller 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 385).
	SREC_NOT_UP_TO_DATE	The stored tag list must be updated.

Table 7- 226 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.	
Return values	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeError-Code.IndexOutOfRange`	The offset lies outside the area range. No value could be written.
	`ERuntimeError-Code.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeError-Code.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeError-Code.TypeMissmatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 385).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

WriteUInt16()

Reads the value of a PLC tag.

Table 7- 227 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	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 virtual controller 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 385).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 228 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.	
Return values	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeError-Code.IndexOutOfRange`	The offset lies outside the area range. No value could be written.
	`ERuntimeError-Code.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeError-Code.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeError-Code.TypeMissmatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 385).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

WriteUInt32()

Writes the value of a PLC tag.

Table 7- 229 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	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 virtual controller 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 385).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 230 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.	
Return values	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeError-Code.IndexOutOfRange`	The offset lies outside the area range. No value could be written.
	`ERuntimeError-Code.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeError-Code.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeError-Code.TypeMissmatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 385).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

WriteUInt64()

Writes the value of a PLC tag.

Table 7- 231 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	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 virtual controller 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 385).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 232 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.	
Return values	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeError-Code.IndexOutOfRange`	The offset lies outside the area range. No value could be written.
	`ERuntimeError-Code.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeError-Code.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeError-Code.TypeMissmatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 385).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

WriteFloat()

Writes the value of a PLC tag.

Table 7- 233 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	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 virtual controller 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 385).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 234 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.	
Return values	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeError-Code.IndexOutOfRange`	The offset lies outside the area range. No value could be written.
	`ERuntimeError-Code.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeError-Code.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeError-Code.TypeMissmatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 385).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

WriteDouble()

Writes the value of a PLC tag.

Table 7- 235 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	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 virtual controller 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 385).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 236 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.	
Return values	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeError-Code.IndexOutOfRange`	The offset lies outside the area range. No value could be written.
	`ERuntimeError-Code.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeError-Code.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeError-Code.TypeMissmatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 385).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

WriteChar()

Writes the value of a PLC tag.

Table 7- 237 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	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 virtual controller 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 385).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 238 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.	
Return values	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeError-Code.IndexOutOfRange`	The offset lies outside the area range. No value could be written.
	`ERuntimeError-Code.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeError-Code.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeError-Code.TypeMissmatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 385).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

WriteWChar()

Writes the value of a PLC tag.

Table 7- 239 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	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 virtual controller 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 385).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 240 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.	
Return values	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.
	`SREC_INDEX_OUT_OF_RANGE`	The offset lies outside the area range. No value could be written.
	`ERuntimeError-Code.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeError-Code.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeError-Code.TypeMissmatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 385).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

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 7- 241 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	Condition
	`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 virtual controller 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 385).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.
	`SREC_WRONG_ARGUMENT`	The expected type is `UNSPECIFIC`.

Table 7- 242 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.	
Return values	None	
Signal error	Error code	Condition
	`ERuntimeErrorCode.IndexOutOfRange`	Offset or bits are invalid.
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.InstanceNotRunning`	The process of the virtual controller 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 387).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.
	`ERuntimeErrorCode.WrongArgument`	The expected type is `UNSPECIFIC`.

7.6.6 Settings for the virtual time

GetSystemTime()

Returns the virtual system time of the virtual controller. Returns an empty structure when the function fails.

Table 7- 243 GetSystemTime() - Native C++

Syntax	`SYSTEMTIME GetSystemTime();`
Parameters	None
Return values	`SYSTEMTIME:` System time of the virtual controller.

SetSystemTime()

Sets the virtual system time of the virtual controller. A system time between "Jan 1 1970 00:00:00:000" and "Dec 31 2200 23:59:59:999" is valid.

Table 7- 244 SetSystemTime() - Native C++

Syntax	`ERuntimeErrorCode SetSystemTime( SYSTEMTIME in_SystemTime );`	
Parameters	• `SYSTEMTIME in_SystemTime:` System time that is to be set for the virtual controller.	
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 virtual controller. A system time between "Jan 1 1970 00:00:00:000" and "Dec 31 2200 23:59:59:999" is valid.

Table 7- 245 SystemTime { get; set; } - .NET (C#)

Syntax	`DateTime SystemTime { get; set; }`	
Parameters	None	
Return values	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.WrongArgument`	The value is outside the limits.

GetScaleFactor()

Returns the scaling factor with which the virtual time advances.

Table 7- 246 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.

Start with a small scaling factor and incrementally approach a scaling factor at which the virtual controller 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 virtual controller runs X-times slower than the real time.
- If the value is greater than 1, the virtual time of the virtual controller runs X-times faster than the real time.

A change in the value during runtime only takes effect at the cycle control point.

Table 7- 247 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 virtual controller 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 virtual controller runs X-times slower than the real time.
- If the value is greater than 1, the virtual time of the virtual controller runs X-times faster than the real time.

A change in the value during runtime only takes effect at the cycle control point.

Table 7- 248 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
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.WrongArgument`	The value is outside the limits.

7.6.7 Cycle control

GetOperatingMode()

Returns the operating mode (Page 381) of the virtual controller.

Table 7- 249 GetOperatingMode() - Native C++

Syntax	`EOperatingMode GetOperatingMode();`
Parameters	None
Return values	`EOperatingMode`: Operating mode of the virtual controller

SetOperatingMode()

Sets the operating mode of the virtual controller.

A change in the value during runtime only takes effect at the synchronization point.

Table 7- 250 SetOperatingMode() - Native C++

Syntax	`void SetOperatingMode( EOperatingMode in_OperatingMode );`	
Parameters	• `EOperatingMode in_OperatingMode:` Operating mode of the virtual controller	
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.

OperatingMode { get; set; }

Returns or sets the operating mode of the virtual controller.

A change in the value during runtime only takes effect at the synchronization point.

Table 7- 251 OperatingMode { get; set; } - .NET (C#)

Syntax	`EOperatingMode OperatingMode { get; set; }`	
Parameters	None	
Return values	`EOperatingMode:` Operating mode of the virtual controller	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeError-Code.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 292).

Table 7- 252 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	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.

IsSendSyncEventInDefaultModeEnabled()

Returns the `SendSyncEventInDefaultMode` mode. When the function fails, the return value is `false`.

Table 7- 253 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 292).

Table 7- 254 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	Condition
	`ERuntimeError-Code.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 7- 255 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 7- 256 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	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.

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 7- 257 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	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.WrongArgument`	The value is outside the limits.

RunToNextSyncPoint()

If the virtual controller is running in a SingleStep operating mode, it is stopped at the synchronization point (Freeze state). The `RunToNextSyncPoint()` function cancels the freeze state. The virtual controller continues to run until the next synchronization point.

Table 7- 258 RunToNextSyncPoint() - Native C++

Syntax	`ERuntimeErrorCode RunToNextSyncPoint();`	
Parameters	None	
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 virtual controller is not running.

Table 7- 259 RunToNextSyncPoint() - .NET (C#)

Syntax	`void RunToNextSyncPoint();`	
Parameters	None	
Return values	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is not running.

StartProcessing()

If the virtual controller is running in a TimespanSynchronized operating mode, it is stopped at the synchronization point (Freeze state). The `StartProcessing()` function cancels the freeze state. The virtual controller will now run for at least the requested time before it changes to Freeze state at the next synchronization point.

Table 7- 260 StartProcessing() - Native C++

Syntax	`ERuntimeErrorCode StartProcessing( INT64 in_MinimalTimeToRun_ns );`	
Parameters	• `INT64 in_MinimalTimeToRun_ns:` The minimum virtual time (in nanoseconds) that the virtual controller runs before it changes to Freeze state.	
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 virtual controller is not running.
	`SREC_WRONG_ARGUMENT`	The value is less than 0.

Table 7- 261 StartProcessing() - .NET (C#)

Syntax	`void StartProcessing( Int64 in_MinimalTimeToRun_ns );`	
Parameters	• `Int64 in_MinimalTimeToRun_ns:` The minimum virtual time (in nanoseconds) that the virtual controller runs before it changes to Freeze state.	
Return values	None	
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.InstanceNotRunning`	The process of the virtual controller is not running.
	`ERuntimeErrorCode.WrongArgument`	The value is less than 0.

Additional information

For further information, see sections Virtual time response (Page 75), Stop simulation (Page 78).

SetCycleTimeMonitoringMode()

With this function the source of the timer for the maximum cycle time monitoring can be changed.

Table 7- 262 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 407). – `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	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 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 7- 263 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 407). – `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.	
Return values	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeError-Code.WrongArgument`	The cycle time monitoring mode is inva-lid.
	`ERuntimeError-Code.IndexOutOfRange`	The user-specific value for the maximum cycle time monitoring is outside the lim-its.

GetCycleTimeMonitoringMode()

This function returns information on the source of the timer for the maximum cycle time monitoring.

Table 7- 264 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	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.

Table 7- 265 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.	
Return values	None	
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.

7.6.8 Acyclic services

7.6.8.1 Overview

The acyclic services of 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 user program (TIA Portal), which have logged on for the notification:

Table 7- 266 Events: Read and write operations

SFB	Name	API method (alarm)	API event for triggering the SFB
52	RDREC	ReadRecordDone (Page 263)	OnDataRecordRead (Page 295)
53	WRREC	WriteRecordDone (Page 263)	OnDataRecordWrite (Page 295)

Flowchart

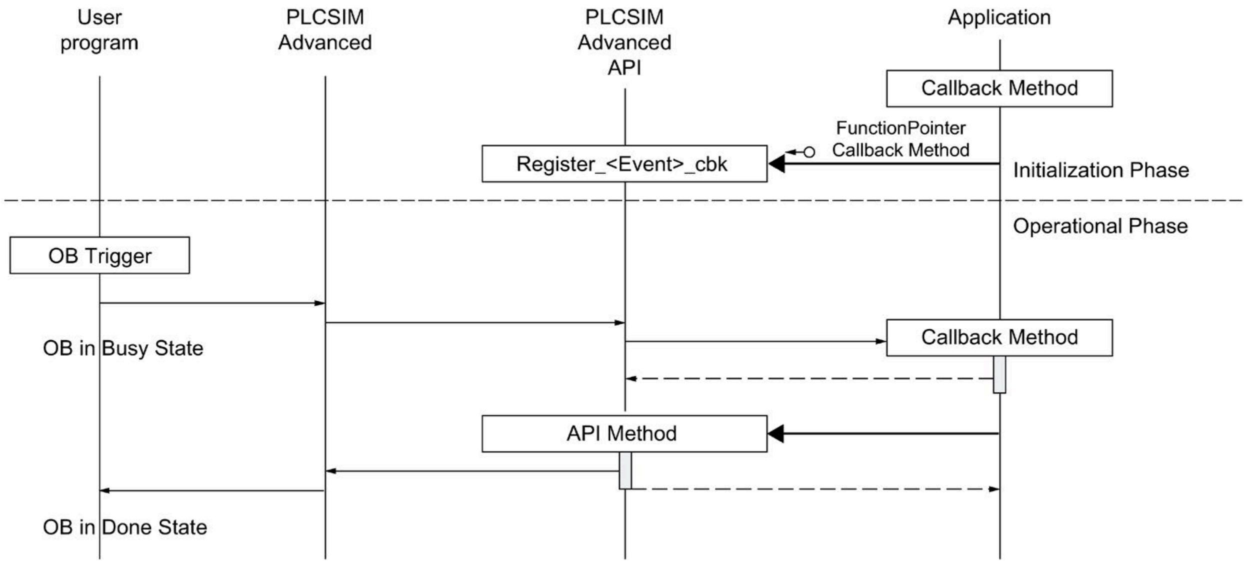


Figure 7-4 Read and write operations flowchart

API methods and associated events

Events which are triggered by the I/O modules and associated API methods:

Table 7- 267 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 266)	OnAlarmNotificationDone (Page 297)
4x	Hardware Interrupt	ProcessEvent (Page 269)	OnProcessEventDone (Page 298)
83	Pull or Plug of module	PullOrPlugEvent (Page 271)	OnPullOrPlugEventDone (Page 299)
55	Status	StatusEvent (Page 273)	OnStatusEventDone (Page 300)
57	Profile	ProfileEvent (Page 275)	OnProfileEventDone (Page 301)
56	Update	UpdateEvent (Page 277)	OnUpdateEventDone (Page 302)
86	Rack or station failure	RackOrStationFaultEvent (Page 279)	OnRackOrStationFaultEventDone (Page 303)

Flowchart

Flowchart for the simulation of events which are triggered by the I/O modules.

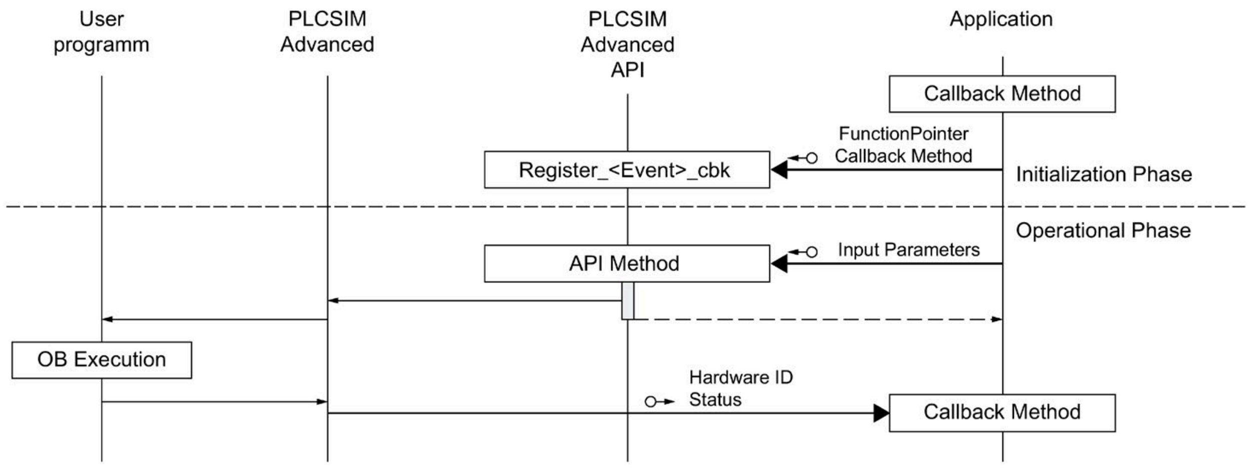


Figure 7-5 Flowchart for the simulation of events

7.6.8.2 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 7- 268 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 368). • `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	Condition
	`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 7- 269 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 368). • `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	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeError-Code.DoesNotExist`	No hardware identifier of the module.
	`ERuntimeError-Code.IndexOutOfRange`	The byte array of the read data record exceeds the length `DataRecord-MaxSize` = 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 7- 270 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 368). • `UINT32 in_Status:` Status of the job execution	
Return values	Runtime error code	Condition
	`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 7- 271 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 368). • `UInt32 in_Status:` Status of the job execution	
Return values	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeError-Code.DoesNotExist`	No hardware identifier of the module.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.

7.6.8.3 AlarmNotification

AlarmNotification()

This function triggers diagnostic alarms according to the PROFINET standard.

Each call of this function calls the OB 82 once, regardless of the number and the severity level of the transferred diagnostic entries.

Table 7- 272 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 zero pointer. For definitions, see SDiagExtChannelDescription (Page 371). • `UINT16* out_SequenceNumber:` 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	Condition
	`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 7- 273 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 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. • `ushort 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 zero pointer. For definitions, see SDiagExtChannelDescription (Page 371). • `Out ushort out_SequenceNumber:` 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.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeError-Code.WrongModuleState`	The module is currently unplugged.
	`ERuntimeError-Code.DoesNotExist`	No hardware identifier of the module.
	`ERuntimeError-Code.WrongArgument`	The value for the module status is invalid.
	`ERuntimeError-Code.WrongModuleType`	The channel number does not exist for the module.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.

Example	`ushort seqNumber;`
	`  var In_ArrayOfDiagnosisEvent = new SDiagExtChannelDescrip-tion[] {`
	`   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 SDi-agExtChannelDescription array above: 4+2+1`

7.6.8.4 ProcessEvent

ProcessEvent()

Process events from central and distributed input modules can be simulated with this function.

Table 7- 274 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 IO module which sends the process event. • `EProcessEventType in_ProcessEventType:` A value from the list of predefined types of events for S7 modules, see EProces-sEventType (Page 394). • `UINT16* out_SequenceNumber:` 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	Condition
	`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 7- 275 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 IO 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 394). • `Out ushort out_SequenceNumber:` 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	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeError-Code.WrongModuleState`	The module is currently unplugged.
	`ERuntimeError-Code.DoesNotExist`	No hardware identifier of the module.
	`ERuntimeError-Code.NotSupportedByModule`	The module is not supported by this user action.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.

7.6.8.5 PullOrPlugEvent

PullOrPlugEvent()

This function triggers pull/plug events. The interrupt OB (OB 83) "Pull or plug of modules" is executed for these events.

Table 7- 276 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 EPullOr-PlugEventType (Page 394). • `UINT16* out_SequenceNumber:` 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	Condition
	`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. For example, if an onboard IO 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 7- 277 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 EPullOr-PlugEventType (Page 394). • `Out ushort out_SequenceNumber` PLCSIM Advanced assigns a unique consecutive number to each interrupt event. According to PROFINET standard the sequence number is 10 bits wide (1 to 7FFH). When the highest number is reached the numbering starts again at 1.	
Return values	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeError-Code.WrongModuleState`	The module is currently unplugged.
	`ERuntimeError-Code.WrongModuleType`	The wrong module type was selected. For example, if an onboard IO of a com-pact CPU is to be pulled.
	`ERuntimeError-Code.DoesNotExist`	No hardware identifier of the module.
	`ERuntimeError-Code.NotSupportedByModule`	The module is not supported by this user action.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.

7.6.8.6 StatusEvent

StatusEvent()

This function is used to trigger the status event OB (OB 55). Status events are only supported for modules in a distributed IO system.

Table 7- 278 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	Condition
	`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 7- 279 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.	
Return values	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeError-Code.WrongModuleState`	The module is currently unplugged.
	`ERuntimeError-Code.DoesNotExist`	No hardware identifier of the module.
	`ERuntimeError-Code.NotSupportedByModule`	The module is not supported by this user action.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.

7.6.8.7 ProfileEvent

ProfileEvent()

This function is used to trigger the Profile event OB (OB 57). Profile events are only supported for modules in a distributed IO system.

Table 7- 280 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	Condition
	`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 7- 281 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.	
Return values	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeError-Code.WrongModuleState`	The module is currently unplugged.
	`ERuntimeError-Code.DoesNotExist`	No hardware identifier of the module.
	`ERuntimeError-Code.NotSupportedByModule`	The module is not supported by this user action.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.

7.6.8.8 UpdateEvent

UpdateEvent()

This function is used to trigger the Update event OB (OB 56). Update events are only supported for modules in a distributed IO system.

Table 7- 282 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	Condition
	`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 7- 283 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.	
Return values	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeError-Code.WrongModuleState`	The module is currently unplugged.
	`ERuntimeError-Code.DoesNotExist`	No hardware identifier of the module.
	`ERuntimeError-Code.NotSupportedByModule`	The module is not supported by this user action.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.

7.6.8.9 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 7- 284 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 SConfiguredProcessEv-ents (Page 369) 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	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.

Table 7- 285 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	Condition
	`ERuntimeErrorCode.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.

7.6.8.10 RackOrStationFaultEvent

Description

This function is used to trigger the RackOrStationFault event OB (OB 86). These events are only supported for distributed devices.

Table 7- 286 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 397).	
Return values	Runtime error code	Condition
	`SREC_OK`	The function is successful.
	`SREC_INTERFACE_REMOVED`	The instance is not registered in Runtime Manager.
	`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 7- 287 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 ERackOrStation-FaultType (Page 397).	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeError-Code.DoesNotExist`	The specified HW identifier of the device does not exist.
	`ERuntimeError-Code.WrongModuleType`	The specified HW identifier is not that of a distributed device.
	`ERuntimeError-Code.WrongModuleState`	The device with the specified HW identifi-er already reports the status Fault/Return.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.

Additional information

You can find additional information on the HW identifier in the STEP 7 online help.

7.6.9 Events for IInstances

7.6.9.1 Events for operating state and cycle control

Events for operating state and cycle control

The following events are triggered for the IInstances interface:

Table 7- 288 Events for IInstances

Event	Cause
OnOperatingStateChanged (Page 281)	The operating state of the virtual controller has changed.
OnLedChanged (Page 285)	The LED display of the virtual controller has changed.
OnConfigurationChanging (Page 287)	The configuration of the virtual controller 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.
OnConfigurationChanged (Page 290)	The configuration of the virtual controller has changed: • After power up from the Virtual SIMATIC Memory Card • At the end of a download • When the IP address changes
OnSyncPointReached (Page 292)	The virtual controller has reached a synchronization point. If the virtual controller is being operated in `Default` mode, the `SendSyncEventInDefaultMode` flag must be set to receive the event. See SendSyncEventInDefaultMode (Page 252).

OnOperatingStateChanged events

OnOperatingStateChanged

Registers or unregisters an event handler method.

Table 7- 289 OnOperatingStateChanged - .NET (C#)

Syntax	`event Delegate_II_EREC_DT_EOS_EOS OnOperatingStateChanged;`
Parameters	None. See Delegate_II_EREC_DT_EOS_EOS (Page 338).
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. Only one callback function can be registered for the event. Registering a new callback function causes the previous callback function to be deleted.

Table 7- 290 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 324).
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 7- 291 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 = Initialize(&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 = Initialize(&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
 …
}
```

UnregisterOnOperatingStateChangedCallback()

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

Table 7- 292 UnregisterOnOperatingStateChangedCallback() - Native C++

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

UnregisterOnOperatingStateChangedEvent()

Unregisters the event object.

Table 7- 293 UnregisterOnOperatingStateChangedEvent() - Native C++

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

Table 7- 294 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 7- 295 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 7- 296 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.

OnLedChanged events

OnLedChanged

Registers or unregisters an event handler method.

Table 7- 297 OnLedChanged - .NET (C#)

Syntax	`event Delegate_II_EREC_DT_ELT_ELM OnLedChanged;`
Parameters	None. See Delegate_II_EREC_DT_ELT_ELM (Page 339).
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. Only one callback function can be registered for the event. Registering a new callback function causes the previous callback function to be deleted.

Table 7- 298 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 328).
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 7- 299 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

UnregisterOnLedChangedCallback()

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

Table 7- 300 UnregisterOnLedChangedCallback() - Native C++

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

UnregisterOnLedChangedEvent()

Unregisters the event object.

Table 7- 301 UnregisterOnLedChangedEvent() - Native C++

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

Table 7- 302 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 7- 303 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 7- 304 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.

OnConfigurationChanging events

OnConfigurationChanging

Registers or unregisters an event handler method.

Table 7- 305 OnConfigurationChanging - .NET (C#)

Syntax	`event Delegate_II_EREC_DT OnConfigurationChanging;`
Parameters	None. See Delegate_II_EREC_DT (Page 337).
Return values	None
Exceptions	None
Note	The event handler method runs in a separate thread.

RegisterOnConfigurationChangingCallback()

When the event occurs, the registered callback function is called. Only one callback function can be registered for the event. Registering a new callback function causes the previous callback function to be deleted.

Table 7- 306 RegisterOnConfigurationChangingCallback() - Native C++

Syntax	`void RegisterOnConfigurationChangingCallback(`
	` 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 (Page 326).
Return values	None
Note	The callback function runs in a separate thread.

RegisterOnConfigurationChangingEvent()

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- 307 RegisterOnConfigurationChangingEvent() - Native C++

Syntax	`void RegisterOnConfigurationChangingEvent();`
	`void RegisterOnConfigurationChangingEvent(`
	` 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

UnregisterOnConfigurationChangingCallback()

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

Table 7- 308 UnregisterOnConfigurationChangingCallback() - Native C++

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

UnregisterOnConfigurationChangingEvent()

Unregisters the event object.

Table 7- 309 UnregisterOnConfigurationChangingEvent() - Native C++

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

Table 7- 310 UnregisterOnConfigurationChangingEvent() - .NET (C#)

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

WaitForOnConfigurationChangingEvent()

The function blocks the program until the registered event object is in the signaled state or the timeout interval is exceeded.

Table 7- 311 WaitForOnConfigurationChangingEvent() - Native C++

Syntax	`bool WaitForOnConfigurationChangingEvent();`
	`bool WaitForOnConfigurationChangingEvent(`
	` 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 7- 312 WaitForOnConfigurationChangingEvent() - .NET (C#)

Syntax	`bool WaitForOnConfigurationChangingEvent();`
	`bool WaitForOnConfigurationChangingEvent(`
	` 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.

OnConfigurationChanged events

OnConfigurationChanged

Registers or unregisters an event handler method.

Table 7- 313 OnConfigurationChanged - .NET (C#)

Syntax	`event Delegate_II_EREC_DT_SRICC_UINT32_UINT32_UINT32_UINT32 OnConfigurationChanged;`
Parameters	None. See Delegate_II_EREC_DT_SRICC_UINT32_UINT32_UINT32_UINT32 (Page 341).
Return values	None
Exceptions	None
Note	The event handler method runs in a separate thread.

RegisterOnConfigurationChangedCallback()

When the event occurs, the registered callback function is called. Only one callback function can be registered for the event. Registering a new callback function causes the previous callback function to be deleted.

Table 7- 314 RegisterOnConfigurationChangedCallback() - Native C++

Syntax	`void RegisterOnConfigurationChangedCallback( EventCallback_II_SREC_ST_SRICC_UINT32_UINT32_UINT32_UINT32 in_CallbackFunction );`
Parameters	• `Event-Callback_II_SREC_ST_SRICC_UINT32_UINT32_UINT32_UINT32 in_CallbackFunction:` A callback function that subscribes to an event. See Event-Callback_II_SREC_ST_SRICC_UINT32_UINT32_UINT32_UINT32 (Page 327).
Return values	None
Note	The callback function runs in a separate thread.

RegisterOnConfigurationChangedEvent()

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 7- 315 RegisterOnConfigurationChangedEvent() - Native C++

Syntax	`void RegisterOnConfigurationChangedEvent();`
	`void RegisterOnConfigurationChangedEvent(`
	` 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

UnregisterOnConfigurationChangedCallback()

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

Table 7- 316 UnregisterOnConfigurationChangedCallback() - Native C++

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

UnregisterOnConfigurationChangedEvent()

Unregisters the event object.

Table 7- 317 UnregisterOnConfigurationChangedEvent() - Native C++

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

Table 7- 318 UnregisterOnConfigurationChangedEvent() - .NET (C#)

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

WaitForOnConfigurationChangedEvent()

The function blocks the program until the registered event object is in the signaled state or the timeout interval is exceeded.

Table 7- 319 WaitForOnConfigurationChangedEvent() - Native C++

Syntax	`bool WaitForOnConfigurationChangedEvent();`
	`bool WaitForOnConfigurationChangedEvent(`
	` 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 7- 320 WaitForOnConfigurationChangedEvent() - .NET (C#)

Syntax	`bool WaitForOnConfigurationChangedEvent();`
	`bool WaitForOnConfigurationChangedEvent(`
	` 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.

OnSyncPointReached events

OnSyncPointReached

Registers or unregisters an event handler method.

Table 7- 321 OnSyncPointReached - .NET (C#)

Syntax	`event Delegate_II_EREC_DT_UINT32_INT64_INT64_UINT32 On-SyncPointReached;`
Parameters	None. See Delegate_II_EREC_DT_UINT32_INT64_INT64_UINT32 (Page 340).
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. Only one callback function can be registered for the event. Registering a new callback function causes the previous callback function to be deleted.

Table 7- 322 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 325).
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 7- 323 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

UnregisterOnSyncPointReachedCallback()

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

Table 7- 324 UnregisterOnSyncPointReachedCallback() - Native C++

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

UnregisterOnSyncPointReachedEvent()

Unregisters the event object.

Table 7- 325 UnregisterOnSyncPointReachedEvent() - Native C++

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

Table 7- 326 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 7- 327 WaitForOnSyncPointReachedEvent() - Native C++

Syntax	`SOnSyncPointReachedResult WaitForOnSyncPointReachedEvent();`
	`SOnSyncPontReachedResult WaitForOnEndOfCycleOnSyncPoin-`
	`tReachedEvent(`
	` 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 366).

Table 7- 328 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 366).

7.6.9.2 Events for acyclic services

OnDataRecordRead / OnDataRecordWrite events

OnDataRecordRead

Registers or unregisters an event handler method.

Table 7- 329 OnDataRecordRead - .NET (C#)

Syntax	`event Delegate_II_EREC_DT_SDRI OnDataRecordRead;`
Parameter	None. See Delegate_II_EREC_DT_SDRI (Page 343).
Return values	None
Exceptions	None
Note	The Event-Handler Methode runs in a separate thread.

OnDataRecordWrite

Registers or unregisters an event handler method.

Table 7- 330 OnDataRecordWrite - .NET (C#)

Syntax	`event Delegate_II_EREC_DT_SDR OnDataRecordWrite;`
Parameter	None. See Delegate_II_EREC_DT_SDR (Page 342).
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. Only one callback function can be registered for the event. Registering a new callback function causes the previous callback function to be deleted.

Table 7- 331 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.
Return values	None
Note	The callback function runs in a separate thread.

UnregisterOnDataRecordReadCallback()

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

Table 7- 332 UnregisterOnDataRecordReadCallback() - Native C++

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

RegisterOnDataRecordWriteCallback()

When the event occurs, the registered callback function is called. Only one callback function can be registered for the event. Registering a new callback function causes the previous callback function to be deleted.

Table 7- 333 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.
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 7- 334 UnregisterOnDataRecordWriteCallback() - Native C++

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

OnAlarmNotification events

OnAlarmNotificationDone()

Registers or unregisters an event handler method.

Table 7- 335 OnAlarmNotificationDone() - .NET (C#)

Syntax	`event Delegate_SREC_ST_UINT32_UINT32 OnAlarmNotifica-tionDone;`
Parameters	None. See Delegate_SREC_ST_UINT32_UINT32 (Page 347).
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. Only one callback function can be registered for the event. Registering a new callback function causes the previous callback function to be deleted.

Table 7- 336 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 331).
Return values	None
Note	The callback function runs in a separate thread.

UnregisterOnAlarmNotificationDoneCallback()

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

Table 7- 337 UnregisterOnAlarmNotificationDoneCallback() - Native C++

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

OnProcessEvent events

OnProcessEventDone()

Registers or unregisters an event handler method.

Table 7- 338 OnProcessEventDone() - .NET (C#)

Syntax	`event Delegate_SREC_ST_UINT32_UINT32_EPET_UINT32 OnProces-sEventDone;`
Parameters	None. See Delegate_SREC_ST_UINT32_UINT32_EPET_UINT32. (Page 345)
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. Only one callback function can be registered for the event. Registering a new callback function causes the previous callback function to be deleted.

Table 7- 339 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 332)
Return values	None
Note	The callback function runs in a separate thread.

UnregisterOnProcessEventDoneCallback()

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

Table 7- 340 UnregisterOnProcessEventDoneCallback() - Native C++

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

OnPullOrPlugEvent events

OnPullOrPlugEventDone()

Registers or unregisters an event handler method.

Table 7- 341 OnPullOrPlugEventDone() - .NET (C#)

Syntax	`event Delegate_SREC_ST_UINT32_EPPET_UINT32 OnPullOr-PlugEventDone;`
Parameters	None. See Delegate_SREC_ST_UINT32_EPPET_UINT32 (Page 344).
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. Only one callback function can be registered for the event. Registering a new callback function causes the previous callback function to be deleted.

Table 7- 342 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 333).
Return values	None
Note	The callback function runs in a separate thread.

UnregisterOnPullOrPlugEventDoneCallback()

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

Table 7- 343 UnregisterOnPullOrPlugEventDoneCallback() - Native C++

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

OnStatusEvent events

OnStatusEventDone()

Registers or unregisters an event handler method.

Table 7- 344 OnStatusEventDone() - .NET (C#)

Syntax	`event Delegate_SREC_ST_UINT32 OnStatusEventDone;`
Parameters	None. See Delegate_SREC_ST_UINT32 (Page 346).
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. Only one callback function can be registered for the event. Registering a new callback function causes the previous callback function to be deleted.

Table 7- 345 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 335).
Return values	None
Note	The callback function runs in a separate thread.

UnregisterOnStatusEventDoneCallback()

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

Table 7- 346 UnregisterOnStatusEventDoneCallback() - Native C++

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

OnProfileEvent events

OnProfileEventDone()

Registers or unregisters an event handler method.

Table 7- 347 OnProfileEventDone() - .NET (C#)

Syntax	`event Delegate_SREC_ST_UINT32 OnProfileEventDone;`
Parameters	None. See Delegate_SREC_ST_UINT32 (Page 346).
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. Only one callback function can be registered for the event. Registering a new callback function causes the previous callback function to be deleted.

Table 7- 348 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 335).
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 7- 349 UnregisterOnProfileEventDoneCallback() - Native C++

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

OnUpdateEvent events

OnUpdateEventDone()

Registers or unregisters an event handler method.

Table 7- 350 OnUpdateEventDone() - .NET (C#)

Syntax	`event Delegate_SREC_ST_UINT32 OnUpdateEventDone;`
Parameters	None. See Delegate_SREC_ST_UINT32 (Page 346).
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. Only one callback function can be registered for the event. Registering a new callback function causes the previous callback function to be deleted.

Table 7- 351 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 335).
Return values	None
Note	The callback function runs in a separate thread.

UnregisterOnUpdateEventDoneCallback()

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

Table 7- 352 UnregisterOnUpdateEventDoneCallback() - Native C++

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

RackOrStationFault events

OnRackOrStationFaultEvent

Registers or unregisters an event handler method.

Table 7- 353 OnRackOrStationFaultEvent - .NET (C#)

Syntax	`event Delegate_SREC_ST_UINT32_ERSFET OnRackOrStationFault;`
Parameter	None. See Delegate_SREC_ST_UINT32_ERSFET (Page 348).
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. Only one callback function can be registered for the event. Registering a new callback function causes the previous callback function to be deleted.

Table 7- 354 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 334)
Return values	None
Note	The callback function runs in a separate thread.

UnregisterOnRackOrStationFaultEventCallback()

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

Table 7- 355 UnregisterOnRackOrStationFaultEventCallback() - Native C++

Syntax	`void UnregisterOnRackOrStationFaultEventCallback ();`
Parameter	None
Return values	None

7.7 API IRemoteRuntimeManager

7.7.1 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 7- 356 Dispose() - .NET (C#)

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

GetVersion()

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

Table 7- 357 GetVersion() - Native C++

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

Table 7- 358 Version { get; } - .NET (C#)

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

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 7- 359 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 7- 360 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.

GetPort() / Port { get; }

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

Table 7- 361 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 7- 362 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.

GetRemoteComputerName() / RemoteComputerName { get; }

Returns the name of the PC on which the remote Runtime Manager is running.

When the name of the PC on which the remote Runtime Manager is running cannot be identified on the local PC, the IP address is returned.

Table 7- 363 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	Condition
	`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 7- 364 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	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The interface is disconnected from the remote Runtime Manager.
	`ERuntimeError-Code.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 7- 365 Disconnect() - Native C++

Syntax	`ERuntimeErrorCode Disconnect();`	
Parameters	None	
Return values	Runtime error code	Condition
	`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 7- 366 Disconnect() - .NET (C#)

Syntax	`void Disconnect();`	
Parameters	None	
Return values	None	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The interface is disconnected from the remote Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.

GetStrictMotionTiming() / StrictMotionTiming { get; }

Returns the current global setting for the "Strict Motion Timing" feature that has an effect on newly created instances.

Table 7- 367 GetStrictMotionTiming() - Native C++

Syntax	`ERuntimeErrorCode GetStrictMotionTiming(bool* enabled);`	
Parameters	bool* enabled: Receives the current setting. `true`: Active `false`: Inactive	
Return values	Runtime error code	Condition
	`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 UserInterfaceConfigura-tion.xml.

Table 7- 368 StrictMotionTiming { get; } - .NET (C#)

Syntax	`bool StrictMotionTiming { get; }`	
Parameters	None	
Return values	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`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.

Table 7- 369 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	Condition
	`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 7- 370 StrictMotionTiming { set; } - .NET (C#)

Syntax	`bool StrictMotionTiming { set; }`	
Parameters	None	
Return values	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`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.

7.7.2 Simulation Runtime instances

7.7.2.1 Simulation Runtime instances (remote)

GetRegisteredInstancesCount()

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

Table 7- 371 GetRegisteredInstancesCount() - Native C++

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

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 7- 372 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. See SInstanceInfo (Page 360).	
Return values	Runtime error code	Condition
	`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 7- 373 RegisterInstanceInfo { get; } - .NET (C#)

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

RegisterInstance()

Registers a new instance of a virtual controller in Runtime Manager. Creates and returns an interface of this instance.

Table 7- 374 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`
	`);`

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`. See Data types (Page 349). • `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	Condition
	`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 or the IInstance pointer is invalid.
	`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 = Initialize(&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 it.

See DestroyInterface() (Page 105).

Table 7- 375 RegisterInstance() - .NET (C#)

Syntax	`IInstance RegisterInstance();`
	`IInstance RegisterInstance(`
	` string in_InstanceName`
	`);`
	`IInstance RegisterInstance(`
	` ECPUType in_CPUType`
	`);`
	`IInstance RegisterInstance(`
	` ECPUType in_CPUType`
	` 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 349).	
Return values	If the function is successful, an interface of a virtual controller. Otherwise, a Null pointer.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeErrorCode.InterfaceRemoved`	The interface is disconnected from the remote Runtime Manager.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.
	`ERuntimeErrorCode.WrongArgument`	The name is invalid.
	`ERuntimeErrorCode.LimitReached`	There are already 16 instances registered in Runtime Manager.
	`ERuntimeErrorCode.AlreadyExists`	An instance with this name already exists.

RegisterCustomInstance()

Registers a new instance of a virtual controller in Runtime Manager. Creates and returns an interface of this instance.

Table 7- 376 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 virtual controller 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 349). • `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	Condition
	`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.
	`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 = Initialize(&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**`("C:\\Temp\\vplc.dll");`
	`}`	

Note

Native C++

If you no longer require the interface, delete it.

See DestroyInterface() (Page 105).

Table 7- 377 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 virtual controller 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 349).	
Return values	If the function is successful, an interface of a virtual controller; otherwise a Null pointer.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`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.

CreateInterface()

Creates and returns an interface of an already registered instance of a virtual controller.

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

Table 7- 378 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	Condition
	`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 = Initialize(&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 = Initialize(&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 it.

See DestroyInterface() (Page 105)

Table 7- 379 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 virtual controller; otherwise a Null pointer.	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`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.

7.7.3 Events for IRemoteRuntimeManager

7.7.3.1 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 7- 380 OnConnectionLost - .NET (C#)

Syntax	`event Delegate_IRRTM OnConnectionLost;`
Parameters	None. See Delegate_IRRTM (Page 341)
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. Only one callback function can be registered for the event. Registering a new callback function causes the previous callback function to be unregistered.

Table 7- 381 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 323).
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 7- 382 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 7- 383 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 7- 384 UnregisterOnConnectionLostCallback() - Native C++

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

UnregisterOnConnectionLostEvent()

Unregisters the event object.

Table 7- 385 UnregisterOnConnectionLostEvent() - Native C++

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

Table 7- 386 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 7- 387 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 7- 388 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.

7.8 Data types

Note

Unsupported data types

The Runtime API does not support the `STRING` and `WSTRING` data types.

Supported data types

In PLCSIM Advanced V4.0, 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 BDC-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.

7.8.1 DLL import functions (Native C++)

7.8.1.1 ApiEntry_Initialize

Description

Type of the central entry point for the API library (DLL).

Table 7- 389 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 initial-ized 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
	`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.

7.8.1.2 ApiEntry_DestroyInterface

Description

Type of the entry point for DestroyInterface (Page 105).

Table 7- 390 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.

7.8.2 Event callback functions (Native C++)

7.8.2.1 EventCallback_VOID

Description

Table 7- 391 EventCallback_VOID - Native C++

Syntax	`typedef void (*EventCallback_VOID)();`
Parameters	None
Return values	None

7.8.2.2 EventCallback_SRCC_UINT32_UINT32_INT32

Description

Table 7- 392 EventCallback_SRCC_UINT32_UINT32_INT32 - Native C++

Syntax	`ERuntimeConfigChanged in_RuntimeConfigChanged,`
	` UINT32 in_Param1,`
	` UINT32 in_Param2,`
	` INT32 in_Param3`
	`);`			

Parameters	`ERuntimeCon-figChanged in_RuntimeCon figChanged`	`UInt32 in_Param1`	`UInt32 in_Param2`	`Int32 in_Param3`
	`SRCC_INSTANCE _REGISTERED`	–	–	ID of the regis-tered instance
	`SRCC_INSTANCE _UNREGISTERED`	–	–	ID of the unregis-tered instance
	`SRCC_CONNECTI ON_OPENED`	IP of the remote Runtime Manager	Port of the remote Runtime Manager	–
	`SRCC_CONNECTI ON_CLOSED`	IP of the remote Runtime Manager	Port of the remote Runtime Manager	–
	`SRCC_PORT_OPE NED`	The open port	–	–
	`SRCC_PORT_CLO SED`	–	–	–
Return values	None			

7.8.2.3 EventCallback_SRRSI_AD

Description

Table 7- 393 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 373). – `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 398).
Return values	None

7.8.2.4 EventCallback_IRRTM

Description

Table 7- 394 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

7.8.2.5 EventCallback_II_SREC_ST_SROS_SROS

Description

Table 7- 395 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 virtual controller 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.

7.8.2.6 EventCallback_II_SREC_ST_UINT32_INT64_INT64_UINT32

Description

Table 7- 396 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_PipId, 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 virtual controller at the time when this event was triggered. • `UINT32 in_PipId:` 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 (Page 82): 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 (Page 82): 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

7.8.2.7 EventCallback_II_SREC_ST

Description

Table 7- 397 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 virtual controller at the time when this event was triggered.
Return values	None

7.8.2.8 EventCallback_II_SREC_ST_SRICC_UINT32_UINT32_UINT32_UINT32

Description

Table 7- 398 EventCallback_II_SREC_ST_SRICC_UINT32_UINT32_UINT32_UINT32 - Native C++

Syntax	`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 );`				
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 virtual controller at the time when this event was triggered.				
	`EInstanceConfigChanged in_InstanceConfigChanged`	`UINT32 in_Param1`	`UINT32 in_Param2`	`UINT32 in_Param3`	`UINT32 in_Param4`
	`SRICC_HARDWARE_SOFTWARE_CHANGED`	-	-	-	-
	`SRICC_IP_CHANGED`	The ID of the interface	The new IP	The new subnet mask	The new standard gateway
Return values	None				

7.8.2.9 EventCallback_II_SREC_ST_SRLT_SRLM

Description

Table 7- 399 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 virtual controller 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

7.8.2.10 EventCallback_II_SREC_ST_SDRI

Description

Table 7- 400 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 virtual controller 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 wants 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

7.8.2.11 EventCallback_II_SREC_ST_SDRI_BYTE

Description

Table 7- 401 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_DataRecordInfo`
	` const 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 virtual controller 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

7.8.2.12 EventCallback_II_SREC_ST_UINT32_UINT32

Description

Table 7- 402 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 virtual controller 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:` 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

7.8.2.13 EventCallback_II_SREC_ST_UINT32_UINT32_EPET_UINT32

Description

Table 7- 403 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 virtual controller at the time when this event was triggered. • `UINT32 in_HardwareIdentifier:` The hardware identifier of the IO module that sends the process event. • `UINT32 in_Channel:` The channel of the IO 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 394). • `UINT32 in_SequenceNumber:` 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

7.8.2.14 EventCallback_II_SREC_ST_UINT32_EPPET_UINT32

Description

Table 7- 404 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 virtual controller 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 394). • `UINT32 in_SequenceNumber:` 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

7.8.2.15 EventCallback_II_SREC_ST_UINT32_ERSFET

Description

Table 7- 405 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 virtual controller 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 397).
Return values	None

7.8.2.16 EventCallback_II_SREC_ST_UINT32

Description

Table 7- 406 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 virtual controller 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

7.8.3 Delegate definitions (managed code)

7.8.3.1 Delegate_Void

Description

Table 7- 407 Delegate_Void - .NET (C#)

Syntax	`delegate void Delegate_Void();`
Parameters	None
Return values	None

7.8.3.2 Delegate_SRCC_UINT32_UINT32_INT32

Description

Table 7- 408 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	`ERuntimeCon-figChanged in_RuntimeConfigChanged`	`UInt32 in_Param1`	`UInt32 in_Param2`	`Int32 in_Param3`
	`InstanceReg-istered`	-	-	ID of the regis-tered instance
	`InstanceUn-registered`	-	-	ID of the unregis-tered instance
	`ConnectionO-pened`	IP of the Remote Runtime Manager	Port of the remote Runtime Manager	-
	`Connection-Closed`	IP of the Remote Runtime Manager	Port of the remote Runtime Manager	-
	`PortOpened`	The open port	-	-
	`PortClosed`	-	-	-
Return values	None			

7.8.3.3 Delegate_SRRSI_AD

Description

Table 7- 409 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 398). – `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 373).
Return values	None

7.8.3.4 Delegate_II_EREC_DT

Description

Table 7- 410 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 virtual controller at the time when this event was triggered.
Return values	None

7.8.3.5 Delegate_II_EREC_DT_EOS_EOS

Description

Table 7- 411 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 virtual controller 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.
	`ERuntimeError-Code.WarningTrialModeActive`	No license available. You can use the instance without restrictions with the Trial License. Afterwards, the instance is shut down.
	`ERuntimeError-Code.LicenseNotFound`	Test mode has expired.
	`ERuntimeError-Code.CommunicationInterfaceNotAvailable`	A problem has occurred with the selected communication interface. Check your settings.

7.8.3.6 Delegate_II_EREC_DT_ELT_ELM

Description

Table 7- 412 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 virtual controller 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

7.8.3.7 Delegate_II_EREC_DT_UINT32_INT64_INT64_UINT32

Description

Table 7- 413 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_PipId, 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 virtual controller at the time when this event was triggered. • `UInt32 in_PipId:` 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. Or the process time for the time-controlled operating modes (Page 82). • `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 (Page 82). • `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

7.8.3.8 Delegate_IRRTM

Description

Table 7- 414 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

7.8.3.9 Delegate_II_EREC_DT_SRICC_UINT32_UINT32_UINT32_UINT32

Description

Table 7- 415 Delegate_II_EREC_DT_SRICC_UINT32_UINT32_UINT32_UINT32 - .NET (C#)

Syntax	`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`
	`);`				

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 virtual controller at the time when this event was triggered.				
	`EInstanceConfigChanged in_InstanceConfigChanged`	`UInt32 in_Param1`	`UInt32 in_Param2`	`UInt32 in_Param3`	`UInt32 in_Param4`
	`HardwareSoftwareChanged`	-	-	-	-
	`IPChanged`	The ID of the interface	The new IP	The new subnet mask	The new standard gateway
Return values	None				

7.8.3.10 Delegate_II_EREC_DT_SDRI

Description

Table 7- 416 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 virtual controller 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

7.8.3.11 Delegate_II_EREC_DT_SDR

Description

Table 7- 417 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 virtual controller 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 wants to write the data record – The index of the supplied data record – Size of data record – The data record
Return values	None

7.8.3.12 Delegate_SREC_ST_UINT32_EPPET_UINT32

Description

Table 7- 418 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 virtual controller 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 394). • `UInt32 in_SequenceNumber:` 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

7.8.3.13 Delegate_SREC_ST_UINT32_UINT32_EPET_UINT32

Description

Table 7- 419 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 virtual controller at the time when this event was triggered. • `UInt32 in_HardwareIdentifier:` The hardware identifier of the module or submodule that sends the process event. • `UInt32 in_Channel:` The channel of the IO 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 394). • `UInt32 in_SequenceNumber:` 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

7.8.3.14 Delegate_SREC_ST_UINT32

Description

Table 7- 420 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 virtual controller at the time when this event was triggered. • `UInt32 in_HardwareIdentifier:` The ID of the module which generates the status event, update event or profile event.
Return values	None

7.8.3.15 Delegate_SREC_ST_UINT32_UINT32

Description

Table 7- 421 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 virtual controller 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:` 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

7.8.3.16 Delegate_SREC_ST_UINT32_ERSFET

Description

Table 7- 422 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 virtual controller 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. See ERackOrStationFaultType (Page 397).
Return values	None

7.8.4 Definitions and constants

The following identifiers are used in the API:

Table 7- 423 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 ZERO termination.
`DTAG_NAME_MAX_LENGTH`	300	The maximum length of the name of a PLC tag. Including ZERO termination.
`DTAG_ARRAY_DIMENSION`	6	The maximum number of dimension for a multi-dimensional field.
`DCONTROLLER_NAME_MAX_LENGTH`	128	The maximum length of the controller name. Including ZERO termination.
`DCONTROLLER_SHORT_DESIGNATION_MAX_LENGTH`	32	The maximum length of the abbreviation of the controller (CPU type). Including ZERO 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
`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 7- 424 Constants - .NET (C#)

Identifier	Value	Description
`RuntimeConstants.InstanceNameLength`	64	The unique name of an instance must be less than this value.
`RuntimeConstants.StoragePathMaxLength`	130	The maximum path length to the virtual memory card. Including ZERO termination.
`RuntimeConstants.TagNameMaxLength`	300	The maximum length of the name of a PLC tag. Including ZERO termination.
`RuntimeConstants.TagArrayDimension`	6	The maximum number of dimension for a multi-dimensional field.
`RuntimeConstants.ControllerNameMaxLength`	128	The maximum length of the controller name. Including ZERO termination.
`RuntimeConstants.ControllerShortDesignationMaxLength`	32	The maximum length of the abbreviation of the controller (CPU type). Including ZERO 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

7.8.5 Unions (Native C++)

7.8.5.1 UIP

Description

Contains an IPv4 address.

Table 7- 425 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

7.8.5.2 UDataValue

Description

Contains the value of a PLC tag.

Table 7- 426 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

7.8.6 Structures

The following structures are available:

- SDataValue (Page 353)
- SDVBNI (Page 356)
- SDataValueByAddress (Page 356)
- SDataValueByAddressWithCheck (Page 357)
- SDataValueByName (Page 358)
- SDataValueByNameWithCheck (Page 359)
- SConnectionInfo (Page 359)
- SInstanceInfo (Page 360)
- SDimension (Page 360)
- STagInfo (Page 361)
- SIP (Page 364)
- SIPSuite4 (Page 364)
- SOnSyncPointReachedResult (Page 366)
- SDataRecordInfo (Page 368)
- SDataRecord (Page 369)
- SConfiguredProcessEvents (Page 369)
- SDiagExtChannelDescription (Page 371)
- SAutodiscoverData (Page 373)

7.8.6.1 SDataValue

Description

The structure contains the value and type of a PLC tag.

Table 7- 427 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 7- 428 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

7.8.6.2 SDVBNI

Description

This structure is for internal use only. Do not change this structure.

Table 7- 429 SDVBNI - Native C++

Syntax	`struct SDVBNI`

Table 7- 430 SDVBNI - .NET (C#)

Syntax	`struct SDVBNI`

7.8.6.3 SDataValueByAddress

Description

This structure represents a PLC tag that is accessed via its address.

Table 7- 431 SDataValueByAddress - Native C++

Syntax	`struct SDataValueByAddress`
	`{`
	` UINT32 Offset;`
	` UINT8 Bit;`
	` SDataValue DataValue;`
	` ERuntimeErrorCode ErrorCode;`
	`};`

Table 7- 432 SDataValueByAddress - .NET (C#)

Syntax	`struct SDataValueByAddress`
	`{`
	` UInt32 Offset;`
	` UInt8 Bit;`
	` SDataValue DataValue;`
	` ERuntimeErrorCode ErrorCode;`
	`}`

7.8.6.4 SDataValueByAddressWithCheck

Description

This structure represents a PLC tag that is accessed via its address.

Table 7- 433 SDataValueByAddressWithCheck - Native C++

Syntax	`struct SDataValueByAddressWithCheck`
	`{`
	` UINT32 Offset;`
	` UINT8 Bit;`
	` SDataValue DataValue;`
	` ERuntimeErrorCode ErrorCode;`
	` bool ValueHasChanged;`
	`};`

Table 7- 434 SDataValueByAddressWithCheck - .NET (C#)

Syntax	`struct SDataValueByAddressWithCheck`
	`{`
	` UInt32 Offset;`
	` UInt8 Bit;`
	` SDataValue DataValue;`
	` ERuntimeErrorCode ErrorCode;`
	` bool ValueHasChanged;`
	`}`

7.8.6.5 SDataValueByName

Description

This structure represents a PLC tag that is called by its name.

Table 7- 435 SDataValueByName - Native C++

Syntax	`struct SDataValueByName`
	`{`
	` WCHAR Name[DTAG_NAME_MAX_LENGTH];`
	` SDataValue DataValue;`
	` ERuntimeErrorCode ErrorCode;`
	` SDVBNI Internal;`
	`};`

Table 7- 436 SDataValueByName - .NET (C#)

Syntax	`struct SDataValueByName`
	`{`
	` String Name;`
	` SDataValue DataValue;`
	` ERuntimeErrorCode ErrorCode;`
	` SDVBNI Internal;`
	`}`

7.8.6.6 SDataValueByNameWithCheck

Description

This structure represents a PLC tag that is called by its name.

Table 7- 437 SDataValueByNameWithCheck - Native C++

Syntax	`struct SDataValueByNameWithCheck`
	`{`
	` WCHAR Name[DTAG_NAME_MAX_LENGTH];`
	` SDataValue DataValue;`
	` ERuntimeErrorCode ErrorCode;`
	` SDVBNI Internal;`
	` bool ValueHasChanged;`
	`};`

Table 7- 438 SDataValueByNameWithCheck - .NET (C#)

Syntax	`struct SDataValueByNameWithCheck`
	`{`
	` String Name;`
	` SDataValue DataValue;`
	` ERuntimeErrorCode ErrorCode;`
	` SDVBNI Internal;`
	` bool ValueHasChanged;`
	`}`

7.8.6.7 SConnectionInfo

Description

This structure contains the IP address and port of a TCP/IP connection.

Table 7- 439 SConnectionInfo - Native C++

Syntax	`struct SConnectionInfo`
	`{`
	` UIP IP;`
	` UINT16 Port;`
	`};`

Table 7- 440 SConnectionInfo - .NET (C#)

Syntax	`struct SConnectionInfo`
	`{`
	` SIP IP;`
	` UInt16 Port;`
	`}`

7.8.6.8 SInstanceInfo

Description

This structure contains an IPv4 address.

Table 7- 441 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 7- 442 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

7.8.6.9 SDimension

Description

This structure contains information about the dimension of a field.

Table 7- 443 SDimension - Native C++

Syntax	`struct SDimension`
	`{`
	` INT32 StartIndex;`
	` UINT32 Count;`
	`};`

Table 7- 444 SDimension - .NET (C#)

Syntax	`struct SDimension`
	`{`
	` Int32 StartIndex;`
	` UInt32 Count;`
	`}`

7.8.6.10 STagInfo

Description

This structure contains information about a PLC tag.

Table 7- 445 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 7- 446 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.

7.8.6.11 SIP

Description

This structure contains an IPv4 address.

Table 7- 447 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"

7.8.6.12 SIPSuite4

Description

This structure contains an IPv4 suite.

Table 7- 448 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 7- 449 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

7.8.6.13 SOnSyncPointReachedResult

Description

This structure contains the results of the OnSyncPointReached event.

Table 7- 450 SOnSyncPointReachedResult - Native C++

Syntax	`struct SOnSyncPointReachedResult`
	`{`
	` ERuntimeErrorCode ErrorCode;`
	` SYSTEMTIME SystemTime;`
	` UINT32 PipId;`
	` INT64 TimeSinceSameSyncPoint_ns;`
	` INT64 TimeSinceAnySyncPoint_ns;`
	` UINT32 SyncPointCount;`
	`};`

Member	• `ERuntimeErrorCode ErrorCode:` – S`REC_TIMEOUT`, if no event was triggered during the defined time interval. – `SREC_WARNING_INVALID_CALL`, if no function `Register-OnSyncPointReachedEvent` was called before. See ERuntimeErrorCode (Page 375). • `SYSTEMTIME SystemTime:` The virtual system time of the virtual controller at the time when this event was triggered. • `UINT32 PipId:` 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 (Page 82): 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 (Page 82): 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 7- 451 SOnSyncPointReachedResult - .NET (C#)

Syntax	`struct SOnSyncPointReachedResult`
	`{`
	` ERuntimeErrorCode ErrorCode;`
	` DateTime SystemTime;`
	` UInt32 PipId;`
	` 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 `RegisterOnSyncPoin-tReachedEvent` was called before. See ERuntimeErrorCode. • `DateTime DateTime:` The virtual system time of the virtual controller at the time when this event was triggered. • `UInt32 PipId:` 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.

7.8.6.14 SDataRecordInfo

Description

This structure contains read/write data record information.

Table 7- 452 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 7- 453 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

7.8.6.15 SDataRecord

Description

This structure contains read/write data record information and data records.

Table 7- 454 SDataRecord - .NET (C#)

Syntax	`struct SDataRecord`
	`{`
	`UInt32 HardwareId;`
	`byte[] Data`
	`}`

Member	• `SDataRecordInfo Info:` The data record information, see SDataRecordInfo (Page 368) • `byte[] Data:` The array length

7.8.6.16 SConfiguredProcessEvents

Description

This structure contains information about the configured process events.

Table 7- 455 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 IO 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 7- 456 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 IO module which generates the process event. • `EProcessEventType ProcessEventType:` The type of the configured process event • `String name:` The name of the process event

7.8.6.17 SDiagExtChannelDescription

Description

This structure contains read/write data record information and data records.

Table 7- 457 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 396). • `EDiagProperty Direction:` The value for the incoming/outgoing information, see EDiagProperty (Page 396). • `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 7- 458 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 396). • `EDiagProperty Direction:` The value for the incoming/outgoing information, see EDiagProperty (Page 396). • `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 7- 459 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 7- 460 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

7.8.6.18 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 7- 461 SAutodiscoverData - Native C++

Syntax	`public struct SAutodiscoverData`
	`{`
	` UIP IP;`
	` UINT16 Port;`
	` DWORD RuntimeVersion;`
	` WCHAR ComputerName[MAX_COMPUTERNAME_LENGTH + 1];`
	`};`

Table 7- 462 SAutodiscoverData - .NET (C#)

Syntax	`public struct SAutodiscoverData`
	`{`
	` public SIP IP;`
	` public ushort Port;`
	` public uint RuntimeVersion;`
	` public string ComputerName;`
	`}`

7.8.7 Enumerations

The following enumerations are available:

- ERuntimeErrorCode (Page 375)
- EArea (Page 379)
- EOperatingState (Page 380)
- EOperatingMode (Page 381)
- ECPUType (Page 382)
- ECommunicationInterface (Page 384)
- ELEDType (Page 384)
- ELEDMode (Page 385)
- EPrimitiveDataType (Page 385)
- EDataType (Page 388)
- ETagListDetails (Page 392)
- ERuntimeConfigChanged (Page 393)
- EInstanceConfigChanged (Page 393)
- EPullOrPlugEventType (Page 394)
- EProcessEventType (Page 394)
- EDirection (Page 395)
- EDiagProperty (Page 396)
- EDiagSeverity (Page 396)
- ERackOrStationFaultType (Page 397)
- ECycleTimeMonitoringMode (Page 397)
- EAutodiscoverType (Page 398)

See also

Global functions (Native C++) (Page 109)

7.8.7.1 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 7- 463 ERuntimeErrorCode - Native C++

Syntax	Code

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,
	```

Syntax	
	```
	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
	};
	```

Table 7- 464 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,`

Syntax	`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,`
	` 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`
	`}`

7.8.7.2 EArea

Description

This enumeration contains all PLC areas that contain the available PLC tags.

Table 7- 465 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 7- 466 EArea - .NET (C#)

Syntax	`public enum EArea`
	`{`
	` InvalidArea = 0,`
	` Input = 1,`
	` Marker = 2,`
	` Output = 3,`
	` Counter = 4,`
	` Timer = 5,`
	` DataBlock = 6,`
	`}`

7.8.7.3 EOperatingState

Description

This enumeration contains all the operating states of a virtual controller.

Table 7- 467 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 7- 468 EOperatingState - .NET (C#)

Syntax	`enum EOperatingState`
	`{`
	` InvalidOperatingState = 0,`
	` Off = 1,`
	` Booting = 2,`
	` Stop = 3,`
	` Startup = 4,`
	` Run = 5,`
	` Freeze = 6,`
	` ShuttingDown = 7,`
	` Hold = 8`
	`}`

7.8.7.4 EOperatingMode

Description

This enumeration contains all the operating modes of a virtual controller.

Table 7- 469 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 7- 470 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`
	`}`

7.8.7.5 ECPUType

Description

This enumeration contains all CPU types that can be loaded in a virtual controller.

Table 7- 471 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_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
}
```

Table 7- 472 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,
	 CPU1500_RH_Unspecified = 0x008005DC,
	 CPU1513R = 0x008005E9,
	 CPU1515R = 0x008005EB,
	 CPU1517H = 0x008005ED,
	 CPU1518HF = 0x008105EE,
	 CPUET200PRO_Unspecified = 0x002005DC,
	 CPU1513PRO = 0x002005E9,
	 CPU1513PROF = 0x002105E9,
	 CPU1516PRO = 0x002005EC,
	 CPU1516PROF = 0x002105EC
	}

7.8.7.6 ECommunicationInterface

Description

This enumeration contains the available communication interfaces of a virtual controller.

Table 7- 473 ECommunicationInterface - Native C++

Syntax	`enum ECommunicationInterface`
	`{`
	` SRCI_NONE = 0,`
	` SRCI_SOFTBUS = 1,`
	` SRCI_TCPIP = 2,`
	` SRCI_ENUMERATION_SIZE = 3`
	`};`

Table 7- 474 ECommunicationInterface - .NET (C#)

Syntax	`enum ECommunicationInterface`
	`{`
	` None = 0,`
	` Softbus = 1,`
	` TCPIP = 2,`
	`}`

7.8.7.7 ELEDType

Description

This list includes all types of LEDs of a virtual controller.

Table 7- 475 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 7- 476 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`
	`}`

7.8.7.8 ELEDMode

Description

This list contains all the LED states of a virtual controller.

Table 7- 477 ELEDMode - Native C++

Syntax	`enum ELEDMode`
	`{`
	` SRLM_OFF = 0,`
	` SRLM_ON = 1,`
	` SRLM_FLASH_FAST = 2,`
	` SRLM_FLASH_SLOW = 3,`
	` SRLM_INVALID = 4`
	`};`

Table 7- 478 ELEDMode - .NET (C#)

Syntax	`enum ELEDMode`
	`{`
	` Off = 0,`
	` On = 1,`
	` FlashFast = 2,`
	` FlashSlow = 3,`
	` Invalid = 4`
	`}`

7.8.7.9 EPrimitiveDataType

Description

This list contains all the primitive data types that are used by the I/O access functions.

Table 7- 479 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 7- 480 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 }`

```
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 PLCSIM Advanced instance that are configured in the stored tag list. The data types that can be used as compatible are marked with "X".

Table 7- 481 Compatible primitive data types - Reading

API	**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 7- 482 Compatible primitive data types - Write

API	**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

7.8.7.10 EDataType

Description

This enumeration contains all the PLC data types (STEP 7).

Table 7- 483 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,
	```

Syntax	
	```
	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 7- 484 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,
	```

Syntax	`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`
	`}`

7.8.7.11 ETagListDetails

Description

This list contains all PLC areas that can be used as a filter to update the tag table.

Table 7- 485 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 7- 486 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`
	`}`

7.8.7.12 ERuntimeConfigChanged

Description

This list contains all possible causes of a OnConfigurationChanged event that the Runtime Manager sends.

Table 7- 487 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 7- 488 ERuntimeConfigChanged - .NET (C#)

Syntax	`enum ERuntimeConfigChanged`
	`{`
	` InstanceRegistered = 0,`
	` InstanceUnregistered = 1,`
	` ConnectionOpened = 2,`
	` ConnectionClosed = 3,`
	` PortOpened = 4,`
	` PortClosed = 5`
	`}`

7.8.7.13 EInstanceConfigChanged

Description

This list contains all possible causes for a OnConfigurationChanged event that the virtual controller sends.

Table 7- 489 EInstanceConfigChanged - Native C++

Syntax	`enum EInstanceConfigChanged`
	`{`
	` SRICC_HARDWARE_SOFTWARE_CHANGED = 0,`
	` SRICC_IP_CHANGED = 1`
	`};`

Table 7- 490 EInstanceConfigChanged - .NET (C#)

Syntax	`enum EInstanceConfigChanged`
	`{`
	` HardwareSoftwareChanged = 0,`
	` IPChanged = 1`
	`}`

7.8.7.14 EPullOrPlugEventType

Description

This enumeration contains predefined types of pull/plug events for S7 modules.

Table 7- 491 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 7- 492 EPullOrPlugEventType - .NET (C#)

Syntax	`enum EPullOrPlugEventType`
	`{`
	` Undefined = 0,`
	` Pull = 1,`
	` Plug = 2,`
	` PlugErrorRemains = 3,`
	` PlugWrongModule = 4`
	`}`

7.8.7.15 EProcessEventType

Description

This enumeration contains predefined types of process events for S7 modules.

Table 7- 493 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 7- 494 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`
	`}`

7.8.7.16 EDirection

Description

This enumeration contains properties of the diagnostic alarm.

Table 7- 495 EDirection - Native C++

Syntax	`enum EDirection`
	`{`
	` SRD_DIRECTION_INPUT = 0,`
	` SRD_DIRECTION_OUTPUT = 1`
	`};`

Table 7- 496 EDirection - .NET (C#)

Syntax	`enum EDirection`
	`{`
	` Input = 0,`
	` Output = 1`
	`}`

7.8.7.17 EDiagProperty

Description

This enumeration contains the incoming/outgoing information of the diagnostic alarm.

Table 7- 497 EDiagProperty - Native C++

Syntax	`enum EDiagProperty`
	`{`
	` SRP_DIAG_APPEAR = 1,`
	` SRP_DIAG_DISAPPEAR = 2`
	`};`

Table 7- 498 EDiagProperty - .NET (C#)

Syntax	`enum EDiagProperty`
	`{`
	` Appear = 1,`
	` Disappear = 2`
	`}`

7.8.7.18 EDiagSeverity

Description

This enumeration contains the severity of the diagnostic alarm (error, maintenance demanded, maintenance required).

Table 7- 499 EDiagSeverity - Native C++

Syntax	`enum EDiagSeverity`
	`{`
	` SRDS_SEVERITY_FAILURE = 0,`
	` SRDS_SEVERITY_MAINTENANCE_DEMANDED = 1,`
	` SRDS_SEVERITY_MAINTENANCE_REQUIRED = 2`
	`};`

Table 7- 500 EDiagSeverity - .NET (C#)

Syntax	`enum EDiagSeverity`
	`{`
	` Failure = 0,`
	` MaintDemanded = 1,`
	` MaintRequired = 2`
	`}`

7.8.7.19 ERackOrStationFaultType

Description

This enumeration contains the types of the RackOrStationFault event.

Table 7- 501 ERackOrStationFaultType - Native C++

Syntax	`enum ERackOrStationFaultType`
	`{`
	` SR_RSF_FAULT = 0,`
	` SR_RSF_RETURN = 1`
	`};`

Table 7- 502 ERackOrStationFaultType - .NET (C#)

Syntax	`enum ERackOrStationFaultType`
	`{`
	` Fault = 0,`
	` Return = 1`
	`}`

7.8.7.20 ECycleTimeMonitoringMode

Description

This enumeration contains the sources of the timer for the maximum cycle time monitoring.

Table 7- 503 ECycleTimeMonitoringMode - Native C++

Syntax	`enum ECycleTimeMonitoringMode`
	`{`
	` SRCTMM_DOWNLOADED = 0,`
	` SRCTMM_IGNORED = 1,`
	` SRCTMM_SPECIFIED = 2`
	`};`

Table 7- 504 ECycleTimeMonitoringMode - .NET (C#)

Syntax	`enum ECycleTimeMonitoringMode`
	`{`
	` Downloaded = 0,`
	` Ignored = 1,`
	` Specified = 2`
	`}`

7.8.7.21 EAutodiscoverType

Description

This enumeration is used in the Autodiscover Callback function.

Table 7- 505 EAutodiscoverType - Native C++

Syntax	`enum EAutodiscoverType`
	`{`
	` SRRSI_DISCOVER_STARTED = 0,`
	` SRRSI_DISCOVER_DATA = 1,`
	` SRRSI_DISCOVER_FINISHED = 2`
	`};`

Table 7- 506 EAutodiscoverType - .NET (C#)

Syntax	`public enum EAutodiscoverType`
	`{`
	` AutodiscoverStarted = 0,`
	` AutodiscoverData = 1,`
	` AutodiscoverFinished = 2`
	`}`

Restrictions, messages and solution 8

8.1 Overview

Certain actions or events can lead to behavior in PLCSIM Advanced or in STEP 7 which deviates from that of a hardware CPU. Messages and possible solutions can be found in the following sections:

- OPC UA server (Page 400)
- Web server (Page 400)
- Backing up and restoring the configuration of a PLCSIM Advanced instance (Page 401)
- Restrictions for file paths (Page 402)
- Restrictions for communications services (Page 402)
- Restrictions for instructions (Page 402)
- Restrictions to local communication via Softbus (Page 403)
- Messages for communication via TCP/IP (Page 404)
- Restrictions of security with VMware vSphere Hypervisor (ESXi) (Page 405)
- Restrictions for Hyper-V (Page 407)
- Monitoring overflow (Page 407)
- Deviating I/O values in the STEP 7 user program (Page 408)
- Multiple simulations and possible collision of IP addresses (Page 408)
- Lacking access to an IP address (Page 408)
- Simulation in standby mode (Page 408)
- ET 200SP CPUs: Use of BusAdapters with fiber-optic interface (Page 409)
- Installing SIMATIC NET (Page 409)

8.2 OPC UA server

With OPC UA, data exchange is performed through an open, standardized and manufacturer-independent communication protocol. The CPU as OPC UA server can communicate with OPC UA clients, for example, with HMI panels and SCADA systems.

Configuring OPC UA server

Start the instances via the communication interface "PLCSIM Virtual Ethernet Adapter" (TCP/IP) to use the OPC UA server.

The OPC UA server functionality is not available if communication takes place via the Softbus.

User authorization for OPC UA

The PLCSIM Advanced license also contains the user authorization for OPC UA.

The user authorization applies for two instances.

8.3 Web server

The Web server integrated in a CPU enables monitoring and administering of the CPU by authorized users over a network. This permits evaluation and diagnostics over long distances.

Each PLCSIM Advanced instance can simulate its own Web server.

The simulation of the Web server is restricted under S7-PLCSIM Advanced V4.0:

- It is not possible to back up and restore a configuration via the Web server.
- The freeze state of a virtual controller is not shown as an internal operating state.

Configuring the Web server

S7 PLCSIM Advanced

Start the instances via the communication interface "PLCSIM Virtual Ethernet Adapter" (TCP/IP) to use the Web server.

The Web server functionality is not available if the communication is performed via the Softbus.

STEP 7

Configure the Web server in STEP 7 in the CPU properties.

Restricted Web server functionality

- The information may not be fully displayed on some websites due to different data handling.
- There is no topology information.
- "Online Backup&Restore" is not available.
- FW updates are not supported.

8.4 Backing up and restoring the configuration of a PLCSIM Advanced instance

Backing up and restoring the configuration

As of PLCSIM Advanced V2.0 it is possible to back up and restore a PLCSIM Advanced **instance**.

You can create as many backups as you want and store a variety of configurations for a PLCSIM Advanced instance.

You perform the backup and restore in the TIA Portal as you would in a real CPU.

Backup and restore via the Web server and (simulated) display are not supported.

A backup that was created with PLCSIM Advanced can only be used with PLCSIM Advanced.

It is not possible to restore the configuration of a real CPU with a backup from PLCSIM Advanced.

Requirements

- The configuration of a PLCSIM Advanced instance is backed up and restored over the TCP/IP protocol, Softbus is not supported.
- It is only possible to restore the configuration of a PLCSIM Advanced instance with the corresponding backup from PLCSIM Advanced.

8.5 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 directories such as the Windows directory (%Windows%) or the program directories (%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`.

8.6 Restrictions for communications services

TUSEND / TURCV

When you run the UDP blocks TUSEND and TURCV via the "PLCSIM" communication interface (Softbus), you get error code 0x80C4 at the transmission end and receiving end:

Temporary communications error. The specified connection is temporarily down.

Solution

Set "PLCSIM Virtual Ethernet Adapter" (TCP/IP) as the communication interface in PLCSIM Advanced.

8.7 Restrictions for instructions

PLCSIM Advanced simulates instructions for CPUs S7-1500 and ET 200SP as close to reality as possible. PLCSIM Advanced checks the input parameters for validity and returns outputs that are valid but do not necessarily correspond to those that a real CPU with physical inputs/outputs would return.

Instructions not supported

Unsupported instructions are handled as not ready, their value is always "OK". PLCSIM Advanced does not support the following instructions:

- DP_TOPOL
- PORT_CFG

8.8 Restrictions to local communication via Softbus

Identical IP addresses for instances

If the "PLCSIM" communication interface (Softbus) is set, then identical IP addresses are created automatically for all instances when creating the instances through the Control Panel.

In STEP 7, only one instance is therefore displayed in the lifelist.

Solution

Use the API function `SetIPSuite()` to assign a unique address for each instance, then all instances are displayed in STEP 7 with their IP addresses.

API function

- SetIPSuite() (Page 147)

Working with multiple instances

When you are working with instances **without unique IP addresses**, note the following procedure for downloading from TIA Portal via "PLCSIM" (Softbus):

1. Start **only one** instance with the symbol ⏻ in the Control Panel.
2. In TIA Portal, download the program to this instance.
3. Repeat the steps until you have created all instances and downloaded all projects.

Online and diagnostics

If the "PLCSIM" (Softbus) communication interface is set, no details are displayed for the "Online and Diagnostics" function under the PROFINET interface (IP address, MAC address, etc.).

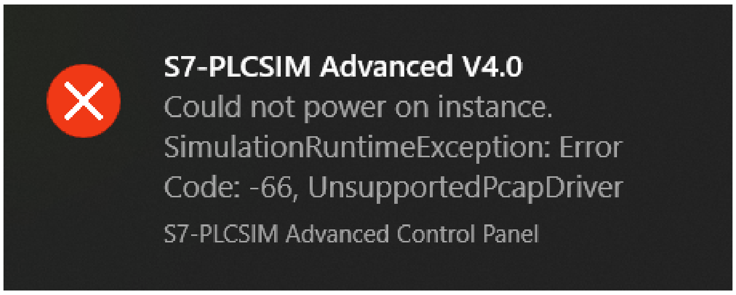
See also

Controller - Information and settings (Page 144)

8.9 Messages for communication via TCP/IP

Error codes

If an ID with error designation appears in the taskbar, you will find the description in section 7 User interfaces (API) (Page 84).



S7-PLCSIM Advanced V4.0
Could not power on instance.
SimulationRuntimeException: Error Code: -66, UnsupportedPcapDriver
S7-PLCSIM Advanced Control Panel

Figure 8-1 Example: Error code -66

Messages and remedy

The settings for TCP/IP communication are checked in the S7-PLCSIM Advanced Control Panel. The messages and the corresponding remedies are listed below:

Message

"Siemens PLCSIM Virtual Ethernet Adapter was not found. Please reinstall PLCSIM Advanced."

Remedy

The PLCSIM Virtual Ethernet Adapter cannot be found on the system.

Run PLCSIM Advanced Setup again:

1. Double-click the download package or insert the installation medium into the drive. The setup program starts up automatically, provided you have not disabled the Autostart function on the computer. If the setup program does not start up automatically, start it manually by double-clicking the "Start.exe" file.
2. Follow the prompts until you reach the "Configuration" window. Select the "Repair" check box.
3. Follow the remaining prompts to repair your installation.
4. Complete the repair operation by restarting your computer.

Message

"Siemens PLCSIM Virtual Ethernet Adapter is disabled. Please enable it."

Remedy

The PLCSIM Virtual Ethernet Adapter is deactivated on the system. In the Control Panel, under "Network and Sharing Center" > "Change Adapter Settings" and activate the network adapter.

Message

"Npcap service is not running. When installed start it from elevated command prompt with 'net start npcap'."

Remedy

The Npcap is not active on the system. Open a command line in administrator mode and enter the command "net start npcap".

Message

"You have to set a valid IP address for the Siemens PLCSIM Virtual Ethernet Adapter."

Remedy

Assign a static IP address to the Siemens PLCSIM Virtual Ethernet Adapter or obtain an IP address via DHCP (default setting).

8.10 Restrictions of security with VMware vSphere Hypervisor (ESXi)

The following information is only important if you want to communicate with the "outside world". This means when you access from a visualization platform on which an instance is being executed to another VM on which the TIA Portal is being executed. When you are using TCP/IP on the local PC, the following changes are not necessary.

When you use the virtualization platform VMware vSphere Hypervisor (ESXi), you must change the policy exception to communicate over TCP/IP.

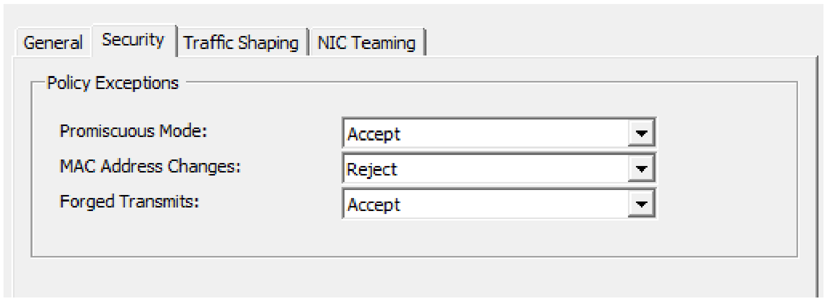
Accept the "Promiscuous mode and "Forged transmit" options for the Virtual Switch of the ESXi.

Promiscuous Mode

In Promiscuous mode, the network adapter reads all incoming telegrams, including those that are not intended for the network adapter, and forwards the data to the operating system for processing.

The Promiscuous mode is necessary for the simulation with PLCSIM Advanced so that telegrams can be forwarded by the Ethernet adapter to the PC or the VM for processing, for example, data for additional PLCSIM Advanced instances in the network.

NOTICE
Restrictions of security For security reasons, Promiscuous mode is disabled by default. If you accept the Promiscuousmode, the real Ethernet adapter even receives telegrams that are not addressed to it.



Promiscuous Mode: Accept
MAC Address Changes: Reject
Forged Transmits: Accept

Figure 8-2 Policy exceptions for VMware vSphere Hypervisor (ESXi)

8.11 Restrictions for Hyper-V

The following information is only important if you want to communicate with the "outside world". This means when you access from a visualization platform on which an instance is being executed to another VM on which the TIA Portal is being executed. When you are using TCP/IP on the local PC, the following settings are not necessary.

When you use the virtualization platform Hyper-V, you must change the following setting to be able to use the communication over TCP/IP.

1. In the Hyper-V Manager, select the Hyper-V server.
2. Select the VM and its settings via "Settings...".
3. In the "Extended Functions", select "MAC Address Spoofing".

This setting is necessary for the simulation with PLCSIM Advanced so that telegrams can be forwarded by the Ethernet adapter to the PC or the VM for processing, for example, data for additional PLCSIM Advanced instances in the network.

NOTICE

Restrictions of security

For security reasons, Promiscuous mode is disabled by default.

If you accept the Promiscuousmode, the real Ethernet adapter even receives telegrams that are not addressed to it.

8.12 Monitoring overflow

Monitoring of main cycle

The maximum cycle time monitoring for PLCSIM Advanced is one minute.

If you want to use the values that are configured in the TIA Portal, set them using the following API function: `SetCycleTimeMonitoringMode()`.

See Cycle control (Page 252)

Monitoring of cyclical events

If your simulation contains cyclic interrupts, the queue of PLCSIM Advanced may overflow for cyclic events. Due to the execution speed of PLCSIM Advanced compared to real hardware, the time required to create the diagnostics buffer entry may 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, PLCSIM Advanced provides visual information in the form of diagnostics buffer messages and a red error icon in the project tree.

See also

Speed up and slow down simulation (Page 77)

8.13 Deviating I/O values in the STEP 7 user program

Updated values

Each value change made by a STEP 7 user program in the I/O address areas is overwritten in the cycle control point with the updated value that was written via the API functions `Write...()`. The API functions `Read...()` only return this updated value and not the value from STEP 7 **for the input range**.

Non-updated values

If the value was not updated via the API functions `Write...()`, the API functions `Read...()` return the value from STEP 7 **for the output range**.

See also

Simulate peripheral I/O (Page 59)

8.14 Multiple simulations and possible collision of IP addresses

You can simultaneously simulate multiple CPUs, but each simulated CPU interface requires a unique IP address.

Make sure your CPUs have different IP addresses before starting the simulation.

8.15 Lacking access to an IP address

Special feature of distributed communication

If you use multiple network nodes on the same subnet through different virtual or real adapters, the operating system may search for the node on the wrong adapter.

Remedy

Repeat your requests or enter "arp -d <IP address>" in the command line editor of Windows.

8.16 Simulation in standby mode

If your computer or programming device goes into standby or hibernation mode, the simulation may be stopped. In this case, the communication between STEP 7 and PLCSIM Advanced is stopped. When your computer or programming device starts up again, the communication may need to be reestablished. In some cases, it may also be necessary to open the simulation project again.

To prevent this situation, disable the standby mode on your computer or programming device.

8.17 ET 200SP CPUs: Use of BusAdapters with fiber-optic interface

ET 200SP CPUs: Use of BusAdapters with fiber-optic interface

If you use BusAdapters with fiber-optic interface for connecting fiber-optic cables (e.g. BA 2xLC), then download via TCP/IP is not possible.

Remedy

Use the communication interface "PLCSIM" (Softbus) in PLCSIM Advanced.

8.18 Installing SIMATIC NET

Problems after installing SIMATIC NET PC software products

If you are using one of the following operating systems with S7-PLCSIM Advanced on your PC, problems may arise with the PLCSIM Advanced instances via TCP/IP as soon as you have installed SIMATIC NET PC software products on your PC:

- Windows 10 Pro, Version 1809
- Windows Server 2019

After installing the SIMATIC NET PC software products, you may no longer be able to start the PLCSIM Advanced instances via TCP/IP.

Remedy

Update the operating system on your PC:

- Windows 10, Version ≥ 1903 or
- Windows Server version > 2019

List of abbreviations A

Abbreviation	Term
ALM	Automation License Manager Tool for managing license keys in STEP 7
API	Application Programming Interface user interface
arp	Address resolution protocol
BCD	Binary Coded Decimal
CPU	Central Processing Unit (Synonym for PLC)
DLL	Dynamic Link Library
HMI	Human Machine Interface user interface
I-device	Intelligent IO device
IE	Industrial Ethernet
GUI	Graphical User Interface
LAN	Local Area Network Computer network that is limited to a local area.
MFP	Multifunctional platform
OB	Organization Block
ODK	Open Development Kit
OPC UA	Open Platform Communications Unified Architecture
PA	Process image output (PIQ)
PE	Process image input (PII)
PG	Programming device
PLC	Programmable Logic Controller
PN	PROFINET
RAM	Random Access Memory
RT	Runtime
SO	Shared Object
TCP/IP	Transmission Control Protocol/Internet Protocol
TIA	Totally Integrated Automation
PIP	Process Image Partition
UTC	Coordinated Universal Time
VM	Virtual Machine
VPLC	Virtual Programmable Logic Controller
WinCC	Windows Control Center