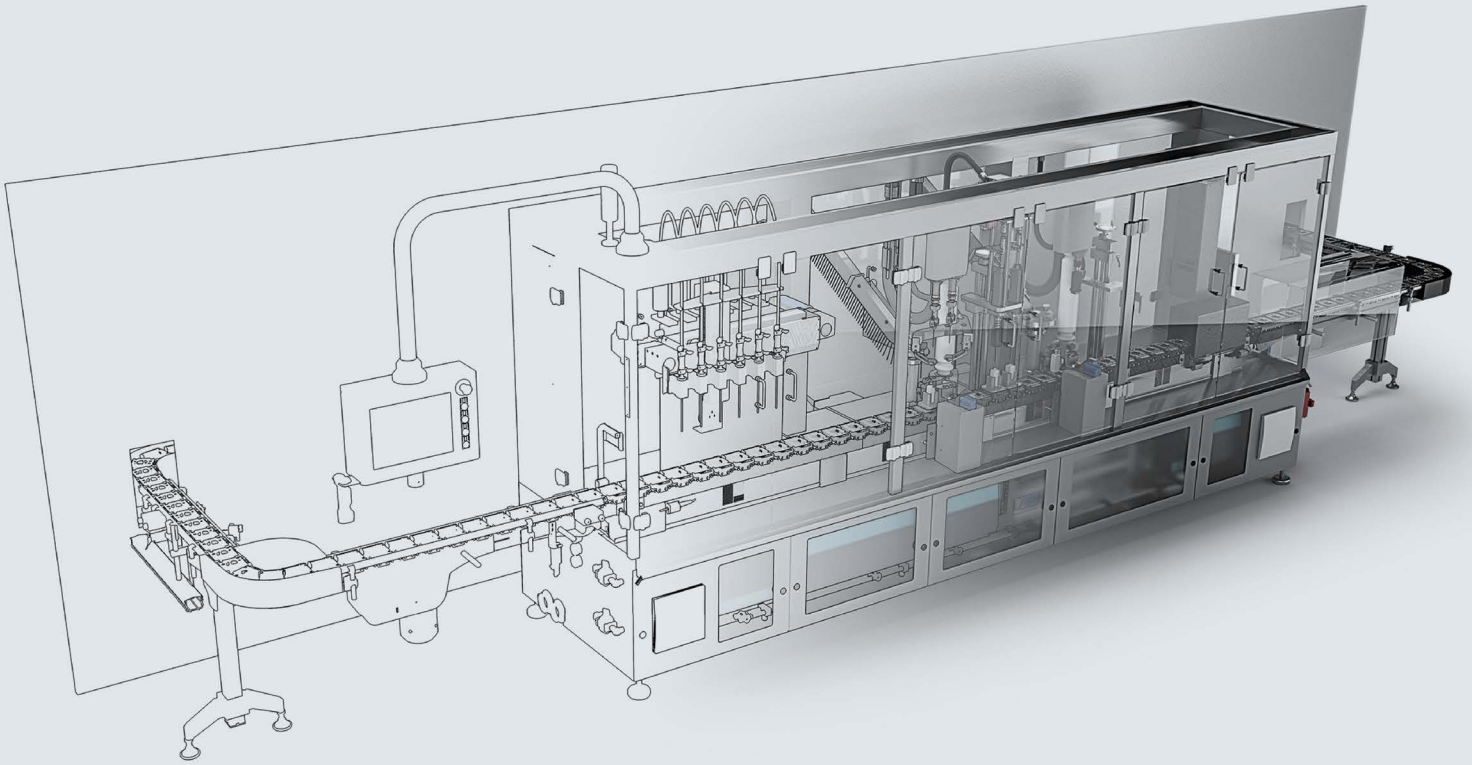
SIEMENS



Function manual

SIMATIC

S7-1500

S7-PLCSIM Advanced

Edition 12/2017

support.industry.siemens.com

SIEMENS

SIMATIC

S7-1500
S7-PLCSIM Advanced

Function Manual

Preface

Guide 1

Product overview 2

Installing 3

Communication paths 4

Simulate CPU 5

Virtual time response 6

User interfaces (API) 7

Restrictions 8

List of abbreviations A

12/2017

A5E37039512-AB

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
Division Digital Factory
Postfach 48 48
90026 NÜRNBERG
GERMANY

A5E37039512-AB
℗ 12/2017 Subject to change

Copyright © Siemens AG 2016 - 2017.
All rights reserved

Preface

Purpose of the documentation

This function manual describes the simulation software, SIMATIC S7-PLCSIM Advanced V2.0. You can use this software to simulate and test your SIMATIC STEP 7 programs on a virtual controller.

You can obtain updates to the function manual as downloads on the Internet (https://support.industry.siemens.com/cs/us/en/view/109739154).

An application example for S7-PLCSIM Advanced is also available on the Internet (https://support.industry.siemens.com/cs/ww/en/view/109739660).

Scope

This function manual is valid for the following order versions

- 6ES7823-1FA01-0YA5 - SIMATIC S7-PLCSIM Advanced V2.0 Floating License (DVD)
- 6ES7823-1FE01-0YA5 - SIMATIC S7-PLCSIM Advanced V2.0 Floating License (Download)
- 6ES7823-1FA01-0YE5 - Upgrade SIMATIC S7-PLCSIM Advanced V1.0 → V2.0 (DVD)
- 6ES7823-1FE01-0YE5 - Upgrade SIMATIC S7-PLCSIM Advanced V1.0 → V2.0 (Download)

The articles each contain one license for up to three instances.

Basic knowledge required

The system may 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
- Development of software in 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 V2.0 as "PLCSIM Advanced".

Please 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.

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 only form one element of such a concept.

Customer is responsible to prevent unauthorized access to its plants, systems, machines and networks. Systems, machines and components should only be connected to the enterprise network or the internet if and to the extent necessary and with appropriate security measures (e.g. use of firewalls and network segmentation) in place.

Additionally, Siemens' guidance on appropriate security measures should be taken into account. For more information about industrial security, please visit (http://www.siemens.com/industrialsecurity).

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

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

Table of contents

Preface **4**

1 Guide **27**

1.1 Documentation guide 27

1.2 S7-PLCSIM products 29

2 Product overview **30**

2.1 What is S7-PLCSIM Advanced? 30

2.2 Compatibility during upgrade 31

2.3 Security for S7-PLCSIM Advanced 32

2.4 Simulations support 33

2.5 Supported CPUs 34

2.6 Differences between a simulated and a real CPU 36

2.6.1 Restrictions for all supported CPUs 37

2.7 Simulating Motion Control 39

3 Installing **40**

3.1 Introduction 40

3.1.1 System requirements 40

3.1.2 Licenses 42

3.1.3 Trial License 42

3.1.4 Installation log 44

3.2 S7-PLCSIM Advanced 45

3.3 Installing S7-PLCSIM Advanced 46

3.4 Changing S7-PLCSIM Advanced 48

3.5 Repairing S7-PLCSIM Advanced 49

3.6 Uninstalling S7-PLCSIM Advanced 50

4 Communication paths **51**

4.1 Local communication 53

4.2 Communication via TCP / IP 54

4.3 Enable distributed communication 56

5 Simulate CPU **58**

5.1 Basic procedure for the simulation 58

5.2 User interface 59

5.2.1 S7 PLCSIM Advanced Symbol 59

5.2.2 S7 PLCSIM Advanced Control Panel 60

5.3 Download 64
5.4 Network addresses in the simulation 66
5.4.1 Siemens PLCSIM Virtual Ethernet Adapter 66
5.4.2 PLCSIM Advanced instances 66
5.5 Simulate peripheral I/O 68
5.6 Simulate communication 69
5.6.1 Communication services that can be simulated 69
5.6.2 Communication between instances 70
5.7 Provide project data offline for simulation 71
6 Virtual time response 73
6.1 Speed up and slow down simulation 75
6.2 Stop simulation 76
6.3 Synchronize simulation partner 78
6.3.1 Synchronize simulation partner cycle-controlled 78
6.3.2 Synchronize simulation partner time-controlled 80
7 User interfaces (API) 82
7.1 Introduction 82
7.1.1 Access to instances 84
7.1.2 User interfaces (API) 85
7.1.3 Overview of user interfaces for native C++ 86
7.1.4 Overview of user interfaces for managed code 90
7.1.5 Overview of data types for native C++ 93
7.1.6 Overview of data types for managed code 94
7.2 Initialize API 96
7.2.1 Load API library 96
7.2.2 Native C++ 96
7.2.2.1 InitializeApi() 96
7.2.2.2 RuntimeApiEntry_Initialize 98
7.2.3 .NET (C#) 99
7.2.3.1 Initialize 99
7.3 Shut down API 100
7.3.1 Native C++ 100
7.3.1.1 DestroyInterface() 101
7.3.1.2 RuntimeApiEntry_DestroyInterface 102
7.3.1.3 FreeApi() 103
7.3.1.4 ShutdownAndFreeApi() 104
7.3.2 .NET (C#) 105
7.3.2.1 Shut down API 105

7.4 Global functions (Native C++) ... 106

7.5 API ISimulationRuntimeManager ... 111
7.5.1 Interfaces - Information and settings ... 111
7.5.2 Simulation Runtime instances ... 113
7.5.3 Remote connections ... 119
7.5.4 Events ... 124
7.5.4.1 OnConfigurationChanged events ... 124
7.5.4.2 OnRuntimeManagerLost events ... 127

7.6 API IInstances ... 130
7.6.1 Interfaces - Information and settings ... 130
7.6.2 Controller - Information and settings ... 136
7.6.3 Operating state ... 144
7.6.4 Tag list ... 152
7.6.5 I/O access ... 158
7.6.5.1 Synchronizing inputs and outputs ... 158
7.6.5.2 I/O access via address - Reading ... 158
7.6.5.3 I/O access via address - Writing ... 166
7.6.5.4 I/O access via tag name - Reading ... 173
7.6.5.5 I/O access via tag name - Writing ... 191
7.6.6 Settings for the virtual time ... 220
7.6.7 Cycle control ... 223
7.6.8 Acyclic services ... 228
7.6.8.1 Overview ... 228
7.6.8.2 ReadRecordDone / WriteRecordDone ... 230
7.6.8.3 AlarmNotification ... 232
7.6.8.4 ProcessEvent ... 234
7.6.8.5 PullOrPlugEvent ... 236
7.6.8.6 StatusEvent ... 238
7.6.8.7 ProfileEvent ... 239
7.6.8.8 UpdateEvent ... 240
7.6.8.9 GetConfiguredProcessEvent ... 241
7.6.8.10 RackOrStationFaultEvent ... 242
7.6.9 Events ... 243
7.6.9.1 OnOperatingStateChanged events ... 243
7.6.9.2 OnSyncPointReached events ... 247
7.6.9.3 OnConfigurationChanging events ... 249
7.6.9.4 OnConfigurationChanged events ... 252
7.6.9.5 OnLedChanged events ... 254
7.6.10 Events for acyclic services ... 257
7.6.10.1 OnDataRecordRead / OnDataRecordWrite events ... 257
7.6.10.2 OnAlarmNotification events ... 258
7.6.10.3 OnProcessEvent events ... 259
7.6.10.4 OnPullOrPlugEvent events ... 260
7.6.10.5 OnStatusEvent events ... 261
7.6.10.6 OnProfileEvent events ... 262
7.6.10.7 OnUpdateEvent events ... 263
7.6.10.8 RackOrStationFault events ... 264

7.7 API IRemoteRuntimeManager 265
7.7.1 Interfaces - Information and settings 265
7.7.2 Simulation Runtime instances 268
7.7.2.1 Simulation Runtime instances (remote) 268
7.7.3 Events 275
7.7.3.1 OnConnectionLost events 275
7.8 Data types 279
7.8.1 DLL import functions (Native C++) 280
7.8.1.1 ApiEntry_Initialize 280
7.8.1.2 ApiEntry_DestroyInterface 280
7.8.2 Event callback functions (Native C++) 281
7.8.2.1 EventCallback_VOID 281
7.8.2.2 EventCallback_II_SREC_ST 281
7.8.2.3 EventCallback_II_SREC_ST_SROS_SROS 282
7.8.2.4 EventCallback_II_SREC_ST_SRLT_SRLM 283
7.8.2.5 EventCallback_II_SREC_ST_UINT32_INT64_INT64_UINT32 284
7.8.2.6 EventCallback_IRRTM 285
7.8.2.7 EventCallback_SRCC_UINT32_UINT32_INT32 285
7.8.2.8 EventCallback_II_SREC_ST_SRICC_UINT32_UINT32_UINT32_UINT32 286
7.8.2.9 EventCallback_II_SREC_ST_SDRI 287
7.8.2.10 EventCallback_II_SREC_ST_SDR 288
7.8.2.11 EventCallback_II_SREC_ST_UINT32_EPPET_UINT32 289
7.8.2.12 EventCallback_II_SREC_ST_UINT32_UINT32_EPET_UINT32 290
7.8.2.13 EventCallback_II_SREC_ST_UINT32 291
7.8.2.14 EventCallback_II_SREC_ST_UINT32_UINT32 292
7.8.2.15 EventCallback_II_SREC_ST_UINT32_ERSFET 293
7.8.3 Delegate definitions (managed code) 293
7.8.3.1 Delegate_Void 293
7.8.3.2 Delegate_II_EREC_DT 294
7.8.3.3 Delegate_II_EREC_DT_EOS_EOS 294
7.8.3.4 Delegate_II_EREC_DT_ELT_ELM 295
7.8.3.5 Delegate_II_EREC_DT_UINT32_INT64_INT64_UINT32 296
7.8.3.6 Delegate_IRRTM 297
7.8.3.7 Delegate_SRCC_UINT32_UINT32_INT32 297
7.8.3.8 Delegate_II_EREC_DT_SRICC_UINT32_UINT32_UINT32_UINT32 298
7.8.3.9 Delegate_II_EREC_DT_SDRI 299
7.8.3.10 Delegate_II_EREC_DT_SDR 300
7.8.3.11 Delegate_SREC_ST_UINT32_EPPET_UINT32 301
7.8.3.12 Delegate_SREC_ST_UINT32_UINT32_EPET_UINT32 302
7.8.3.13 Delegate_SREC_ST_UINT32 303
7.8.3.14 Delegate_SREC_ST_UINT32_UINT32 304
7.8.3.15 Delegate_SREC_ST_UINT32_ERSFET 305
7.8.4 Definitions and constants 306
7.8.5 Unions (Native C++) 307
7.8.5.1 UIP 307
7.8.5.2 UDataValue 308

7.8.6 Structures ... 309
7.8.6.1 SDataValue ... 309
7.8.6.2 SDVBNI ... 311
7.8.6.3 SDataValueByAddress ... 311
7.8.6.4 SDataValueByAddressWithCheck ... 312
7.8.6.5 SDataValueByName ... 312
7.8.6.6 SDataValueByNameWithCheck ... 313
7.8.6.7 SConnectionInfo ... 313
7.8.6.8 SInstanceInfo ... 314
7.8.6.9 SDimension ... 314
7.8.6.10 STagInfo ... 315
7.8.6.11 SIP ... 317
7.8.6.12 SIPSuite4 ... 317
7.8.6.13 SOnSyncPointReachedResult ... 319
7.8.6.14 SDataRecordInfo ... 321
7.8.6.15 SDataRecord ... 322
7.8.6.16 SConfiguredProcessEvents ... 323
7.8.6.17 SDiagExtChannelDescription ... 324
7.8.7 Enumerations ... 326
7.8.7.1 ERuntimeErrorCode ... 327
7.8.7.2 EArea ... 330
7.8.7.3 EOperatingState ... 330
7.8.7.4 EOperatingMode ... 331
7.8.7.5 ECPUType ... 332
7.8.7.6 ECommunicationInterface ... 334
7.8.7.7 ELEDType ... 334
7.8.7.8 ELEDMode ... 335
7.8.7.9 EPrimitiveDataType ... 335
7.8.7.10 EDataType ... 338
7.8.7.11 ETagListDetails ... 342
7.8.7.12 ERuntimeConfigChanged ... 343
7.8.7.13 EInstanceConfigChanged ... 343
7.8.7.14 EPullOrPlugEventType ... 344
7.8.7.15 EProcessEventType ... 344
7.8.7.16 EDirection ... 345
7.8.7.17 EDiagProperty ... 345
7.8.7.18 EDiagSeverity ... 346
7.8.7.19 ERackOrStationFaultType ... 346

8 Restrictions ... 347

8.1 Overview ... 347
8.2 Restrictions with fail-safe CPUs ... 347
8.3 OPC UA server ... 348
8.4 Web server ... 349
8.5 Backing up and restoring the configuration of a PLCSIM Advanced instance ... 350
8.6 Restrictions for file paths ... 350
8.7 Restrictions for communications services ... 351
8.8 Restrictions for instructions ... 351

8.9 Restrictions with Motion Control 352
8.9.1 MCR and EMCR 352
8.9.2 Technology modules 353
8.9.3 OB 91 and OB 92 353
8.10 Restrictions to local communication via Softbus 354
8.11 Restrictions of security with VMware vSphere Hypervisor (ESXi) 355
8.12 Error with overflow of cyclical events and motion OBs 355
8.13 Deviating I/O values in the STEP 7 user program 356
8.14 Multiple simulations and possible collision of IP addresses 356
8.15 Lacking access to an IP address 356
8.16 Simulation in standby mode 356
8.17 Error installing the antivirus software from Kaspersky 357

A List of abbreviations 358

Tables

Table 1- 1 Comparison of S7-PLCSIM products 29
Table 2- 1 Compatibility of API and Runtime versions 31
Table 2- 2 Supported CPUs 34
Table 2- 3 Compatibility to the TIA Portal 35
Table 3- 1 Minimum requirements for hardware 40
Table 5- 1 Assignment of the Ethernet interfaces, for example, for a CPU 1518-4 PN/DP 67
Table 5- 2 Supported communication options 69
Table 6- 1 Cycle-controlled operating modes (SingleStep) 78
Table 6- 2 Time-controlled operating modes (TimespanSynchronized) 80
Table 7- 1 Overview of initializing and shutting down API - Native C++ 86
Table 7- 2 Overview of global functions - Native C++ 86
Table 7- 3 Overview of API ISimulationRuntimeManager functions - Native C++ 87
Table 7- 4 Overview of API ISimulationRuntimeManager events - Native C++ 87
Table 7- 5 Overview of IInstances functions - Native C++ 87
Table 7- 6 Overview of IInstances events - Native C++ 89
Table 7- 7 Overview of IRemoteRuntimeManager functions - Native C++ 89
Table 7- 8 Overview of IRemoteRuntimeManager events - Native C++ 89
Table 7- 9 Overview of initializing and shutting down API - .NET (C#) 90
Table 7- 10 Overview of ISimulationRuntimeManager functions - .NET (C#) 90
Table 7- 11 Overview of ISimulationRuntimeManager events - .NET (C#) 90
Table 7- 12 Overview of IInstances functions - .NET (C#) 91
Table 7- 13 Overview of IInstances events - .NET (C#) 92
Table 7- 14 Overview of IRemoteRuntimeManager functions - .NET (C#) 92
Table 7- 15 Overview IRemoteRuntimeManager events - .NET (C#) 92
Table 7- 16 Overview of data types - Native C++ 93
Table 7- 17 Overview of data types - .NET (C#) 94
Table 7- 18 InitializeApi() - Native C++ 97
Table 7- 19 RuntimeApiEntry_Initialize - Native C++ 98
Table 7- 20 Initialize - .NET (C#) 99
Table 7- 21 DestroyInterface() - Native C++ 101
Table 7- 22 RuntimeApiEntry_DestroyInterface() - Native C++ 102
Table 7- 23 FreeApi() - Native C++ 103
Table 7- 24 ShutdownAndFreeApi() - Native C++ 104

Table 7- 25 GetNameOfAreaSection() - Native C++ 106
Table 7- 26 GetNameOfCPUType() - Native C++ 106
Table 7- 27 GetNameOfCommunicationInterface() - Native C++ 106
Table 7- 28 GetNameOfDataType() - Native C++ 106
Table 7- 29 GetNameOfErrorCode() - Native C++ 107
Table 7- 30 GetNameOfLEDMode() - Native C++ 107
Table 7- 31 GetNameOfLEDType() - Native C++ 107
Table 7- 32 GetNameOfOperatingMode() - Native C++ 107
Table 7- 33 GetNameOfErrorCode() - Native C++ 108
Table 7- 34 GetNameOfOperatingState() - Native C++ 108
Table 7- 35 GetNameOfPrimitiveDataType() - Native C++ 108
Table 7- 36 GetNameOfTagListDetails() - Native C++ 108
Table 7- 37 GetNameOfRuntimeConfigChanged() - Native C++ 109
Table 7- 38 GetNameOfInstanceConfigChanged() - Native C++ 109
Table 7- 39 GetNameOfDirection() - Native C++ 109
Table 7- 40 GetNameOfDiagSeverity() - Native C++ 109
Table 7- 41 GetNameOfRackOrStationFaultType() - Native C++ 110
Table 7- 42 GetVersion() - Native C++ 111
Table 7- 43 Version { get; } - .NET (C#) 111
Table 7- 44 IsInitialized() - Native C++ 111
Table 7- 45 IsInitialized { get; } - .NET (C#) 111
Table 7- 46 IsRuntimeManagerAvailable() - Native C++ 112
Table 7- 47 IsRuntimeManagerAvailable { get; } - .NET (C#) 112
Table 7- 48 Shutdown() - Native C++ 112
Table 7- 49 Shutdown() - .NET (C#) 112
Table 7- 50 GetRegisteredInstancesCount() - Native C++ 113
Table 7- 51 GetRegisteredInstanceInfoAt() - Native C++ 113
Table 7- 52 RegisteredInstanceInfo { get; } - .NET (C#) 113
Table 7- 53 RegisterInstance() - Native C++ 114
Table 7- 54 RegisterInstance() - .NET (C#) 115
Table 7- 55 RegisterCustomInstance() - Native C++ 116
Table 7- 56 RegisterCustomInstance() - .NET (C#) 117
Table 7- 57 CreateInterface() - Native C++ 118
Table 7- 58 CreateInterface() - .NET (C#) 119
Table 7- 59 OpenPort() - Native C++ 119
Table 7- 60 OpenPort() - .NET (C#) 120

Table 7- 61 ClosePort() - Native C++ 120
Table 7- 62 ClosePort() - .NET (C#) 120
Table 7- 63 GetPort() - Native C++ 120
Table 7- 64 Port { get; } - .NET (C#) 121
Table 7- 65 GetRemoteConnectionsCount() - Native C++ 121
Table 7- 66 GetRemoteConnectionInfoAt()- Native C++ 121
Table 7- 67 RemoteConnectionInfo { get; } - .NET (C#) 121
Table 7- 68 RemoteConnect() - Native C++ 122
Table 7- 69 RemoteConnect() - .NET (C#) 123
Table 7- 70 Events for the ISimulationRuntimeManager interface 124
Table 7- 71 OnConfigurationChanged - .NET (C#) 124
Table 7- 72 RegisterOnConfigurationChangedCallback() - Native C++ 124
Table 7- 73 RegisterOnConfigurationChangedEvent() - Native C++ 125
Table 7- 74 RegisterOnConfigurationChangedEvent() - .NET (C#) 125
Table 7- 75 UnregisterOnConfigurationChangedCallback() - Native C++ 125
Table 7- 76 UnregisterOnConfigurationChangedEvent() - Native C++ 125
Table 7- 77 UnregisterOnConfigurationChangedEvent() - .NET (C#) 126
Table 7- 78 WaitForOnConfigurationChangedEvent() - Native C++ 126
Table 7- 79 WaitForOnConfigurationChangedEvent - .NET (C#) 126
Table 7- 80 OnRuntimeManagerLost - .NET (C#) 127
Table 7- 81 RegisterOnRuntimeManagerLostCallback() - Native C++ 127
Table 7- 82 RegisterOnRuntimeManagerLostEvent() - Native C++ 127
Table 7- 83 RegisterOnRuntimeManagerLostEvent() - .NET (C#) 128
Table 7- 84 UnregisterOnRuntimeManagerLostCallback() - Native C++ 128
Table 7- 85 UnregisterOnRuntimeManagerLostEvent() - Native C++ 128
Table 7- 86 UnregisterOnRuntimeManagerLostEvent() - .NET (C#) 128
Table 7- 87 WaitForOnRuntimeManagerLostEvent() - Native C++ 129
Table 7- 88 WaitForOnRuntimeManagerLostEvent() - .NET (C#) 129
Table 7- 89 Dispose() - .NET (C#) 130
Table 7- 90 GetID() - Native C++ 130
Table 7- 91 ID { get; } - .NET (C#) 130
Table 7- 92 GetName() - Native C++ 131
Table 7- 93 Name { get; } - .NET (C#) 131
Table 7- 94 GetCPUType() - Native C++ 132
Table 7- 95 SetCPUType() - Native C++ 132
Table 7- 96 CPUType { get; set; } - .NET (C#) 132

Table 7- 97 GetCommunicationInterface() - Native C++ 133
Table 7- 98 SetCommunicationInterface() - Native C++ 133
Table 7- 99 CommunicationInterface { get; set; } - .NET (C#) 134
Table 7- 100 GetInfo() - Native C++ 134
Table 7- 101 Info { get; } - .NET (C#) 134
Table 7- 102 UnregisterInstance() - Native C++ 135
Table 7- 103 UnregisterInstance() - .NET (C#) 135
Table 7- 104 GetControllerName() - Native C++ 136
Table 7- 105 ControllerName { get; } - .NET (C#) 136
Table 7- 106 GetControllerShortDesignation() - Native C++ 137
Table 7- 107 ControllerShortDesignation { get; } - .NET (C#) 137
Table 7- 108 GetControllerIPCount() - Native C++ 137
Table 7- 109 GetControllerIP() - Native C++ 138
Table 7- 110 ControllerIP { get; } - .NET (C#) 138
Table 7- 111 GetControllerIPSuite4() Native C++ 138
Table 7- 112 ControllerIPSuite4 { get; } - .NET (#) 138
Table 7- 113 SetIPSuite() - Native C++ 139
Table 7- 114 SetIPSuite() - .NET (C#) 140
Table 7- 115 GetStoragePath() - Native C++ 140
Table 7- 116 SetStoragePath() - Native C++ 141
Table 7- 117 StoragePath { get; set; } - .NET (C#) 141
Table 7- 118 ArchiveStorage() - Native C++ 142
Table 7- 119 ArchiveStorage() - .NET (C#) 142
Table 7- 120 RetrieveStorage() - Native C++ 143
Table 7- 121 RetrieveStorage() - .NET (C#) 143
Table 7- 122 GetOperatingState() - Native C++ 144
Table 7- 123 OperatingState { get; } - .NET (C#) 145
Table 7- 124 PowerOn() - Native C++ 146
Table 7- 125 PowerOn() - .NET (C#) 147
Table 7- 126 PowerOff() - Native C++ 148
Table 7- 127 PowerOff() - .NET (C#) 148
Table 7- 128 MemoryReset() - Native C++ 149
Table 7- 129 MemoryReset() - .NET (C#) 149
Table 7- 130 Run() - Native C++ 150
Table 7- 131 Run() - .NET (C#) 150
Table 7- 132 Stop() - Native C++ 151

Table 7- 133 Stop() - .NET (C#) 151
Table 7- 134 UpdateTagList() - Native C++ 153
Table 7- 135 UpdateTagList() - .NET (C#) 154
Table 7- 136 GetTagListStatus() - Native C++ 155
Table 7- 137 GetTagListStatus() - .NET (C#) 155
Table 7- 138 GetTagInfoCount() - Native C++ 155
Table 7- 139 GetTagInfos() - Native C++ 156
Table 7- 140 TagInfos { get; } - .NET (C#) 156
Table 7- 141 CreateConfigurationFile() - Native C++ 157
Table 7- 142 CreateConfigurationFile() - .NET (C#) 157
Table 7- 143 InputArea { get; } MarkerArea { get; } OutputArea { get; } - .NET (C#) 158
Table 7- 144 GetAreaSize() - Native C++ 159
Table 7- 145 AreaSize { get; } - .NET (C#) 159
Table 7- 146 ReadBit() - Native C++ 160
Table 7- 147 ReadBit() - .NET (C#) 160
Table 7- 148 ReadByte() - Native C++ 161
Table 7- 149 ReadByte() - .NET (C#) 162
Table 7- 150 ReadByte() - Native C++ 163
Table 7- 151 ReadBytes() - .NET (C#) 164
Table 7- 152 ReadSignals() - Native C++ 165
Table 7- 153 ReadSignals() - .NET (C#) 166
Table 7- 154 WriteBit() - Native C++ 167
Table 7- 155 WriteBit() - .NET (C#) 168
Table 7- 156 WriteByte() - Native C++ 169
Table 7- 157 WriteByte() - .NET (C#) 169
Table 7- 158 WriteBytes() - Native C++ 170
Table 7- 159 WriteBytes() - .NET (C#) 171
Table 7- 160 WriteSignals() - Native C++ 172
Table 7- 161 WriteSignals() - .NET (C#) 173
Table 7- 162 Read() - Native C++ 174
Table 7- 163 Read() - .NET (C#) 175
Table 7- 164 ReadBool() - Native C++ 176
Table 7- 165 ReadBool() - .NET (C#) 176
Table 7- 166 ReadInt8() - Native C++ 177
Table 7- 167 ReadInt8() - .NET (C#) 177
Table 7- 168 ReadInt16() - Native C++ 178

Table 7- 169 ReadInt16() - .NET (C#) 178
Table 7- 170 ReadInt32() - Native C++ 179
Table 7- 171 ReadInt32() - .NET (C#) 179
Table 7- 172 ReadInt64() - Native C++ 180
Table 7- 173 ReadInt64() - .NET (C#) 180
Table 7- 174 ReadUInt8() - Native C++ 181
Table 7- 175 ReadUInt8() - .NET (C#) 181
Table 7- 176 ReadUInt16() - Native C++ 182
Table 7- 177 ReadUInt16() - .NET (C#) 182
Table 7- 178 ReadUInt32() - Native C++ 183
Table 7- 179 ReadUInt32() - .NET (C#) 183
Table 7- 180 ReadInt64() - Native C++ 184
Table 7- 181 ReadUInt64() - .NET (C#) 184
Table 7- 182 ReadFloat() - Native C++ 185
Table 7- 183 ReadFloat() - .NET (C#) 185
Table 7- 184 ReadDouble() - Native C++ 186
Table 7- 185 ReadDouble() - .NET (C#) 186
Table 7- 186 ReadChar() - Native C++ 187
Table 7- 187 ReadChar() - .NET (C#) 187
Table 7- 188 ReadWChar() - Native C++ 188
Table 7- 189 ReadWChar() - .NET (C#) 188
Table 7- 190 ReadSignals() - Native C++ 189
Table 7- 191 ReadSignals() - .NET (C#) 190
Table 7- 192 Write() - Native C++ 191
Table 7- 193 Write() - .NET (C#) 192
Table 7- 194 WriteBool() - Native C++ 193
Table 7- 195 WriteBool() - .NET (C#) 194
Table 7- 196 WriteInt8() - Native C++ 195
Table 7- 197 WriteInt8() - .NET (C#) 196
Table 7- 198 WriteInt16() - Native C++ 197
Table 7- 199 WriteInt16() - .NET (C#) 198
Table 7- 200 WriteInt32() - Native C++ 199
Table 7- 201 WriteInt32() - .NET (C#) 200
Table 7- 202 WriteInt64() - Native C++ 201
Table 7- 203 WriteInt64() - .NET (C#) 202
Table 7- 204 WriteUInt8() - Native C++ 203

Table 7- 205 WriteUInt8() - .NET (C#) 204
Table 7- 206 WriteUInt16() - Native C++....205
Table 7- 207 WriteUInt16() - .NET (C#) 206
Table 7- 208 WriteUInt32() - Native C++....207
Table 7- 209 WriteUInt32() - .NET (C#) 208
Table 7- 210 WriteUInt64() - Native C++....209
Table 7- 211 WriteUInt64() - .NET (C#) 210
Table 7- 212 WriteFloat() - Native C++ 211
Table 7- 213 WriteFloat() - .NET (C#)....212
Table 7- 214 WriteDouble() - Native C++....213
Table 7- 215 WriteDouble() - .NET (C#)....214
Table 7- 216 WriteChar() - Native C++ 215
Table 7- 217 WriteChar() - .NET (C#) 216
Table 7- 218 WriteWChar() - Native C++....217
Table 7- 219 WriteWChar() - .NET (C#)....218
Table 7- 220 WriteSignals() - Native C++ 219
Table 7- 221 WriteSignals() - .NET (C#) 220
Table 7- 222 GetSystemTime() - Native C++....220
Table 7- 223 SetSystemTime() - Native C++ 221
Table 7- 224 SystemTime { get; set; } - .NET (C#) 221
Table 7- 225 GetScaleFactor() - Native C++ 221
Table 7- 226 SetScaleFactor() - Native C++....222
Table 7- 227 ScaleFactor { get; set; } - .NET (C#) 223
Table 7- 228 GetOperatingMode() - Native C++....223
Table 7- 229 SetOperatingMode() - Native C++ 223
Table 7- 230 OperatingMode { get; set; } - .NET (C#)....224
Table 7- 231 SetSendSyncEventInDefaultModeEnabled() - Native C++....224
Table 7- 232 IsSendSyncEventInDefaultModeEnabled() - Native C++ 224
Table 7- 233 IsSendSyncEventInDefaultModeEnabled { get; set; } - .NET (C#) 225
Table 7- 234 GetOverwrittenMinimalCycleTime_ns() - Native C++ 225
Table 7- 235 SetOverwrittenMinimalCycleTime_ns() - Native C++ 225
Table 7- 236 OverwrittenMinimalCycleTime_ns { get; set; } - .NET (C#)....226
Table 7- 237 RunToNextSyncPoint() - Native C++ 226
Table 7- 238 RunToNextSyncPoint() - .NET (C#)....226
Table 7- 239 StartProcessing() - Native C++ 227
Table 7- 240 StartProcessing() - .NET (C#)....227

Table 7- 241 Overview - API methods and associated events 228
Table 7- 242 ReadRecordDone() - Native C++ 230
Table 7- 243 ReadRecordDone() - .NET (C#) 230
Table 7- 244 WriteRecordDone() - Native C++ 231
Table 7- 245 WriteRecordDone() - .NET (C#) 231
Table 7- 246 AlarmNotification() - Native C++ 232
Table 7- 247 AlarmNotification() - .NET (C#) 233
Table 7- 248 ProcessEvent() - Native C++ 234
Table 7- 249 ProcessEvent() - .NET (C#) 235
Table 7- 250 PullOrPlugEvent() - Native C++ 236
Table 7- 251 PullOrPlugEvent() - .NET (C#) 237
Table 7- 252 StatusEvent() - Native C++ 238
Table 7- 253 StatusEvent() - .NET (C#) 238
Table 7- 254 ProfileEvent() - Native C++ 239
Table 7- 255 ProfileEvent() - .NET (C#) 239
Table 7- 256 UpdateEvent() - Native C++ 240
Table 7- 257 UpdateEvent() - .NET (C#) 240
Table 7- 258 GetConfiguredProcessEvents() - Native C++ 241
Table 7- 259 GetConfiguredProcessEvents() - .NET (C#) 241
Table 7- 260 RackOrStationFaultEvent() - Native C++ 242
Table 7- 261 RackOrStationFaultEvent() - .NET (C#) 242
Table 7- 262 Events for the IInstances interface 243
Table 7- 263 OnOperatingStateChanged - .NET (C#) 243
Table 7- 264 RegisterOnOperatingStateChangedCallback() - Native C++ 244
Table 7- 265 RegisterOnOperatingStateChangedEvent() - Native C++ 244
Table 7- 266 UnregisterOnOperatingStateChangedCallback() - Native C++ 245
Table 7- 267 UnregisterOnOperatingStateChangedEvent() - Native C++ 245
Table 7- 268 UnregisterOnOperatingStateChangedEvent() - .NET (C#) 246
Table 7- 269 WaitForOnOperatingStateChangedEvent() - Native C++ 246
Table 7- 270 WaitForOnOperatingStateChangedEvent() - .NET (C#) 246
Table 7- 271 OnSyncPointReached - .NET (C#) 247
Table 7- 272 RegisterOnSyncPointReachedCallback() - Native C++ 247
Table 7- 273 RegisterOnSyncPointReachedEvent() - Native C++ 248
Table 7- 274 UnregisterOnSyncPointReachedCallback() - Native C++ 248
Table 7- 275 UnregisterOnSyncPointReachedEvent() - Native C++ 248
Table 7- 276 UnregisterOnSyncPointReachedEvent() - .NET (C#) 248

Table 7- 277 WaitForOnSyncPointReachedEvent() - Native C++ 249
Table 7- 278 WaitForOnSyncPointReachedEvent() - .NET (C#) 249
Table 7- 279 OnConfigurationChanging - .NET (C#) 249
Table 7- 280 RegisterOnConfigurationChangingCallback() - Native C++ 250
Table 7- 281 RegisterOnConfigurationChangingEvent() - Native C++ 250
Table 7- 282 UnregisterOnConfigurationChangingCallback() - Native C++ 250
Table 7- 283 UnregisterOnConfigurationChangingEvent() - Native C++ 251
Table 7- 284 UnregisterOnConfigurationChangingEvent() - .NET (C#) 251
Table 7- 285 WaitForOnConfigurationChangingEvent() - Native C++ 251
Table 7- 286 WaitForOnConfigurationChangingEvent() - .NET (C#) 251
Table 7- 287 OnConfigurationChanged - .NET (C#) 252
Table 7- 288 RegisterOnConfigurationChangedCallback() - Native C++ 252
Table 7- 289 RegisterOnConfigurationChangedEvent() - Native C++ 253
Table 7- 290 UnregisterOnConfigurationChangedCallback() - Native C++ 253
Table 7- 291 UnregisterOnConfigurationChangedEvent() - Native C++ 253
Table 7- 292 UnregisterOnConfigurationChangedEvent() - .NET (C#) 253
Table 7- 293 WaitForOnConfigurationChangedEvent() - Native C++ 254
Table 7- 294 WaitForOnConfigurationChangedEvent() - .NET (C#) 254
Table 7- 295 OnLedChanged - .NET (C#) 254
Table 7- 296 RegisterOnLedChangedCallback() - Native C++ 255
Table 7- 297 RegisterOnLedChangedEvent() - Native C++ 255
Table 7- 298 UnregisterOnLedChangedCallback() - Native C++ 255
Table 7- 299 UnregisterOnLedChangedEvent() - Native C++ 256
Table 7- 300 UnregisterOnLedChangedEvent() - .NET (C#) 256
Table 7- 301 WaitForOnLedChangedEvent() - Native C++ 256
Table 7- 302 WaitForOnLedChangedEvent() - .NET (C#) 256
Table 7- 303 OnDataRecordRead - .NET (C#) 257
Table 7- 304 OnDataRecordWrite - .NET (C#) 257
Table 7- 305 RegisterOnDataRecordReadCallback() - Native C++ 257
Table 7- 306 UnregisterOnDataRecordReadCallback() - Native C++ 258
Table 7- 307 OnAlarmNotificationDone() - .NET (C#) 258
Table 7- 308 RegisterOnAlarmNotificationDoneCallback() - Native C++ 258
Table 7- 309 UnregisterOnAlarmNotificationDoneCallback() - Native C++ 259
Table 7- 310 OnProcessEventDone() - .NET (C#) 259
Table 7- 311 RegisterOnProcessEventDoneCallback() - Native C++ 259
Table 7- 312 UnregisterOnProcessEventDoneCallback() - Native C++ 260

Table 7- 313 OnPullOrPlugEventDone() - .NET (C#) 260
Table 7- 314 RegisterOnPullOrPlugEventDoneCallback() - Native C++ 260
Table 7- 315 UnregisterOnPullOrPlugEventDoneCallback() - Native C++ 261
Table 7- 316 OnStatusEventDone() - .NET (C#) 261
Table 7- 317 RegisterOnStatusEventDoneCallback() - Native C++ 261
Table 7- 318 UnregisterOnStatusEventDoneCallback() - Native C++ 262
Table 7- 319 OnProfileEventDone() - .NET (C#) 262
Table 7- 320 RegisterOnProfileEventDoneCallback() - Native C++ 262
Table 7- 321 UnregisterOnProfileEventDoneCallback() - Native C++ 263
Table 7- 322 OnUpdateEventDone() - .NET (C#) 263
Table 7- 323 RegisterOnUpdateEventDoneCallback() - Native C++ 263
Table 7- 324 UnregisterOnUpdateEventDoneCallback() - Native C++ 264
Table 7- 325 OnRackOrStationFaultEvent - .NET (C#) 264
Table 7- 326 RegisterOnRackOrStationFaultEventCallback() - Native C++ 264
Table 7- 327 UnregisterOnRackOrStationFaultEventCallback() - Native C++ 264
Table 7- 328 Dispose() - .NET (C#) 265
Table 7- 329 GetVersion() - Native C++ 265
Table 7- 330 Version { get; } - .NET (C#) 265
Table 7- 331 GetIP() - Native C++ 266
Table 7- 332 IP { get; } - .NET (C#) 266
Table 7- 333 GetPort() - Native C++ 266
Table 7- 334 Port { get; } - .NET (C#) 266
Table 7- 335 GetRemoteComputerName() - Native C++ 267
Table 7- 336 RemoteComputerName { get; } - .NET (C#) 267
Table 7- 337 Disconnect() - Native C++ 267
Table 7- 338 Disconnect() - .NET (C#) 268
Table 7- 339 GetRegisteredInstancesCount() - Native C++ 268
Table 7- 340 GetRegisteredInstanceInfoAt() - Native C++ 268
Table 7- 341 RegisterInstanceInfo { get; } - .NET (C#) 269
Table 7- 342 RegisterInstance() - Native C++ 269
Table 7- 343 RegisterInstance() - .NET (C#) 271
Table 7- 344 RegisterCustomInstance() - Native C++ 272
Table 7- 345 RegisterCustomInstance() - .NET (C#) 273
Table 7- 346 CreateInterface() - Native C++ 274
Table 7- 347 CreateInterface() - .NET (C#) 275
Table 7- 348 OnConnectionLost - .NET (C#) 275

Table 7- 349 RegisterOnConnectionLostCallback() - Native C++ 276
Table 7- 350 RegisterOnConnectionLostEvent() - Native C++ 276
Table 7- 351 RegisterOnConnectionLostEvent() - .NET (C#) 276
Table 7- 352 UnregisterOnConnectionLostCallback() - Native C++ 277
Table 7- 353 UnregisterOnConnectionLostEvent() - Native C++ 277
Table 7- 354 UnregisterOnConnectionLostEvent() - .NET (C#) 277
Table 7- 355 WaitForOnConnectionLostEvent() - Native C++ 277
Table 7- 356 WaitForOnConnectionLostEvent() - .NET (C#) 278
Table 7- 357 ApiEntry_Initialize - Native C++ 280
Table 7- 358 ApiEntry_DestroyInterface - Native C++ 280
Table 7- 359 EventCallback_VOID - Native C++ 281
Table 7- 360 EventCallback_II_SREC_ST - Native C++ 281
Table 7- 361 EventCallback_II_SREC_ST_SROS_SROS - Native C++ 282
Table 7- 362 EventCallback_II_SREC_ST_SRLT_SRLM - Native C++ 283
Table 7- 363 EventCallback_II_SREC_ST_UINT32_INT64_INT64_UINT32 - Native C++ 284
Table 7- 364 EventCallback_IRRTM - Native C++ 285
Table 7- 365 EventCallback_SRCC_UINT32_UINT32_INT32 - Native C++ 285
Table 7- 366 EventCallback_II_SREC_ST_SRICC_UINT32_UINT32_UINT32_UINT32 - Native C++ 286
Table 7- 367 EventCallback_II_SREC_ST_SDRI - Native C++ 287
Table 7- 368 EventCallback_II_SREC_ST_SDR - .NET (C#) 288
Table 7- 369 EventCallback_II_SREC_ST_UINT32_EPPET_UINT32 - Native C++ 289
Table 7- 370 EventCallback_II_SREC_ST_UINT32_UINT32_EPET_UINT32 - Native C++ 290
Table 7- 371 EventCallback_II_SREC_ST_UINT32 - Native C++ 291
Table 7- 372 EventCallback_II_SREC_ST_UINT32_UINT32 - Native C++ 292
Table 7- 373 EventCallback_II_SREC_ST_UINT32_ERSFET - Native C++ 293
Table 7- 374 Delegate_Void - .NET (C#) 293
Table 7- 375 Delegate_II_EREC_DT - .NET (C#) 294
Table 7- 376 Delegate_II_EREC_DT_EOS_EOS - .NET (C#) 294
Table 7- 377 Delegate_II_EREC_DT_ELT_ELM - .NET (C#) 295
Table 7- 378 Delegate_II_EREC_DT_UINT32_INT64_INT64_UINT32 - .NET (C#) 296
Table 7- 379 Delegate_IRRTM - .NET (C#) 297
Table 7- 380 Delegate_SRCC_UINT32_UINT32_INT32 - .NET (C#) 297
Table 7- 381 Delegate_II_EREC_DT_SRICC_UINT32_UINT32_UINT32_UINT32 - .NET (C#) 298
Table 7- 382 Delegate_II_EREC_DT_SDRI - .NET (C#) 299
Table 7- 383 Delegate_II_EREC_DT_SDR - .NET (C#) 300
Table 7- 384 Delegate_SREC_ST_UINT32_EPPET_UINT32 - .NET (C#) 301

Table 7- 385 Delegate_SREC_ST_UINT32_UINT32_EPET_UINT32 - Native C++ 302
Table 7- 386 Delegate_SREC_ST_UINT32 - .NET (C#) 303
Table 7- 387 Delegate_SREC_ST_UINT32_UINT32 - .NET (C#) 304
Table 7- 388 Delegate_SREC_ST_UINT32_ERSFET - .NET (C#) 305
Table 7- 389 Definitions - Native C++ 306
Table 7- 390 Constants - .NET (C#) 306
Table 7- 391 UIP - Native C++ 307
Table 7- 392 UDataValue - Native C++ 308
Table 7- 393 SDataValue - Native C++ 309
Table 7- 394 SDataValue - .NET (C#) 310
Table 7- 395 SDVBNI - Native C++ 311
Table 7- 396 SDVBNI - .NET (C#) 311
Table 7- 397 SDataValueByAddress - Native C++ 311
Table 7- 398 SDataValueByAddress - .NET (C#) 311
Table 7- 399 SDataValueByAddressWithCheck - Native C++ 312
Table 7- 400 SDataValueByAddressWithCheck - .NET (C#) 312
Table 7- 401 SDataValueByName - Native C++ 312
Table 7- 402 SDataValueByName - .NET (C#) 312
Table 7- 403 SDataValueByNameWithCheck - Native C++ 313
Table 7- 404 SDataValueByNameWithCheck - .NET (C#) 313
Table 7- 405 SConnectionInfo - Native C++ 313
Table 7- 406 SConnectionInfo - .NET (C#) 313
Table 7- 407 SInstanceInfo - Native C++ 314
Table 7- 408 SInstanceInfo - .NET (C#) 314
Table 7- 409 SDimension - Native C++ 314
Table 7- 410 SDimension - .NET (C#) 314
Table 7- 411 STagInfo - Native C++ 315
Table 7- 412 STagInfo - .NET (C#) 316
Table 7- 413 SIP - .NET (C#) 317
Table 7- 414 SIPSuite4 - Native C++ 317
Table 7- 415 SIPSuite4 - .NET (C#) 318
Table 7- 416 SOnSyncPointReachedResult - Native C++ 319
Table 7- 417 SOnSyncPointReachedResult - .NET (C#) 320
Table 7- 418 SDataRecordInfo - Native C++ 321
Table 7- 419 SDataRecordInfo - .NET (C#) 321
Table 7- 420 SDataRecord - Native C++ 322

Table 7- 421 SDataRecord - .NET (C#) 322
Table 7- 422 SConfiguredProcessEvents - Native C++ 323
Table 7- 423 SConfiguredProcessEvents - .NET (C#) 323
Table 7- 424 SDiagExtChannelDescription - Native C++ 324
Table 7- 425 SDiagExtChannelDescription - .NET (C#) 325
Table 7- 426 Error types according to PROFINET standard 325
Table 7- 427 ExtChannelErrType error types 326
Table 7- 428 ERuntimeErrorCode - Native C++ 327
Table 7- 429 ERuntimeErrorCode - .NET (C#) 329
Table 7- 430 EArea - Native C++ 330
Table 7- 431 EArea - .NET (C#) 330
Table 7- 432 EOperatingState - Native C++ 330
Table 7- 433 EOperatingState - .NET (C#) 331
Table 7- 434 EOperatingMode - Native C++ 331
Table 7- 435 EOperatingMode - .NET (C#) 331
Table 7- 436 ECPUType - Native C++ 332
Table 7- 437 ECPUType - .NET (C#) 333
Table 7- 438 ECommunicationInterface - Native C++ 334
Table 7- 439 ECommunicationInterface - .NET (C#) 334
Table 7- 440 ELEDType - Native C++ 334
Table 7- 441 ELEDType - .NET (C#) 334
Table 7- 442 ELEDMode - Native C++ 335
Table 7- 443 ELEDMode - .NET (C#) 335
Table 7- 444 EPrimitiveDataType - Native C++ 335
Table 7- 445 EPrimitiveDataType - .NET (C#) 336
Table 7- 446 Compatible primitive data types - Reading 336
Table 7- 447 Compatible primitive data types - Write 337
Table 7- 448 EDataType - Native C++ 338
Table 7- 449 EDataType - .NET (C#) 340
Table 7- 450 ETagListDetails - Native C++ 342
Table 7- 451 ETagListDetails - .NET (C#) 342
Table 7- 452 ERuntimeConfigChanged - Native C++ 343
Table 7- 453 ERuntimeConfigChanged - .NET (C#) 343
Table 7- 454 EInstanceConfigChanged - Native C++ 343
Table 7- 455 EInstanceConfigChanged - .NET (C#) 343
Table 7- 456 EPullOrPlugEventType - Native C++ 344

Table 7- 457 EPullOrPlugEventType - .NET (C#) 344
Table 7- 458 EProcessEventType - Native C++ 344
Table 7- 459 EProcessEventType - .NET (C#) 345
Table 7- 460 EDirection - Native C++ 345
Table 7- 461 EDirection - .NET (C#) 345
Table 7- 462 EDiagProperty - Native C++ 345
Table 7- 463 EDiagProperty - .NET (C#) 345
Table 7- 464 EDiagSeverity - Native C++ 346
Table 7- 465 EDiagSeverity - .NET (C#) 346
Table 7- 466 ERackOrStationFaultType - Native C++ 346
Table 7- 467 ERackOrStationFaultType - .NET (C#) 346
Table 8- 1 CPUs with limited Motion Control resources (MCR, EMCR) during the simulation 352

Figures

Figure 2-1 Enable simulation capability 33
Figure 4-1 Local communication via Softbus 53
Figure 4-2 Local communication via TCP/IP 53
Figure 4-3 Distributed communication via Ethernet 54
Figure 4-4 Distributed communication via network adapters 55
Figure 4-5 Activate PLCSIM Virtual Switch 56
Figure 4-6 Accessible devices on the Virtual Ethernet Adapter 57
Figure 5-1 PLCSIM Advanced Symbol 59
Figure 5-2 Example: Message in the taskbar 59
Figure 5-3 Control Panel 61
Figure 5-4 Example: Download via the "PLCSIM Virtual Ethernet Adapter" (TCP/IP) after naming 65
Figure 5-5 Structure of the MAC address for an instance 67
Figure 5-6 Add card reader 71
Figure 5-7 Preview of download dialog 71
Figure 6-1 Freeze state of the virtual controller 76
Figure 6-2 Overview of the synchronization points 77
Figure 6-3 Example: Sequence in the SingleStep_CP operating mode 79
Figure 6-4 Example: Sequence in the TimespanSynchronized_CP operating mode 81
Figure 7-1 External applications and Simulation Runtime 83
Figure 7-2 Access to instances with distributed communication 84
Figure 7-3 API and external applications 85
Figure 7-4 Flowchart for the simulation of events 228
Figure 7-5 Flowchart 229
Figure 8-1 Policy exceptions for VMware vSphere Hypervisor (ESXi) 355

Guide

1

1.1 Documentation guide

The documentation for the SIMATIC S7-1500 automation system and the SIMATIC ET 200SP distributed I/O system is arranged into three areas.

Basic information

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

Device information

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

General information

The function manuals contain detailed descriptions on general topics such as 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 additions to the manuals are documented in product information sheets.

You will find the product information on the Internet:

- S7-1500 (https://support.industry.siemens.com/cs/us/en/view/68052815)
- ET 200SP (https://support.industry.siemens.com/cs/us/en/view/73021864)

Manual Collections

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

You will find the Manual Collections on the Internet:

- S7-1500 (https://support.industry.siemens.com/cs/ww/en/view/86140384)
- ET 200SP (https://support.industry.siemens.com/cs/ww/en/view/84133942)

"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 (http://support.industry.siemens.com/My/ww/en/documentation).

"mySupport" - Documentation

In the Documentation area in "mySupport" you can combine entire manuals or only parts of these to your own manual.
You can export the manual as PDF file or in a format that can be edited later.

You can find "mySupport" - Documentation 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/sc/ww/en/sc/2054).

TIA Selection Tool

The TIA Selection Tool can be used to select, configure and order devices for Totally Integrated Automation (TIA). It assembles the configuration editors for automation technology already familiar into a single tool.

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

You can find the TIA Selection Tool on the Internet (http://w3.siemens.com/mcms/topics/en/simatic/tia-selection-tool).

1.2 S7-PLCSIM products

PLCSIM Advanced V2.0, PLCSIM V15 and PLCSIM V5.x

Table 1- 1 Comparison of S7-PLCSIM products

Function	PLCSIM Advanced V2.0	PLCSIM V15	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 only	Softbus only
Supported CPU families	S7-1500 (C, T, F), ET 200SP, ET 200SP F	S7-1200 (F), S7-1500 (C, T, F), ET 200SP, ET 200SP F	S7-300, S7-300F S7-400, S7-400F
API for co-simulation[1]	✓	-	-
Web server	✓	-	-
OPC UA	✓	-	-
Process diagnostics	✓	✓	-
S7 communication	✓	Softbus	Softbus
Open user communication	✓, UDP only via TCP/IP	Softbus	-
Traces[2]	✓	(✓)	-
Motion[3]	✓	(✓)	-
Protected blocks (KHP)	✓	✓, for S7-1500 CPUs only	-
Multiple instances	Up to 16	Up to 2	-
Support of distributed instances	✓	-	-
Virtual time	✓	-	-
Connection of real CPUs/HMIs	✓	-	-
DNS usage	✓	-	-
Virtual memory card	✓	-	-
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 2.0 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 V15 in the TIA Portal; can also be monitored with PLCSIM Advanced V2.0 in the Web server.

3 With PLCSIM V15 the axes are always in simulation mode irrespective of the axis configuration.
With PLCSIM Advanced V2.0 the axes can also be operated in "Real" mode over the API.

Product overview

2

2.1 What is S7-PLCSIM Advanced?

Using 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, PLCSIM Advanced provides a full Ethernet connection and can thus also communicate distributed.

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 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 versions

The installation of PLCSIM Advanced V2.0 leads to an upgrade of an existing earlier version. However, the Runtime Manager of PLCSIM Advanced V2.0 is compatible with projects that were created with the API version V1.0. You can therefore continue to use already created projects.

Note

PLCSIM Advanced V2.0 contains the API versions V1.0 (SP1) and V2.0 and the Runtime version V2.0.

An API with higher version number cannot connect with the Runtime of an earlier version.

Table 2- 1 Compatibility of API and Runtime versions

API version	Runtime version	Supported
V1.0 (SP1)	V1.0 (SP1)	Yes
V1.0 (SP1)	V2.0	Yes
V2.0	V1.0 (SP1)	No[1]
V2.0	V2.0	Yes

1 The combination leads to a "WRONG_VERSION" error.

2.3 Security for S7-PLCSIM Advanced

Restrictions for security

Note the following restrictions when using 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 WinPcap component 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

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

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.



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.

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

PLCSIM Advanced V2.0 supports the simulation of the following CPUs:

Table 2- 2 Supported CPUs

Type	Version V1.8, V2.0, V2.1, V2.5	
Standard CPUs	CPU 1511-1 PN	CPU 1511F-1 PN
	CPU 1513-1 PN	CPU 1513F-1 PN
	CPU 1515-2 PN	CPU 1515F-2 PN
	CPU 1516-3 PN/DP	CPU 1516F-3 PN/DP
	CPU 1517-3 PN/DP[2]	CPU 1517F-3 PN/DP[2]
	CPU 1518-4 PN/DP[2]	CPU 1518F-4 PN/DP[2]
	CPU 1518-4 PN/DP ODK[2, 3]	CPU 1518F-4 PN/DP ODK[2, 3]
	CPU 1518-4 PN/DP MFP[2, 3]	CPU 1518F-4 PN/DP MFP[2, 3]
Compact CPUs[1]	CPU 1511C-1 PN	
	CPU 1512C-1 PN	
ET 200SP CPUs	CPU 1510SP-1 PN	CPU 1510SP F-1 PN
	CPU 1512SP-1 PN	CPU 1512SP F-1 PN
Technology CPUs	CPU 1511T-1 PN	CPU 1511TF-1 PN
	CPU 1515T-2 PN	CPU 1515TF-2 PN
	CPU 1516T-3 PN/DP[2]	CPU 1516TF-3 PN/DP[2]
	CPU 1517T-3 PN/DP[2]	CPU 1517TF-3 PN/DP[2]

1 The on-board I/O within the compact CPUs is not simulated. The simulation interface corresponds to the process image.

2 The simulation of this CPU only supports Motion Control Resources (MCR) and Extended Motion Control Resources (EMCR) to a limited extent.

3 The ODK functionality of this CPU is not simulated.

Unsupported CPUs

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

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

Compatibility to the TIA Portal

The following versions of PLCSIM Advanced and the TIA Portal are compatible:

Table 2- 3 Compatibility to the TIA Portal

PLCSIM Advanced version	TIA Portal version	Supported CPU version
V1.0 SP1	V14 (SP1), V15	V1.8, V2.0
V2.0	V14 (SP1), V15	V1.8, V2.0, V2.1, V2.5

Note

Firmware version

The firmware used in PLCSIM Advanced corresponds to that of a CPU S7-15xx V2.5.

You can configure a CPU with an earlier firmware version (for example V1.8) and load it onto PLCSIM Advanced. The online diagnostics then also displays the firmware version V2.5.

Downloading a CPU with a newer firmware version results in aborting.

See also

Restrictions with Motion Control (Page 352)

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

PLCSIM Advanced runs on a PC with the Windows operating system. Therefore, the scan cycle time and the exact time of actions in 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, PLCSIM Advanced as of V2.0 requires one free Core (CPU core) per instance.

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 351).

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 & diagnositics".

2.6.1 Restrictions for all supported CPUs

Bus systems

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

I/O

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

Process image partitions

Process image partitions are supported as with the real CPU.

Address spaces that are not assigned to a process image partition are updated at the cycle control point.

Update

PLCSIM Advanced does not support updating of the process image partitions for IO systems that are connected to the CPU via CMs or CPs.

Diagnostics / diagnostic alarms

With PLCSIM Advanced, simple diagnostics buffer entries can be simulated according to 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. These include, for example, the functions "Format memory card" and "Firmware update".

Status indicators LED flashing

In STEP 7, you can have the LED displayed on an CPU flash using the "Extended download to device" dialog. PLCSIM Advanced does not simulate this function.

Data logging

PLCSIM Advanced does not simulate data logging.

Recipes

PLCSIM Advanced does not simulate the use of recipes.

Copy protection

PLCSIM Advanced does not simulate copy protection.

Limited support

PLCSIM Advanced simulates some functions to a limited extent. You can find an overview in the section Restrictions (Page 347).

2.7 Simulating Motion Control

Restrictions

PLCSIM Advanced supports STEP 7 projects with configurations and functions for motion control for the CPUs S7-1500 (F), S7-1500C, S7-1500T (F) und ET 200SP (F).

PLCSIM Advanced provides only limited support for technology objects in some CPUs, see Restrictions with Motion Control (Page 352).

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

Information on "Setting in the drive and encoder connection" for actual value calculation of a virtual axis and on the topic "Virtual axis/Simulation" is available in the S7-1500T Motion Control (https://support.industry.siemens.com/cs/ww/en/view/109481326) function manual.

Installing

3

3.1 Introduction

3.1.1 System requirements

You need high-performance computer hardware if you intend to run multiple instances of PLCSIM Advanced at the same time or to simulate communication between PLCSIM Advanced V2.0 and HMI devices as of version 14.0.

Recommendation: Create up to 4 local instances on your PC or virtualization platform.

Minimum requirements for hardware

Preferably, you should install PLCSIM Advanced stand-alone on a Siemens field programming device or PC, independent of STEP 7. Alternatively, you can install PLCSIM Advanced on the configuration PC on which STEP 7 is already installed.

For PLCSIM Advanced to operate efficiently, the computer hardware must meet minimum requirements.

Table 3- 1 Minimum requirements for hardware

Hardware	Requirement
Processor	• 2.2 GHz Intel® Celeron® Dual Core for one instance • One further free core per additional instance
RAM	• 4 GB for one instance • 8 GB for 4 instances
Free hard disk space	5 GB
Screen resolution	Minimum 1024 x 768

Operating systems (64-bit versions)

PLCSIM Advanced V2.0 supports the following operating systems:

- Windows 7 Home Premium SP1
- Windows 7 Professional SP1
- Windows 7 Enterprise SP1
- Windows 7 Ultimate SP1
- Windows 10 Home Version 1703
- Windows 10 Pro Version 1703
- Windows 10 Enterprise Version 1703 (for Field PG M5)
- Windows 10 Enterprise 2016 LTSB
- Windows 10 IoT Enterprise 2016 LTSB
- Windows Server 2012 R2 StdE (full installation)
- Windows Server 2016 Standard (full installation)

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) 6.5
- VMware Workstation 12.5.5
- VMware Workstation Player 12.5.5

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/en/en/view/78788417).

3.1.2 Licenses

Floating license

PLCSIM Advanced is supplied with a floating type license. It can be stored locally and shared for a network.

Note

Validity

A license is valid for up to three instances within a PLCSIM Advanced installation.

PLCSIM Advanced V2.0 cannot be used with a V1.0 license.

PLCSIM Advanced V1.0 can be used with a V2.0 license.

Handling licenses

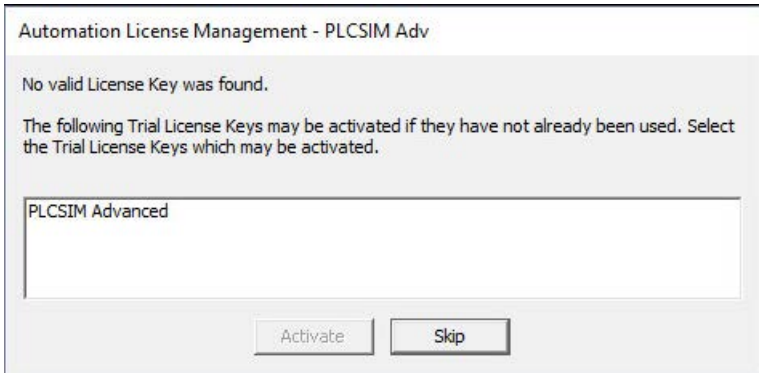
Handling of licenses is described in the Help for SIMATIC Automation License Manager (ALM).

3.1.3 Trial License

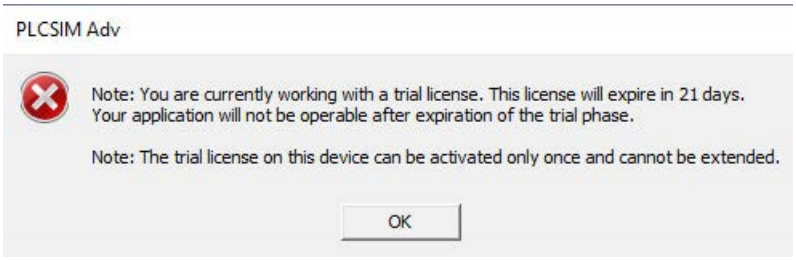
A license is available for the limited period of 21 days for S7-PLCSIM Advanced V2.0.

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.



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



Note

Remote access

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

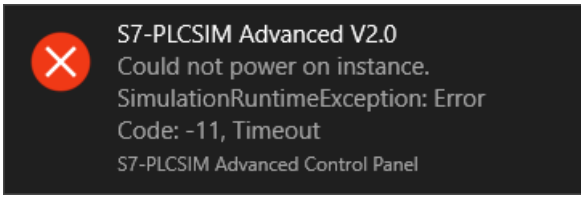
Note

Sequence of the Trial License

If no valid license is available after the Trial License has elapsed, the instance is no longer 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:



Remedy

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

API functions for licenses

- Return values for API function PowerOn() (Page 144) and callback function OnOperatingStateChanged (Page 243)
  - `SREC_OK` when a floating license is available.
  - `SREC_WARNING_TRIAL_MODE_ACTIVE` when an instance is started with the Trial License.
- Return value for callback function `OnOperatingStateChanged`
  - `SREC_LICENSE_NOT_FOUND` when the instance is automatically shut down after 21 days.

3.1.4 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_V02@<DATE_TIME>.log
- SIA_S7-PLCSIM_Advanced_V02@<DATE_TIME>_summary.log

Setup_Report (CAB file)

An archive file with the installation log and all other required files is stored in CAB format. This archive file can be found at "%autinstlog%\Reports\Setup_report.cab".

If you need help during installation, send this CAB file to SIEMENS Technical Support. Technical Support personnel can troubleshoot your installation based on the information in the CAB file.

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

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
- WinPcap

The package is available as download and on DVD.

- SIMATIC S7-PLCSIM Advanced V2.0 Floating License Download
- SIMATIC S7-PLCSIM Advanced V2.0 Floating License (DVD)

  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).

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 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.
5. 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.
6. 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".

7. The overview screen shows your installation details. Click the "Install" button. The installation then starts.
8. 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.
9. 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 "Siemens S7-PLCSIM Advanced V2.0" 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
- WinPcap

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:

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
possible 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

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 66).

API functions for selecting the communication interface

- GetCommunicationInterface() (Page 133)
- SetCommunicationInterface() (Page 133)
- CommunicationInterface { get; set; } (Page 134)

See also

Interfaces - Information and settings (Page 130)

S7 PLCSIM Advanced Control Panel (Page 60)

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 (VMware) 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.

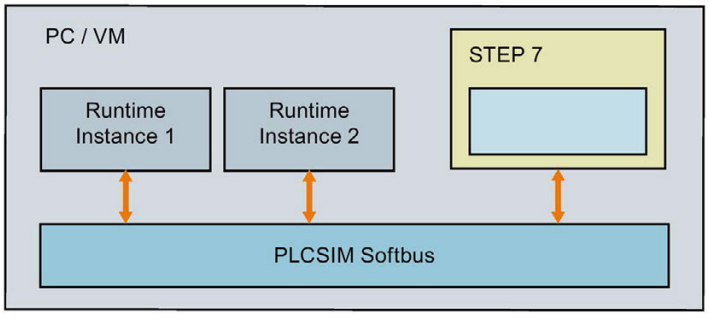


Figure 4-1 Local communication via Softbus

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.

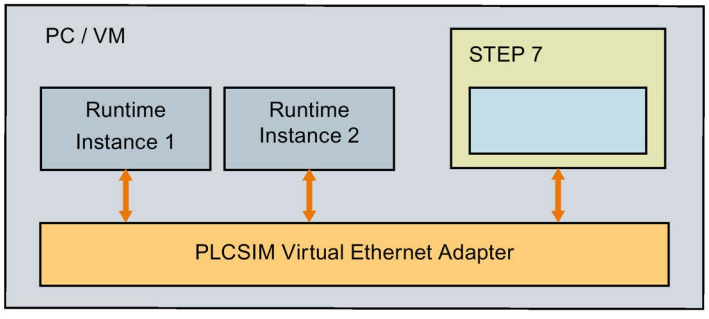


Figure 4-2 Local communication via TCP/IP

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.

Example 1: Distributed communication

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

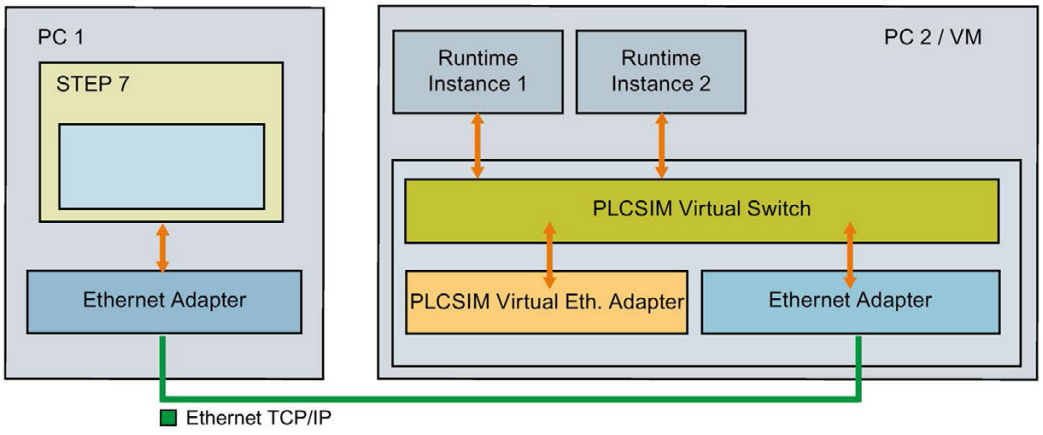


Figure 4-3 Distributed communication via Ethernet

Example 2: Distributed communication

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 network adapters.

Recommendation

Use to ensure the VMware settings as the network adapter type for the Bridged Mode to ensure error-free operation.

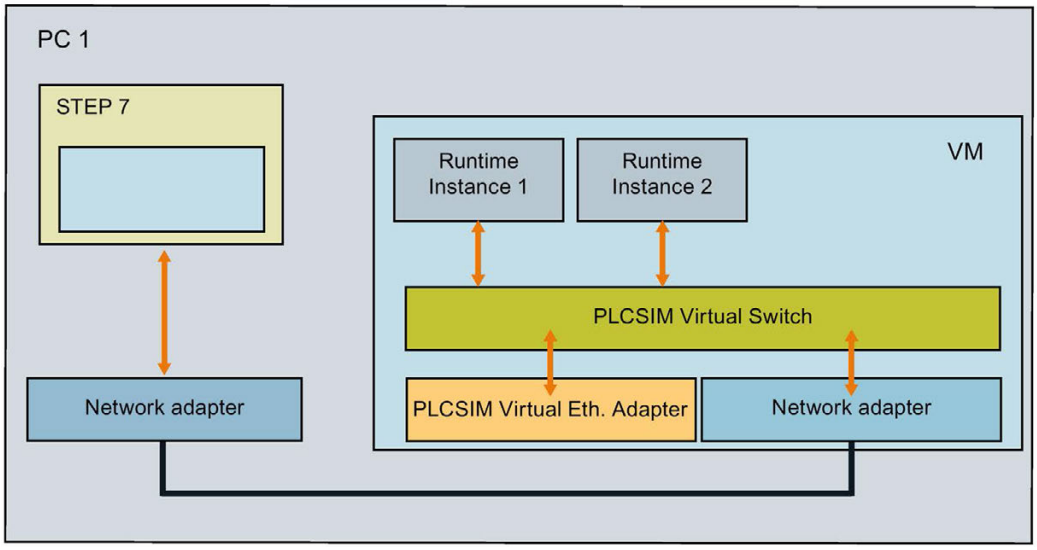


Figure 4-4 Distributed communication via network adapters

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.

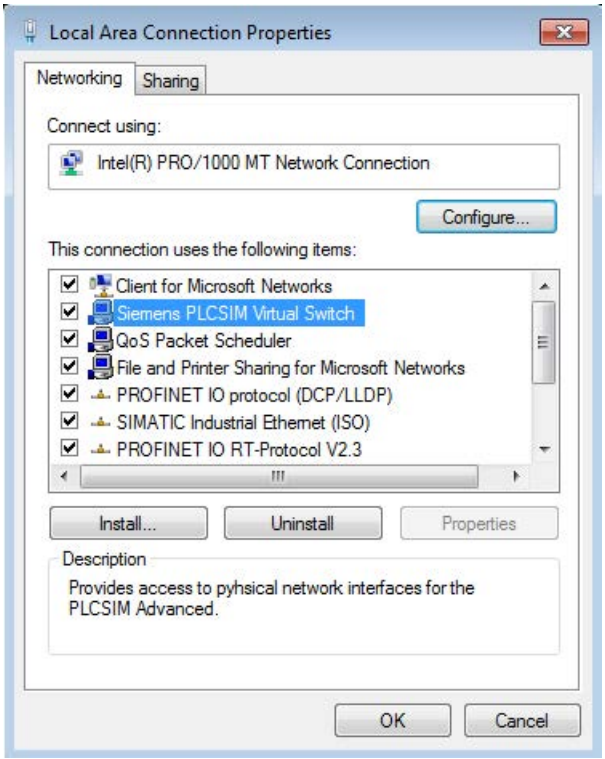


Figure 4-5 Activate PLCSIM Virtual Switch

Accessible devices

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

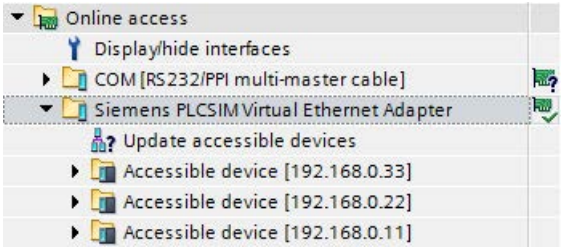


Figure 4-6 Accessible devices on the Virtual Ethernet Adapter

Distributed communication via WLAN

When using distributed communication via WLAN, it may happen that the WinPcap software installed by PLCSIM Advanced does not work with the integrated WLAN adapter of the computer. 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.

Simulate CPU 5

5.1 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 V2.0 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".

Create and activate an instance via the Control Panel

- Open PLCSIM Advanced Control Panel
- Open the "Start Virtual S7-1500 PLC" options
- Enter a name for an instance
- Select CPU type
- Create an instance using the "Start" button

In STEP 7, perform the download and start the simulation

- Download the program to the virtual controller
- Switch the controller to RUN to start the simulation
- Perform diagnostics
- ...

See also

Simulations support (Page 33)

5.2 User interface

5.2.1 S7 PLCSIM Advanced Symbol

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



Figure 5-1 PLCSIM Advanced Symbol

After double-clicking the icon, the icon appears in the taskbar of the information area.

You can use Windows functions to permanently display the icon in the information area of the taskbar.

Opening a graphical interface

Right-clicking on the icon in the taskbar opens the graphical interface of PLCSIM Advanced, the Control Panel.

If the Control Panel is open, you can use the mouse-over function to display messages about the current status of the instances.

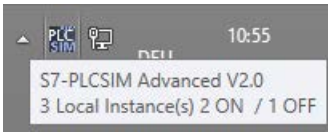
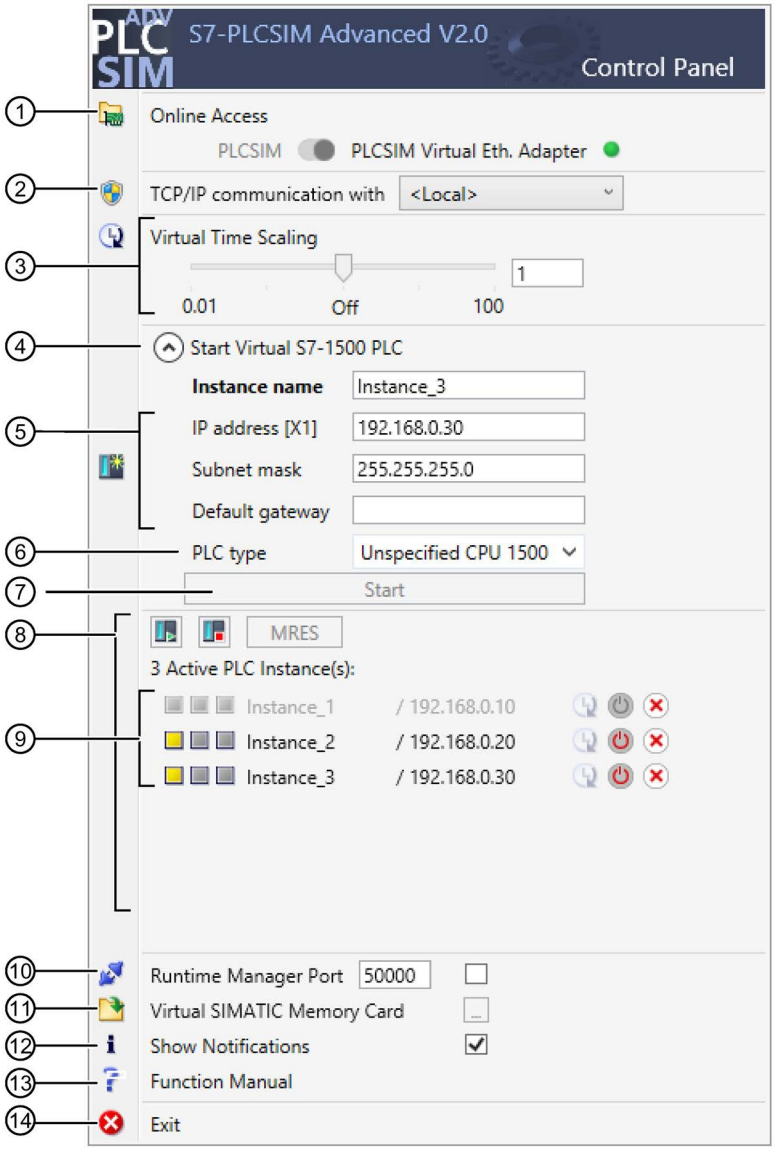


Figure 5-2 Example: Message in the taskbar

5.2.2 S7 PLCSIM Advanced Control Panel

Control Panel

The Control Panel is optional and is not needed to use PLCSIM Advanced via the API. It is available in English in version 2.0.



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	**Start Virtual S7-1500 PLC**	
	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 "Start" button is enabled.

⑤	• 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.
⑥	CPU type	Here you select the type of CPU to be simulated.
⑦	"Start" button	Use the button to create and start an instance with the entered or selected name.
⑧	Instance list	The "RUN", "STOP" and "MRES" buttons for the operating mode and memory reset. The list displays the available **local** instances of the CPUs.
⑨	LED displays	The meaning of the LED is displayed when you move the mouse over it.
⑩	Runtime Manager Port	Here you open a port on the local PC.
⑪	Virtual SIMATIC Memory Card ...	Open an Explorer window here in which you select the path to the virtual memory card.
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 switches all instances off and closes the Control Panel.

Figure 5-3 Control Panel

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

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 more information, seeVirtual time response (Page 73).

Create instance (local)

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-Type" drop-down list, select the Unspecified CPU 1500 or Unspecified ET 200SP CPU type. The CPU is named with the first download of the TIA Portal.

Recommendation: Create up to 4 local instances on your PC or virtualization platform. Remote instances cannot be created through the Control Panel.

Instance list

The list contains the active instances of the CPU that are started 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:

 Apply scaling factor for the virtual time,  disable virtual time,

 Switch on instance,  Switch off instance,

 Switch off instance and log out of the Runtime Manager

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 are stored on the Virtual SIMATIC Memory Card. Use the button to open an Explorer window in which you select the path to the virtual memory card or in which the path is displayed.

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

- The command switches off all local instances on the PC or the VM and logs them off from the Runtime Manager.
- This command closes the Runtime Manager if there is 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.

Minimize Control Panel

Clicking on an empty area on the desktop minimizes the Control Panel. The instances are not affected.

5.3 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.

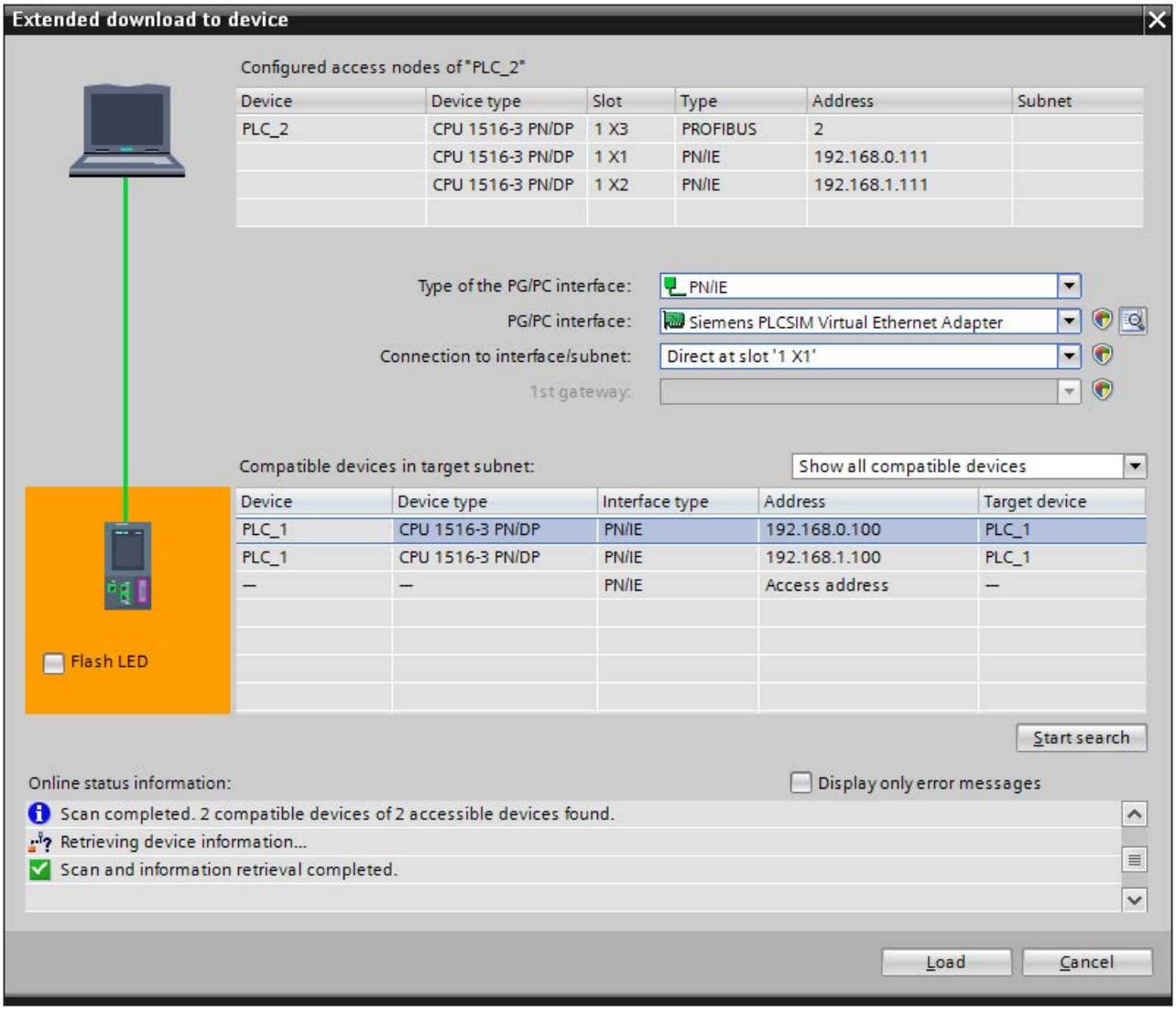


Figure 5-4 Example: Download via the "PLCSIM Virtual Ethernet Adapter" (TCP/IP) after naming

5.4 Network addresses in the simulation

5.4.1 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.

5.4.2 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:

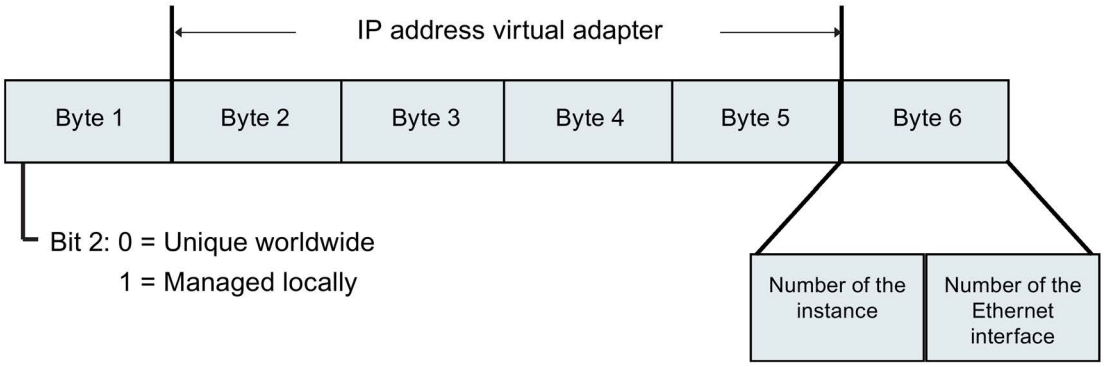


Figure 5-5 Structure of the MAC address for an instance

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 V2.0. 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.

5.5 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 356)

5.6 Simulate communication

5.6.1 Communication services that can be simulated

PLCSIM Advanced V2.0 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, 3] • T_DISCON
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	• MB_CLIENT • MB_SERVER
E-mail[2, 3]	• TMAIL_C
S7 communication	• PUT / GET • BSEND / BRCV • USEND / URCV
OPC UA Server[2]	Data exchange with OPC UA clients
Web server[2, 3]	Data exchange via HTTP

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 Secure TCP communication is not supported.

Special conditions apply when communicating with TUSEND/TURCV, see Restrictions for communications services (Page 351).

TMAIL_C

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

Remedy

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

5.6.2 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.

5.7 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.

Provide project data offline

1. Create a "User-defined Card Reader" for your project data under Card Reader/USB storage 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.

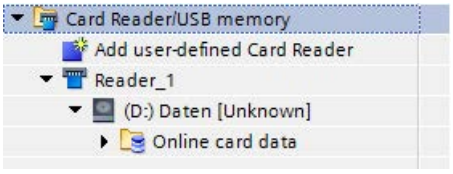


Figure 5-6 Add card reader

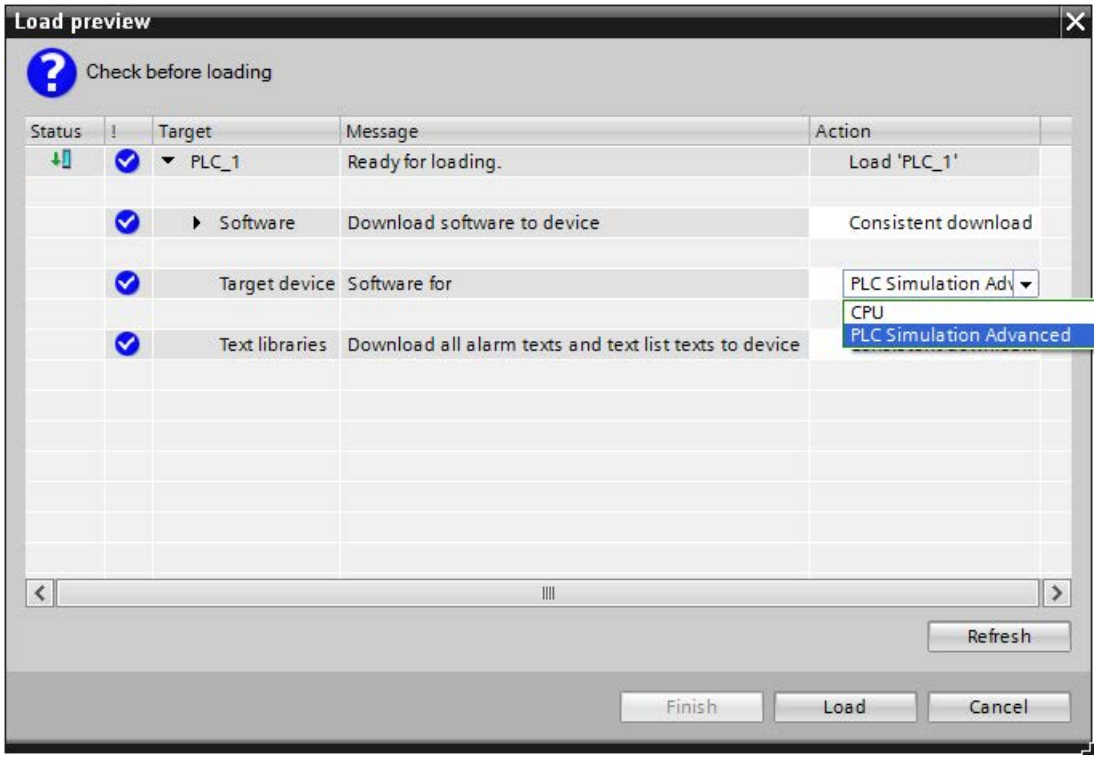


Figure 5-7 Preview of download dialog

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 140)
- StoragePath { get; set; } (Page 141)

See also

Controller - Information and settings (Page 136)

Virtual time response

6

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.

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 220)
- SetSystemTime() (Page 221)
- SystemTime { get; set; } (Page 221)

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.

- 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 221)
- SetScaleFactor() (Page 222)
- ScaleFactor { get; set; } (Page 223)

See also

Settings for the virtual time (Page 220)

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.

Fast forward

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

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.

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 221)
- SetScaleFactor() (Page 222)
- ScaleFactor { get; set; } (Page 223)

See also

Cycle control (Page 223)

Settings for the virtual time (Page 220)

Error with overflow of cyclical events and motion OBs (Page 355)

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.

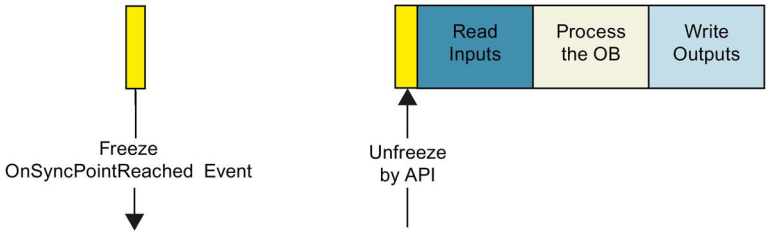


Figure 6-1 Freeze state of the virtual controller

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

Cancelation of the freeze state during downloading

During a download the freeze state is canceled. After completion of the download the virtual controller at the synchronization point is reset to the freeze state.

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 330).

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.

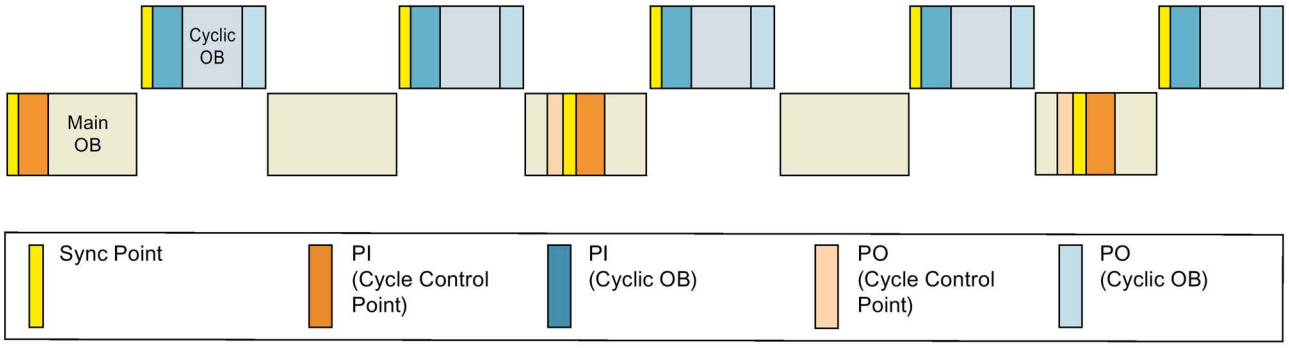


Figure 6-2 Overview of the synchronization points

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 78).

- TimespanSynchronized operating modes

  See Synchronize simulation partner time-controlled (Page 80).

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

API functions

- Settings for the cycle control (Page 223)
- GetOperatingMode() (Page 223)
- SetOperatingMode() (Page 223)
- OperatingMode { get; set; } (Page 224)
- EOperatingMode (Page 331)

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]
	Cycle control point	Before reading in the process image partition	
	"C"	"P"	"T"
`SingleStep_C`	✓		
`SingleStep_P`		✓	
`SingleStep_CP`	✓	✓	
`SingleStep_CT`	✓		✓
`SingleStep_CPT`	✓	✓	✓

[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.

API functions / events

- GetOverwrittenMinimalCycleTime_ns() (Page 225)
- SetOverwrittenMinimalCycleTime_ns() (Page 225)
- OverwrittenMinimalCycleTime_ns { get; set; } (Page 226)
- RunToNextSyncPoint() (Page 226)
- OnSyncPointReached (Page 247)
- EventCallback_II_SREC_ST_UINT32_INT64_INT64_UINT32 (Page 284) / Delegate_II_EREC_DT_UINT32_INT64_INT64_UINT32 (Page 296)

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.

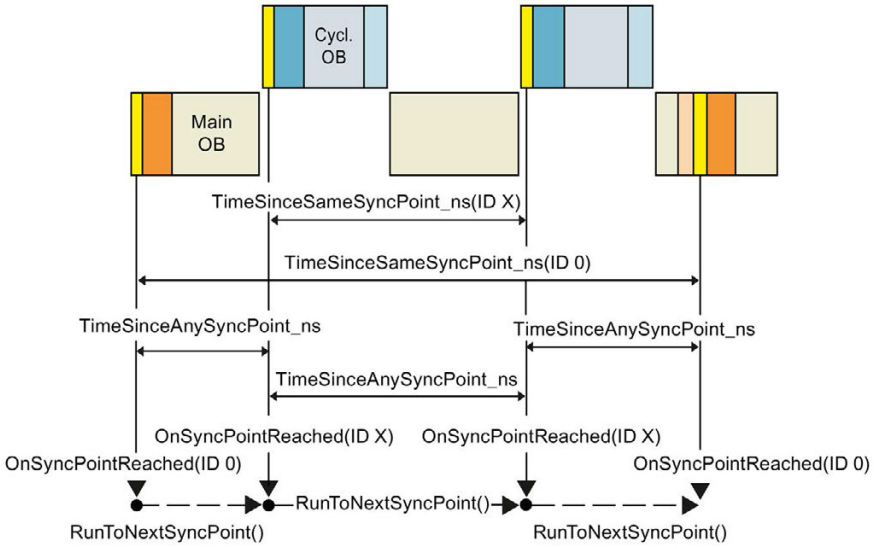


Figure 6-3 Example: Sequence in the SingleStep_CP operating mode

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 223)

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 223)
- StartProcessing() (Page 227)
- OnSyncPointReached (Page 247)

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.

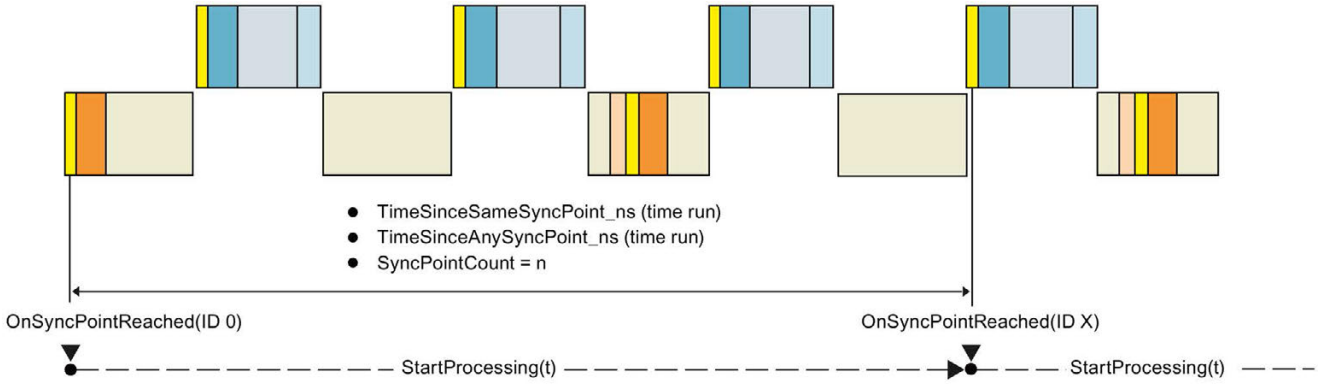


Figure 6-4 Example: Sequence in the TimespanSynchronized_CP operating mode

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:

- Runtime
  - Siemens.Simatic.Simulation.Runtime.Manager.exe
  - Siemens.Simatic.Simulation.Runtime.Instance.exe
- Libraries
  - Siemens.Simatic.Simulation.Runtime.Api.x86.dll
  - Siemens.Simatic.Simulation.Runtime.Api.x64.dll
  - SimulationRuntimeApi.h
- Documentation of interfaces with examples in native C++ and .NET (Managed Code)

Description

Runtime and libraries	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.

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 other simulation software or a graphical user interface (GUI), for example.

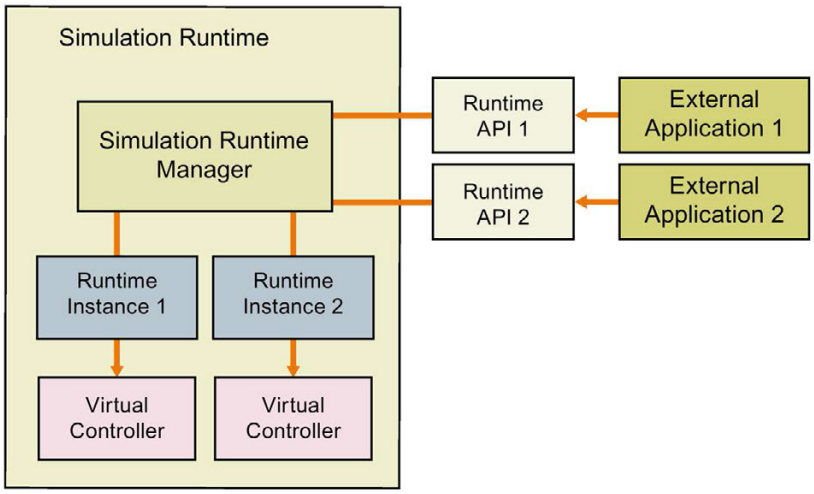
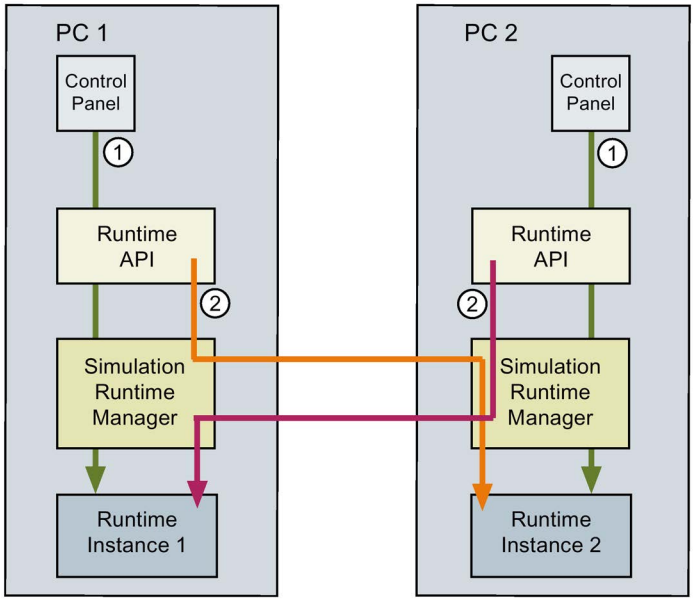


Figure 7-1 External applications and Simulation Runtime

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 7-7 Overview of IRemoteRuntimeManager functions - Native C++ (Page 89)
- Table 7-14 Overview of IRemoteRuntimeManager functions - .NET (C#) (Page 92)
- Table 7-5 Overview of IInstances functions - Native C++ (Page 87)
- Table 7-12 Overview of IInstances functions - .NET (C#) (Page 91)

See also

Overview of user interfaces for managed code (Page 90)

S7 PLCSIM Advanced Control Panel (Page 60)

7.1.2 User interfaces (API)

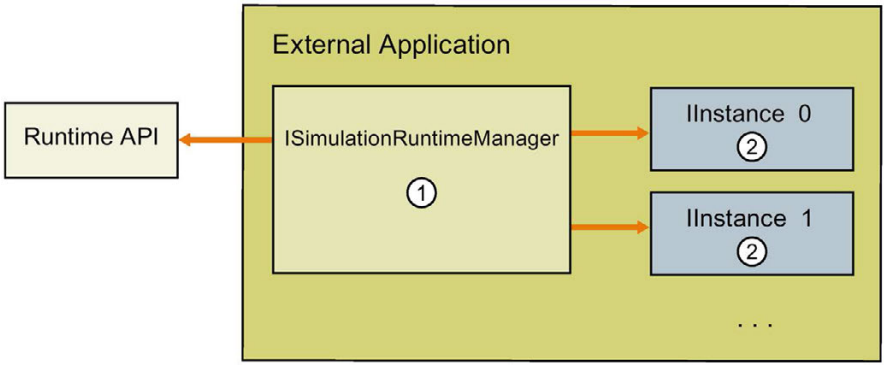
The user interfaces of Simulation Runtime include functions you use, for example, to create instances, to change the operating state of a virtual controller, or to exchange I/O data.

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 86)
- Overview of data types for native C++ (Page 93)
- Overview of user interfaces for managed code (Page 90)
- Overview of data types for managed code (Page 94)

Note

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

7.1.3 Overview of user interfaces for native C++

Initializing and shutting down API

Table 7- 1 Overview of initializing and shutting down API - Native C++

Actions	Functions
Initialize API	InitializeApi (Page 96) RuntimeApiEntry_Initialize (Page 98)
Shut down API (Page 100)	`DestroyInterface` `RuntimeApiEntry_DestroyInterface`
Logging off API library (Page 103)	`FreeApi` `ShutdownAndFreeApi`

Global functions

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

Actions	Functions

Global functions (Page 106)	`GetNameOfAreaSection()`
	`GetNameOfCPUType()`
	`GetNameOfCommunicationInterface()`
	`GetNameOfDataType()`
	`GetNameOfLEDMode()`
	`GetNameOfLEDType()`
	`GetNameOfOperatingMode()`
	`GetNameOfOperatingState()`
	`GetNameOfPrimitiveDataType()`
	`GetNameOfTagListDetails()`
	`GetNameOfErrorCode()`
	`GetNameOfRuntimeConfigChanged()`
	`GetNameOfInstanceConfigChanged()`
	`GetNameOfDiagSeverity()`
	`GetNameOfDirection()`
	`GetNameOfRackOrStationFaultType()`

API ISimulationRuntimeManager

Table 7- 3 Overview of API ISimulationRuntimeManager functions - Native C++

Settings and information	Functions

Interface (Page 111)	`GetVersion()`
	`IsInitialized()`
	`IsRuntimeManagerAvailable()`
	`Shutdown()`

Simulation Runtime instances (Page 113)	`GetRegisteredInstancesCount()`
	`GetRegisteredInstanceInfoAt()`
	`RegisterInstance()`
	`RegisterCustomInstance()`
	`CreateInterface()`

Remote connections (Page 119)	`OpenPort()`
	`ClosePort()`
	`GetPort()`
	`GetRemoteConnectionsCount()`
	`GetRemoteConnectionInfoAt()`
	`RemoteConnect()`

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

Events	Functions

OnConfigurationChanged (Page 252)	`RegisterOnConfigurationChangedCallback()`
	`UnregisterOnConfigurationChangedCallback()`
	`RegisterOnConfigurationChangedEvent()`
	`UnregisterOnConfigurationChangedEvent()`
	`WaitForOnConfigurationChangedEvent()`

OnRuntimeManagerLost (Page 127)	`RegisterOnRuntimeManagerLostCallback()`
	`UnregisterOnRuntimeManagerLostCallback()`
	`RegisterOnRuntimeManagerLostEvent()`
	`UnregisterOnRuntimeManagerLostEvent()`
	`WaitForOnRuntimeManagerLostEvent()`

API IInstances

Table 7- 5 Overview of IInstances functions - Native C++

Settings and information	Functions

Interface (Page 130)	`GetID()`
	`GetName()`
	`GetCPUType()`
	`SetCPUType()`
	`GetCommunicationInterface()`
	`SetCommunicationInterface()`
	`GetInfo()`
	`UnregisterInstance()`

Controller (Page 136)	`GetControllerName()`
	`GetControllerShortDesignation()`
	`GetControllerIPCount()`
	`GetControllerIP()`
	`GetControllerIPSuite4()`
	`SetIPSuite()`
	`GetStoragePath()`
	`SetStoragePath()`
	`ArchiveStorage()`
	`RetrieveStorage()`

Settings and information	Functions

Operating state (Page 144)	`GetOperatingState()`
	`PowerOn()`
	`PowerOff()`
	`MemoryReset()`
	`Run()`
	`Stop()`

Tag list (Page 152)	`UpdateTagList()`
	`GetTagListStatus()`
	`GetTagInfoCount()`
	`GetTagInfos()`
	`CreateConfigurationFile()`

I/O access via address - Reading (Page 158)	`GetAreaSize()`
	`ReadBit()`
	`ReadByte()`
	`ReadBytes()`
	`ReadSignals()`

I/O access via address - Writing (Page 166)	`WriteBit()`
	`WriteByte()`
	`WriteBytes()`
	`WriteSignals()`

I/O access via tag name - Reading (Page 173)	`Read()`
	`ReadBool()`
	`ReadChar(), ReadWChar()`
	`ReadDouble()`
	`ReadFloat()`
	`ReadInt8(), ReadInt16(), ReadInt32(), ReadInt64()`
	`ReadUInt8(), ReadUInt16(), ReadUInt32(), ReadUInt64()`
	`ReadSignals()`

I/O access via tag name - Writing (Page 191)	`Write()`
	`WriteBool()`
	`WriteChar(), WriteWChar()`
	`WriteDouble()`
	`WriteFloat()`
	`WriteInt8(), WriteInt16(), WriteInt32(), WriteInt64(),`
	`WriteUInt8(),WriteUInt16(), WriteUInt32(), WriteUInt64()`
	`WriteSignals()`

Virtual time (Page 220)	`GetSystemTime()`
	`SetSystemTime()`
	`GetScaleFactor()`
	`SetScaleFactor()`

Cycle control (Page 223)	`GetOperatingMode()`
	`SetOperatingMode()`
	`SetSendSyncEventInDefaultModeEnabled()`
	`IsSendSyncEventInDefaultModeEnabled`
	`GetOverwrittenMinimalCycleTime_ns()`
	`SetOverwrittenMinimalCycleTime_ns()`
	`RunToNextSyncPoint()`
	`StartProcessing()`

Acyclic services (Page 228)	

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

Events	Functions

OnOperatingStateChanged (Page 243)	`RegisterOnOperatingStateChangedCallback()`
	`UnregisterOnOperatingStateChangedCallback()`
	`RegisterOnOperatingStateChangedEvent()`
	`UnregisterOnOperatingStateChangedEvent()`
	`WaitForOnOperatingStateChangedEvent()`

OnSyncPointReached (Page 247)	`RegisterOnSyncPointReachedCallback()`
	`UnregisterOnSyncPointReachedCallback()`
	`RegisterOnSyncPointReachedEvent()`
	`UnregisterOnSyncPointReachedEvent()`
	`WaitForOnSyncPointReachedEvent()`

OnConfigurationChanging (Page 249)	`RegisterOnConfigurationChangingCallback()`
	`UnregisterOnConfigurationChangingCallback()`
	`RegisterOnConfigurationChangingEvent()`
	`UnregisterOnConfigurationChangingEvent()`
	`WaitForOnConfigurationChangingEvent()`

OnConfigurationChanged (Page 252)	`RegisterOnConfigurationChangedCallback()`
	`UnregisterOnConfigurationChangedCallback()`
	`RegisterOnConfigurationChangedEvent()`
	`UnregisterOnConfigurationChangedEvent()`
	`WaitForOnConfigurationChangedEvent()`

OnLedChanged (Page 254)	`RegisterOnLedChangedCallback()`
	`UnregisterOnLedChangedCallback()`
	`RegisterOnLedChangedEvent()`
	`UnregisterOnLedChangedEvent()`
	`WaitForOnLedChangedEvent()`

API IRemoteRuntimeManager

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

Settings and information	Functions

Interface (Page 265)	`GetVersion()`
	`GetIP()`
	`GetPort()`
	`GetRemoteComputerName()`
	`Disconnect()`

Simulation Runtime instances (Page 268)	`GetRegisteredInstancesCount()`
	`GetRegisteredInstanceInfoAt()`
	`RegisterInstance()`
	`RegisterCustomInstance()`
	`CreateInterface()`

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

Events	Functions

OnConnectionLost (Page 275)	`RegisterOnConnectionLostCallback()`
	`UnregisterOnConnectionLostCallback()`
	`RegisterOnConnectionLostEvent()`
	`UnregisterOnConnectionLostEvent()`
	`WaitForOnConnectionLostEvent()`

7.1.4 Overview of user interfaces for managed code

Initializing and shutting down API

Table 7- 9 Overview of initializing and shutting down API - .NET (C#)

Actions	Functions
Initialize API (Page 99)	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeManager`
Shut down API (Page 105)	

API ISimulationRuntimeManager

Table 7- 10 Overview of ISimulationRuntimeManager functions - .NET (C#)

Settings and information	Functions

Interface (Page 111)	`Version { get; }`
	`IsInitialized { get; }`
	`IsRuntimeManagerAvailable { get; }`
	`Shutdown()`

Simulation Runtime instances (Page 113)	`RegisterInstanceInfo { get; }`
	`RegisterInstance()`
	`RegisterCustomInstance()`
	`CreateInterface()`

Remote connections (Page 119)	`OpenPort()`
	`ClosePort()`
	`Port { get; }`
	`RemoteConnectionInfo { get; }`
	`RemoteConnect()`

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

Events	Functions

OnConfigurationChanged (Page 124)	`OnConfigurationChanged`
	`RegisterOnConfigurationChangedEvent()`
	`UnregisterOnConfigurationChangedEvent()`
	`WaitForOnConfigurationChangedEvent()`

OnRuntimeManagerLost (Page 127)	`OnRuntimeManagerLost()`
	`RegisterOnRuntimeManagerLostEvent()`
	`UnregisterOnRuntimeManagerLostEvent()`
	`WaitForOnRuntimeManagerLostEvent()`

API IInstances

Table 7- 12 Overview of IInstances functions - .NET (C#)

Settings and information	Functions

Interface (Page 130)	`Dispose ()`
	`ID { get; }`
	`Name { get; }`
	`CPUType { get; set; }`
	`CommunicationInterface { get; }`
	`Info { get; }`
	`UnregisterInstance()`

Controller - Information and settings (Page 136)	`ControllerName { get; }`
	`ControllerShortDesignation { get; }`
	`ControllerIPSuite4 { get; }`
	`SetIPSuite()`
	`StoragePath { get; set; }`
	`ArchiveStorage()`
	`RetrieveStorage()`

Operating state (Page 144)	`OperatingState { get; }`
	`PowerOn()`
	`PowerOff()`
	`MemoryReset()`
	`Run()`
	`Stop()`

Tag list (Page 152)	`UpdateTagList()`
	`GetTagListStatus()`
	`TagInfos { get; }`
	`CreateConfigurationFile()`

I/O access via address - Reading (Page 158)	`InputArea | MarkerArea | OutputArea { get; }`
	`AreaSize { get; }`
	`ReadBit()`
	`ReadByte()`
	`ReadBytes()`
	`ReadSignals()`

I/O access via address - Writing (Page 166)	`WriteBit()`
	`WriteByte()`
	`WriteBytes()`
	`WriteSignals()`

I/O access via tag name - Reading (Page 173)	`Read()`
	`ReadBool()`
	`ReadChar(), ReadWChar()`
	`ReadDouble()`
	`ReadFloat()`
	`ReadInt8(), ReadInt16(), ReadInt32(), ReadInt64()`
	`ReadUInt8(), ReadUInt16(), ReadUInt32(), ReadUInt64()`
	`ReadSignals()`

I/O access via tag name - Writing (Page 191)	`Write()`
	`WriteBool()`
	`WriteChar(), WriteWChar()`
	`WriteDouble()`
	`WriteFloat()`
	`WriteInt8(), WriteInt16(), WriteInt32(), WriteInt64(),`
	`WriteUInt8(),WriteUInt16(), WriteUInt32(), WriteUInt64()`
	`WriteSignals()`

Virtual time (Page 220)	`SystemTime { get; set; }` `ScaleFactor { get; set; }`

Cycle control (Page 223)	`OperatingMode { get; set; }`
	`IsSendSyncEventInDefaultModeEnabled { get; set; }`
	`OverwrittenMinimalCycleTime_ns { get; set; }`
	`RunToNextSyncPoint`
	`StartProcessing()`

Acyclic services (Page 228)	

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

Events	Functions

OnOperatingStateChanged (Page 243)	`OnOperatingStateChanged`
	`RegisterOnOperatingStateChangedEvent()`
	`UnregisterOnOperatingStateChangedEvent()`
	`WaitForOnOperatingStateChangedEvent()`

OnSyncPointReached (Page 247)	`OnSyncPointReached`
	`RegisterOnSyncPointReachedEvent()`
	`UnregisterOnSyncPointReachedEvent()`
	`WaitForOnSyncPointReachedEvent()`

OnConfigurationChanging (Page 249)	`OnConfigurationChanging`
	`RegisterOnConfigurationChangingEvent()`
	`UnregisterOnConfigurationChangingEvent()`
	`WaitForOnConfigurationChangingEvent()`

OnConfigurationChanged (Page 252)	`OnConfigurationChanged`
	`RegisterOnConfigurationChangedEvent()`
	`UnregisterOnConfigurationChangedEvent()`
	`WaitForOnConfigurationChangedEvent()`

OnLedChanged (Page 254)	`OnLedChanged`
	`RegisterOnLedChangedEvent()`
	`UnregisterOnLedChangedEvent()`
	`WaitForOnLedChangedEvent()`

API IRemoteRuntimeManager

Table 7- 14 Overview of IRemoteRuntimeManager functions - .NET (C#)

Settings and information	Functions

Interface (Page 265)	`Dispose()`
	`Version { get; }`
	`IP { get; }`
	`Port { get; }`
	`RemoteComputerName { get; }`
	`Disconnect()`

Simulation Runtime instances (Page 113)	`RegisterInstanceInfo { get; }`
	`RegisterInstance()`
	`RegisterCustomInstance()`
	`CreateInterface()`

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

Events	Functions

OnConnectionLost() (Page 275)	`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- 16 Overview of data types - Native C++

Data type	
DLL import functions (Page 280)	ApiEntry_Initialize ApiEntry_DestroyInterface
Event callback functions (Page 281)	EventCallback_VOID EventCallback_II_SREC_ST EventCallback_II_SREC_ST_SROS_SROS EventCallback_II_SREC_ST_SRLT_SRLM EventCallback_IRRTM EventCallback_SRCC_UINT32_UINT32_INT32 EventCallback_II_SREC_ST_SRICC_UINT32_UINT32_UINT32_UINT32 New in V2.0: EventCallback_II_SREC_ST_SDRI EventCallback_II_SREC_ST_SDR EventCallback_II_SREC_ST_UINT32_INT64_INT64_UINT32 EventCallback_II_SREC_ST_UINT32_EPPET_UINT32 EventCallback_II_SREC_ST_UINT32_UINT32_EPET_UINT32 EventCallback_II_SREC_ST_UINT32 EventCallback_II_SREC_ST_UINT32_UINT32 EventCallback_II_SREC_ST_UINT32_ERSFET
Definitions and constants (Page 306)	
Unions (Page 307)	UIP UDataValue
Structures (Page 309)	SDataValue SDVBNI SDataValueByAddress SDataValueByName SConnectionInfo SInstanceInfo SDimension STagInfo SIP SIPSuite4 SOnSyncPointReachedResult New in V2.0: SDataValueByAddressWithCheck SDataValueByNameWithCheck SDataRecordInfo SDataRecord SConfiguredProcessEvents SDiagExtChannelDescription

Data type	
Enumerations (Page 327)	ERuntimeErrorCode EArea EOperatingState EOperatingMode ECPUType ECommunicationInterface ELEDType ELEDMode EPrimitiveDataTypes EDataType ETagListDetails ERuntimeConfigChanged EInstanceConfigChanged New in V2.0: EPullOrPlugEventType EProcessEventType EDirection EDiagProperty EDiagSeverity ERackOrStationFaultType

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- 17 Overview of data types - .NET (C#)

Data type	
Delegate definitions (Page 293) - Event handler methods	Delegate_Void Delegate_II_EREC_DT Delegate_II_EREC_DT_EOS_EOS Delegate_II_EREC_DT_ELT_ELM Delegate_IRRTM Delegate_SRCC_UINT32_UINT32_INT32 Delegate_II_EREC_DT_SRICC_UINT32_UINT32_UINT32_UINT32 New in V2.0: Delegate_II_EREC_DT_UINT32_INT64_INT64_UINT32 Delegate_II_EREC_DT_SDRI Delegate_II_EREC_DT_SDR Delegate_SREC_ST_UINT32_EPPET_UINT32 Delegate_SREC_ST_UINT32_UINT32_EPET_UINT32 Delegate_SREC_ST_UINT32 Delegate_SREC_ST_UINT32_UINT32 Delegate_SREC_ST_UINT32_ERSFET
Definitions and constants (Page 306)	

Data type	
Structures (Page 309)	SDataValue SDVBNI SDataValueByAddress SDataValueByName SConnectionInfo SInstanceInfo SDimension STagInfo SIP SIPSuite4 SOnSyncPointReachedResult New in V2.0: SDataValueByAddressWithCheck SDataValueByNameWithCheck SDataRecordInfo SDataRecord SConfiguredProcessEvents SDiagExtChannelDescription
Enumerations (Page 326)	ERuntimeErrorCode EArea EOperatingState EOperatingMode ECPUType ECommunicationInterface ELEDType ELEDMode EPrimitiveDataTypes EDataType ETagListDetails ERuntimeConfigChanged EInstanceConfigChanged New in V2.0: EPullOrPlugEventType EProcessEventType EDirection EDiagProperty EDiagSeverity ERackOrStationFaultType

7.2 Initialize API

7.2.1 Load API library

Description

With PLCSIM Advanced the interfaces of the API V2.0 are not compatible with the interfaces of the API V1.0. However, the Runtime Manager of PLCSIM Advanced V2.0 is compatible with the API of PLCSIM Advanced V1.0.

Earlier versions of the API are also installed during the installation of PLCSIM Advanced V2.0.

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

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\2.0").

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 96).
- 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- 18 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 279). • `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 31).
	`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.

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 101).

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- 19 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 279). • `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 31).
	`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.

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 101).

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- 20 Initialize - .NET (C#)

Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationInitializationException`	
	Runtime error code	Condition
	`ERuntimeError-Code.ConnectionError`	Unable to establish a connection to the Runtime Manager.
	`ERuntimeError-Code.WrongVersion`	The version of the API is not compatible with Runtime. See Compatibility during upgrade (Page 31).
	`ERuntimeError-Code.RuntimeNotAvailable`	No Runtime Manager runs in this Windows user session.

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 103)
- ShutdownAndFreeApi() (Page 104)

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- 21 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++	`// 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 101) function.

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

Table 7- 22 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_DestroyInterface)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- 23 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- 24 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 130) 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 130)
2. Call the Shutdown() (Page 111) function of the ISimulationRuntimeManager interface.

7.4 Global functions (Native C++)

The global functions `GetNameOf...` return the name of the enumeration entry.

GetNameOfAreaSection()

Table 7- 25 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- 26 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- 27 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- 28 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- 29 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- 30 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- 31 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- 32 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- 33 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- 34 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- 35 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- 36 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- 37 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- 38 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- 39 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- 40 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- 41 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

See also

EPrimitiveDataType (Page 335)

EDataType (Page 338)

Enumerations (Page 326)

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- 42 GetVersion() - Native C++

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

Table 7- 43 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- 44 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- 45 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 124).

Table 7- 46 IsRuntimeManagerAvailable() - Native C++

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

Table 7- 47 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- 48 Shutdown() - Native C++

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

Table 7- 49 Shutdown() - .NET (C#)

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

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- 50 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- 51 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 306).	
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- 52 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- 53 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 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 306). • `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 101).

Table 7- 54 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 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 306).	
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.
	`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- 55 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 306). • `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("C:\\Temp\\vplc.dll");`
	`}`	

Note

Native C++

If you no longer require the interface, delete it.

See DestroyInterface() (Page 101).

Table 7- 56 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 306).	
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.
	`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- 57 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 101).

Table 7- 58 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.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.

7.5.3 Remote connections

OpenPort()

Opens a port to which another Runtime Manager can connect.

Table 7- 59 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- 60 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.
	`ERuntimeErrorCode.AlreadyExists`	A port is already open.
	`ERuntimeErrorCode.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- 61 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- 62 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- 63 GetPort() - Native C++

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

Table 7- 64 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- 65 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- 66 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- 67 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- 68 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 31).

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 101).

Table 7- 69 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
	`ERuntimeError-Code.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 31).

See also

Tag list (Page 152)
Data types (Page 279)

7.5.4 Events

Events for the ISimulationRuntimeManager interface

The following events are differentiated in the API:

Table 7- 70 Events for the ISimulationRuntimeManager interface

Event	Cause
OnConfigurationChanged (Page 124)	This event is triggered at a change of the Runtime Manager configuration. For example, when a new instance is registered, an instance is removed or a connection to a client is established. The Control Panel uses such an event to update the list of available instances.
OnRuntimeManagerLost (Page 127)	The event is triggered when the connection to Runtime Manager is interrupted.

7.5.4.1 OnConfigurationChanged events

OnConfigurationChanged

Registers or unregisters an event handler method.

Table 7- 71 OnConfigurationChanged - .NET (C#)

Syntax	`event Delegate_SRCC_UINT32_UINT32_INT32 OnConfigurationChanged;`
Parameters	None. See Delegate_SRCC_UINT32_UINT32_INT32 (Page 297).
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- 72 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 285).
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- 73 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- 74 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- 75 UnregisterOnConfigurationChangedCallback() - Native C++

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

UnregisterOnConfigurationChangedEvent()

Unregisters the event object.

Table 7- 76 UnregisterOnConfigurationChangedEvent() - Native C++

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

Table 7- 77 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- 78 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- 79 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- 80 OnRuntimeManagerLost - .NET (C#)

Syntax	`event Delegate_Void OnRuntimeManagerLost;`
Parameters	None. See Delegate_Void (Page 293).
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- 81 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 281).
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- 82 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- 83 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- 84 UnregisterOnRuntimeManagerLostCallback() - Native C++

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

UnregisterOnRuntimeManagerLostEvent()

Unregisters the event object.

Table 7- 85 UnregisterOnRuntimeManagerLostEvent() - Native C++

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

Table 7- 86 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- 87 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- 88 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.

7.6 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- 89 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- 90 GetID() - Native C++

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

Table 7- 91 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- 92 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 306). • `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++	`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- 93 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- 94 GetCPUType() - Native C++

Syntax	`ECPUType GetCPUType();`
Parameters	None
Return values	An enumeration element that defines the CPU type. See ECPUType (Page 332).

SetCPUType()

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

Table 7- 95 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- 96 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- 97 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- 98 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- 99 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- 100 GetInfo() - Native C++

Syntax	`SInstanceInfo GetInfo();`
Parameters	None
Return values	`SInstanceInfo`: A structure that provides information about the instance. See SInstanceInfo (Page 314).

Table 7- 101 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- 102 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- 103 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- 104 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- 105 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- 106 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- 107 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- 108 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- 109 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- 110 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- 111 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- 112 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- 113 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 completed the boot process or is already in the shutdown phase.

Table 7- 114 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
	`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.DoesNotExist`	There is no network interface with this ID.
	`ERuntimeError-Code.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- 115 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 279). • `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- 116 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 279).	
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- 117 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.
	`ERuntimeErrorCode.WrongArgument`	The path contains invalid characters.

ArchiveStorage()

The user program, the hardware configuration and the retentive data are 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- 118 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.

Table 7- 119 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	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.InvalidOperatingState`	The instance is not in OFF operating state.
	`ERuntimeError-Code.InvalidArchivePath`	The archive path is invalid.
	`ERuntimeError-Code.CreateDirectoriesFailed`	The directory for the ZIP file could not be created.
	`ERuntimeError-Code.ArchiveStorageNotCreated`	The ZIP file could not be created.
	`ERuntimeError-Code.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- 120 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.

Table 7- 121 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	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.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.

7.6.3 Operating state

GetOperatingState() / OperatingState { get; }

Returns the operating state of the virtual controller. When the operating state changes, the OnOperatingStateChanged() (Page 243) event is triggered. For details about the operating state, see Data types (Page 330).

Table 7- 122 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- 123 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.

PowerOn()

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

Table 7- 124 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`	The path could not be created. The length of the `DSTORAGE_PATH_MAX_LENGTH` characters may be exceeded.
	`SREC_WARNING_ALREADY_EXISTS`	Warning. The instance is started.
	`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_VIRTUAL_SWITCH_MISCONFIGURED`	The virtual switch is configured incorrectly.
	`SREC_INSTANCE_NOT_RUNNING`	The process of the virtual controller is no longer running.

Table 7- 125 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.
	`ERuntimeError-Code.WarningAlreadyExists`	Warning: The instance is started.
	`ERuntimeError-Code.WarningTrialModeActive`	No license available. You can use the instance without restrictions with the Trial License. Afterwards, the instance is shut down.
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.ErrorLoadingDll`	The Siemens.Simatic.Simulation. Runtime.Instance.exe cannot load the Siemens.Simatic.PlcSim.Vplc1500.dll.
	`ERuntimeError-Code.StoragePathAlreadyInUse`	The selected path for this instance is already being used by another instance.
	`ERuntimeError-Code.NoStoragePathSet`	The path could not be created. The length of the `DSTORAGE_PATH_MAX_LENGTH` characters may be exceeded.
	`ERuntimeError-Code.VirtualSwitchMisconfigured`	The virtual switch is configured incorrectly.
	`ERuntimeError-Code.InstanceNotRunning`	The process of the virtual controller is no longer running.

PowerOff()

Shuts down the Simulation Runtime and closes its process.

Table 7- 126 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 states 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- 127 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 states when this function is successful: `{ EOperatingState.Off }`	
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.

MemoryReset()

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

Table 7- 128 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 when 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- 129 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.

Run()

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

Table 7- 130 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 when 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- 131 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- 132 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 states 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- 133 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 states 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.

7.6.4 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]", where {a} and {b} correspond to the first index of the respective dimension

..

Entry_N: "ArrayName[x,y]", where {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- 134 UpdateTagList() - Native C++

<table>
<tr><td>Syntax</td><td colspan="2">

```
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
);
```

</td></tr>
<tr><td>Parameters</td><td colspan="2">

<code>ETagListDetails in_TagListDetails:</code>
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.
<code>bool in_IsHMIVisibleOnly:</code>
If <code>true</code>, only tags marked with "HMI Visible" are read. The default setting is <code>true</code>.
<code>WCHAR* in_DataBlockFilterList:</code>
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, <code>in_DataBlockFilterList</code> has to be unequal to NULL and <code>in_TagListDetails</code> has to contain "DB".

</td></tr>
<tr><td rowspan="8">Return values</td><td>Runtime error code</td><td>Condition</td></tr>
<tr><td><code>SREC_OK</code></td><td>The function is successful.</td></tr>
<tr><td><code>SREC_INTERFACE_REMOVED</code></td><td>The instance is not registered in Runtime Manager.</td></tr>
<tr><td><code>SREC_TIMEOUT</code></td><td>The function does not return on time.</td></tr>
<tr><td><code>SREC_INSTANCE_NOT_RUNNING</code></td><td>The process of the virtual controller is not running.</td></tr>
<tr><td><code>SREC_NOT_ENOUGH_MEMORY</code></td><td>More than 500000 entries are requested.</td></tr>
<tr><td><code>SREC_WARNING_ALREADY_EXISTS</code></td><td>The tag table is current.</td></tr>
<tr><td><code>SREC_WRONG_ARGUMENT</code></td><td>The syntax of <code>in_DataBlockFilterList</code> is invalid. The list has to be 3 characters long; the first and last character have to be a quotation mark.</td></tr>
</table>

Table 7- 135 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
	`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.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- 136 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- 137 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- 138 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- 139 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- 140 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- 141 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- 142 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.
	`ERuntimeErrorCode.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 158)). 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 vales 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- 143 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- 144 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 330).
Return values	`UINT32`: Size of the area in bytes. If the function was successful, the value is not equal to 0.

Table 7- 145 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 173) and not via the address areas.

Table 7- 146 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 330). • `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- 147 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- 148 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 330). • `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- 149 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- 150 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 330). • `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- 151 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 173) and not via the address areas.

Table 7- 152 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 330). • `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 preceding 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 335).
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- 153 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 191) and not via the address areas.

Table 7- 154 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 330). • `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- 155 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 191) and not via the address areas.

Table 7- 156 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 330). • `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- 157 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 191) and not via the address areas.

Table 7- 158 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 330). • `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- 159 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 191) and not via the address areas.

Table 7- 160 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 330). • `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- 161 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- 162 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_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 335).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 163 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.
	`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 336).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

ReadBool()

Reads the value of a PLC tag.

Table 7- 164 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_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 335).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 165 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.
	`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 335).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

ReadInt8()

Reads the value of a PLC tag.

Table 7- 166 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_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 335).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 167 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.
	`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 335).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

ReadInt16()

Reads the value of a PLC tag.

Table 7- 168 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_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 335).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 169 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.
	`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 335).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

ReadInt32()

Reads the value of a PLC tag.

Table 7- 170 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_DOES_NOT_EXIST`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 335).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 171 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.
	`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 335).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

ReadInt64()

Reads the value of a PLC tag.

Table 7- 172 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_DOES_NOT_EXIST`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMismatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 335).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 173 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.
	`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 335).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

ReadUInt8()

Reads the value of a PLC tag.

Table 7- 174 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_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 335).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 175 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.
	`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 335).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

ReadUInt16()

Reads the value of a PLC tag.

Table 7- 176 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_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 335).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 177 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.
	`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 335).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

ReadUInt32()

Reads the value of a PLC tag.

Table 7- 178 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_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 335).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 179 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.
	`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 335).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

ReadUInt64()

Reads the value of a PLC tag.

Table 7- 180 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_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 335).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 181 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.
	`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 335).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

ReadFloat()

Reads the value of a PLC tag.

Table 7- 182 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.
	`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 335).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 183 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.
	`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 335).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

ReadDouble()

Reads the value of a PLC tag.

Table 7- 184 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.
	`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 335).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 185 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.
	`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 335).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

ReadChar()

Reads the value of a PLC tag.

Table 7- 186 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.
	`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 335).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 187 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.
	`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 335).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

ReadWChar()

Reads the value of a PLC tag.

Table 7- 188 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.
	`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 335).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 189 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.
	`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 335).
	`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- 190 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 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 335).
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- 191 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.
	`ERuntimeErrorCode.DoesNotExist`	The entry does not exist in the stored tag table.
	`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 335).
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- 192 Write() - Native C++

Syntax	`ERuntimeErrorCode Write(`
	`  WCHAR* in_Tag,`
	`  SDataValue* in_Value`
	`);`	

Parameters	• `WCHAR* in_Tag:` The name of the PLC tag that is to be written. • `SDataValue* in_Value:` Contains the value and the expected type of the PLC tag. The `UNSPECIFIC` and `STRUCT` types are not supported. Structures and fields can be emulated through signal lists and then be read by using the `ReadSignals()` function and written by using the `WriteSignals()` function.	
Return values	Runtime error code	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 335).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.
	`SREC_WRONG_ARGUMENT`	The expected type is `UNSPECIFIC`.

Table 7- 193 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.
	`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 335).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.
	`ERuntimeErrorCode.WrongArgument`	The expected type is `UNSPECIFIC`.

WriteBool()

Writes the value of a PLC tag.

Table 7- 194 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_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 335).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 195 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.
	`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 337).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

WriteInt8()

Writes the value of a PLC tag.

Table 7- 196 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_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 335).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 197 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.
	`ERuntimeErrorCode.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMissmatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 335).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

WriteInt16()

Writes the value of a PLC tag.

Table 7- 198 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_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 335).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 199 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.
	`ERuntimeErrorCode.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMissmatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 335).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

WriteInt32()

Writes the value of a PLC tag.

Table 7- 200 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_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 335).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 201 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.
	`ERuntimeErrorCode.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMissmatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 335).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

WriteInt64()

Writes the value of a PLC tag.

Table 7- 202 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_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 335).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 203 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.
	`ERuntimeErrorCode.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMissmatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 335).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

WriteInt8()

Writes the value of a PLC tag.

Table 7- 204 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_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 335).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 205 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.
	`ERuntimeErrorCode.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMissmatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 335).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

WriteUInt16()

Reads the value of a PLC tag.

Table 7- 206 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_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 335).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 207 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.
	`ERuntimeErrorCode.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMissmatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 335).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

WriteUInt32()

Writes the value of a PLC tag.

Table 7- 208 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_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 335).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 209 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.
	`ERuntimeErrorCode.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMissmatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 335).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

WriteUInt64()

Writes the value of a PLC tag.

Table 7- 210 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_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 335).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 211 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.
	`ERuntimeErrorCode.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMissmatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 335).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

WriteFloat()

Writes the value of a PLC tag.

Table 7- 212 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_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 335).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 213 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.
	`ERuntimeErrorCode.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMissmatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 335).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

WriteDouble()

Writes the value of a PLC tag.

Table 7- 214 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_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 335).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 215 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.
	`ERuntimeErrorCode.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMissmatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 335).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

WriteChar()

Writes the value of a PLC tag.

Table 7- 216 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_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 335).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 217 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.
	`ERuntimeErrorCode.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMissmatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 335).
	`ERuntimeErrorCode.NotUpToData`	The stored tag list must be updated.

WriteWChar()

Writes the value of a PLC tag.

Table 7- 218 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_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 335).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.

Table 7- 219 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.
	`ERuntimeErrorCode.DoesNotExist`	The entry does not exist in the stored tag list.
	`ERuntimeErrorCode.NotSupported`	Access to entire structures or arrays is not supported.
	`ERuntimeErrorCode.TypeMissmatch`	The expected type does not match the stored type. See Compatible primitive data types (Page 335).
	`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- 220 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.	
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 335).
	`SREC_NOT_UP_TO_DATE`	The stored tag list must be updated.
	`SREC_WRONG_ARGUMENT`	The expected type is `UNSPECIFIC`.

Table 7- 221 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	
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.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 337).
	`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- 222 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- 223 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- 224 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.
	`ERuntimeErrorCode.WrongArgument`	The value is outside the limits.

GetScaleFactor()

Returns the scaling factor with which the virtual time advances.

Table 7- 225 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- 226 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- 227 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.
	`ERuntimeErrorCode.WrongArgument`	The value is outside the limits.

7.6.7 Cycle control

GetOperatingMode()

Returns the operating mode (Page 331) of the virtual controller.

Table 7- 228 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- 229 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- 230 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 247).

Table 7- 231 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- 232 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 247).

Table 7- 233 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- 234 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- 235 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- 236 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.
	`ERuntimeErrorCode.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- 237 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- 238 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- 239 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- 240 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
	`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.WrongArgument`	The value is less than 0.

Additional information

For further information, see sections Virtual time response (Page 73), Stop simulation (Page 76).

7.6.8 Acyclic services

7.6.8.1 Overview

The acyclic services of PLCSIM Advanced include:

- Interrupt and event information which the I/O modules send to the CPU.
- Read and write processes of parameter and status data from the user program of the PLC to the I/O modules.

API methods and associated events

Events which are triggered by the I/O modules and associated API methods:

Table 7- 241 Overview - API methods and associated events

OB	Name	API methods for triggering the OB (query)	API event after OB execution (alarm)
4x	Hardware Interrupt	ProcessEvent (Page 234)	OnProcessEventDone (Page 259)
55	Status	StatusEvent (Page 238)	OnStatusEventDone (Page 261)
56	Update	UpdateEvent (Page 240)	OnUpdateEventDone (Page 263)
57	Profile	ProfileEvent (Page 239)	OnProfileEventDone (Page 262)
82	Diagnostic error Interrupt	AlarmNotification (Page 232)	OnAlarmNotificationDone (Page 258)
83	Pull or Plug of module	PullOrPlugEvent (Page 236)	OnPullOrPlugEventDone (Page 260)
86	Rack or station failure	RackOrStationFaultEvent (Page 242)	OnRackOrStationFaultEvent-Done (Page 264)

Flowchart for the simulation of events which are triggered by the I/O modules.

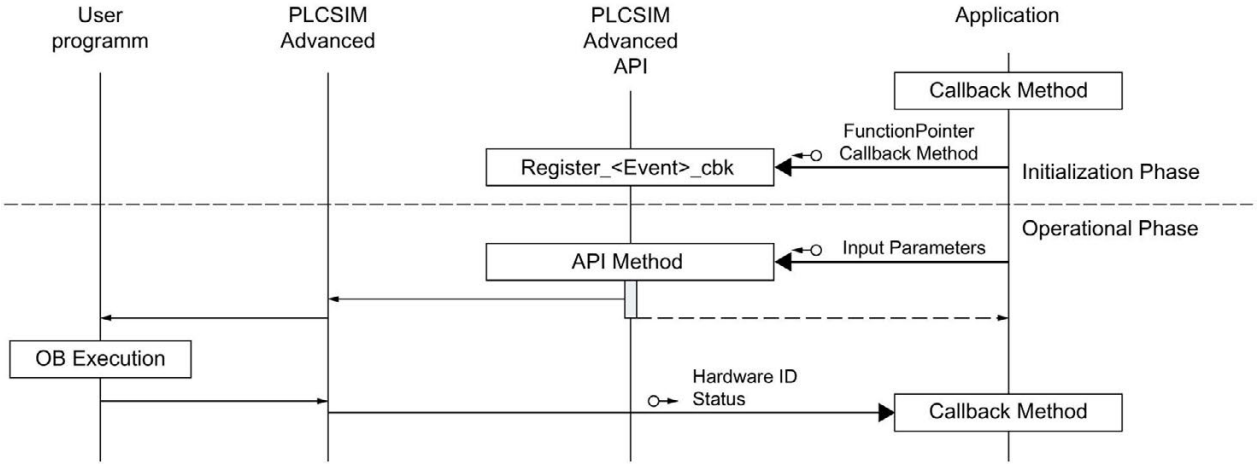


Figure 7-4 Flowchart for the simulation of events

Events

Events triggered by the user program (TIA Portal), which have logged on for the notification:

SFB	Name	API event for triggering the SFB	API method (alarm)
52	RDREC	OnDataRecordRead (Page 257)	ReadRecordDone (Page 230)
53	WRREC	OnDataRecordWrite (Page 257)	WriteRecordDone (Page 230)

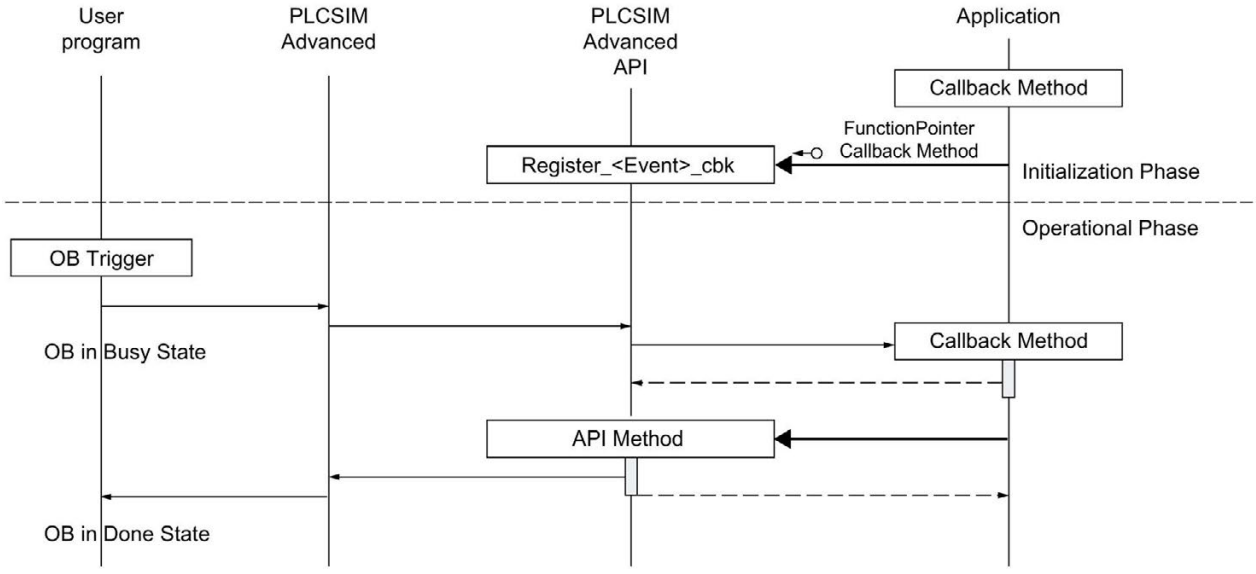


Figure 7-5 Flowchart

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- 242 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 321). • `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_TIMEOUT`	The function does not return on time.

Table 7- 243 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 321). • `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.
	`ERuntimeErrorCode.DoesNotExist`	No hardware identifier of the module.
	`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- 244 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 321). • `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- 245 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 321). • `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.
	`ERuntimeErrorCode.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- 246 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, DMODULE_STATE_ERROR, DMODULE_STATE_MAINT_DEMANDED, DMODULE_STATE_MAINT_REQUIRED • `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 324). • `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_DOES_NOT_EXIST`	No hardware identifier of the module.
	`SREC_INDEX_OUT_OF_RANGE`	The field size of `in_ArrayOfDiagnosisEvents` is less than the number of the currently available diagnostic events.
	`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- 247 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, ModuleState.Error, ModuleState.MaintenanceDemanded, ModuleState.MaintenanceRequired • `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. • `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.
	`ERuntimeErrorCode.DoesNotExist`	No hardware identifier of the module.
	`ERuntimeErrorCode.WrongArgument`	The value for the module status is invalid.
	`ERuntimeError-Code.WrongModuleType`	The channel number does not exist for the module.
	`ERuntimeError-Code.IndexOutOfRange`	The field size of `in_ArrayOfDiagnosisEvents` is less than the number of the currently available diagnostic events.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.

7.6.8.4 ProcessEvent

ProcessEvent()

Process events from central and distributed input modules can be simulated with this function.

Table 7- 248 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 EProcessEventType (Page 344). • `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_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- 249 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 344). • `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.
	`ERuntimeErrorCode.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- 250 PullOrPlugEvent() - Native C++

Syntax	`ERuntimeErrorCode PullOrPlugEvent(`
	`UINT16 in_HardwareIdentifier,`
	`EPullOrPlugEventType in_PullOrPlugEventType,`
	`UINT16* out_SequenceNumber`
	`);`	

Parameters	• `UINT16 in_HardwareIdentifier:` The hardware identifier of the module or submodule which generates the pull/plug event. The identifier must belong to a hardware component in the currently loaded project. • `EPullOrPlugEventType in_PullOrPlugEventType:` A value from the list of predefined types of pull/plug events, see EPullOrPlugEventType (Page 344). • `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- 251 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 344). • `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.WrongModuleType`	The wrong module type was selected. For example, if an onboard IO of a compact CPU is to be pulled.
	`ERuntimeErrorCode.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- 252 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_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- 253 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.
	`ERuntimeErrorCode.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- 254 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_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- 255 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.
	`ERuntimeErrorCode.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- 256 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_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- 257 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.
	`ERuntimeErrorCode.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- 258 GetConfiguredProcessEvents() - Native C++

Syntax	`ERuntimeErrorCode GetConfiguredProcessEvents( UINT16* out_EventsCount, );`	
Parameters	• `SConfiguredProcessEvents* inout_ProcessEvents:` Pointer or reference to a user-defined memory which contains the field with the downloaded configured process events. The structure SConfiguredProcessEvents (Page 323) 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- 259 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
	`ERuntimeError-Code.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- 260 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. • `ERackOrStationFaultType in_EventType:` A value from the list of predefined types of events, see ERackOrStation-FaultType (Page 346).	
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- 261 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. • `ERackOrStationFaultType in_EventType:` A value from the list of predefined types of events, see ERackOrStation-FaultType (Page 346).	
Exceptions	`Siemens.Simatic.Simulation.Runtime.SimulationRuntimeException`	
	Runtime error code	Condition
	`ERuntimeError-Code.InterfaceRemoved`	The instance is not registered in Runtime Manager.
	`ERuntimeErrorCode.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 identifier already reports the status Fault/Return.
	`ERuntimeErrorCode.Timeout`	The function does not return on time.

7.6.9 Events

Events for IInstances

The following events are differentiated in the API:

Table 7- 262 Events for the IInstances interface

Event	Cause
OnOperatingStateChanged (Page 243)	This event is always triggered when the operating state of the virtual controller has changed.
OnSyncPointReached (Page 247)	This event is triggered when the virtual controller reaches 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 223).
OnConfigurationChanging (Page 249)	This event is triggered when changing of the configuration of the virtual controller starts. • 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 252)	This event is triggered when 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
OnLedChanged (Page 254)	This event is triggered when the LED display of the virtual controller has changed.

7.6.9.1 OnOperatingStateChanged events

OnOperatingStateChanged

Registers or unregisters an event handler method.

Table 7- 263 OnOperatingStateChanged - .NET (C#)

Syntax	`event Delegate_II_EREC_DT_EOS_EOS OnOperatingStateChanged;`
Parameters	None. See Delegate_II_EREC_DT_EOS_EOS (Page 294).
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- 264 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 282).
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- 265 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- 266 UnregisterOnOperatingStateChangedCallback() - Native C++

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

UnregisterOnOperatingStateChangedEvent()

Unregisters the event object.

Table 7- 267 UnregisterOnOperatingStateChangedEvent() - Native C++

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

Table 7- 268 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- 269 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- 270 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.

7.6.9.2 OnSyncPointReached events

OnSyncPointReached

Registers or unregisters an event handler method.

Table 7- 271 OnSyncPointReached - .NET (C#)

Syntax	`event Delegate_II_EREC_DT_UINT32_INT64_INT64_UINT32 OnSyncPoin-tReached;`
Parameters	None. See Delegate_II_EREC_DT_UINT32_INT64_INT64_UINT32 (Page 296).
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- 272 RegisterOnSyncPointReachedCallback() - Native C++

Syntax	`void RegisterOnSyncPointReachedCallback( EventCallback_II_SREC_ST_UINT32_INT64_INT64_UINT32 in_CallbackFunction );`
Parameters	• `EventCallback_II_SREC_ST in_CallbackFunction:` A callback function that subscribes to an event. See EventCallback_II_SREC_ST_UINT32_INT64_INT64_UINT32 (Page 281).
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- 273 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- 274 UnregisterOnSyncPointReachedCallback() - Native C++

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

UnregisterOnSyncPointReachedEvent()

Unregisters the event object.

Table 7- 275 UnregisterOnSyncPointReachedEvent() - Native C++

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

Table 7- 276 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- 277 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 319).

Table 7- 278 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 319).

7.6.9.3 OnConfigurationChanging events

OnConfigurationChanging

Registers or unregisters an event handler method.

Table 7- 279 OnConfigurationChanging - .NET (C#)

Syntax	`event Delegate_II_EREC_DT OnConfigurationChanging;`
Parameters	None. See Delegate_II_EREC_DT (Page 294).
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- 280 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 281).
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- 281 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- 282 UnregisterOnConfigurationChangingCallback() - Native C++

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

UnregisterOnConfigurationChangingEvent()

Unregisters the event object.

Table 7- 283 UnregisterOnConfigurationChangingEvent() - Native C++

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

Table 7- 284 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- 285 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- 286 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.

7.6.9.4 OnConfigurationChanged events

OnConfigurationChanged

Registers or unregisters an event handler method.

Table 7- 287 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 298).
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- 288 RegisterOnConfigurationChangedCallback() - Native C++

Syntax	`void RegisterOnConfigurationChangedCallback( EventCallback_II_SREC_ST_SRICC_UINT32_UINT32_UINT32_UINT32 in_CallbackFunction );`
Parameters	• `EventCallback_II_SREC_ST_SRICC_UINT32_UINT32_UINT32_UINT32 in_CallbackFunction:` A callback function that subscribes to an event. See EventCallback_II_SREC_ST_SRICC_UINT32_UINT32_UINT32_UINT32 (Page 286) .
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- 289 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- 290 UnregisterOnConfigurationChangedCallback() - Native C++

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

UnregisterOnConfigurationChangedEvent()

Unregisters the event object.

Table 7- 291 UnregisterOnConfigurationChangedEvent() - Native C++

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

Table 7- 292 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- 293 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- 294 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.

7.6.9.5 OnLedChanged events

OnLedChanged

Registers or unregisters an event handler method.

Table 7- 295 OnLedChanged - .NET (C#)

Syntax	`event Delegate_II_EREC_DT_ELT_ELM OnLedChanged;`
Parameters	None. See Delegate_II_EREC_DT_ELT_ELM (Page 295).
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- 296 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 283).
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- 297 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- 298 UnregisterOnLedChangedCallback() - Native C++

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

UnregisterOnLedChangedEvent()

Unregisters the event object.

Table 7- 299 UnregisterOnLedChangedEvent() - Native C++

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

Table 7- 300 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- 301 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- 302 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.

7.6.10 Events for acyclic services

7.6.10.1 OnDataRecordRead / OnDataRecordWrite events

OnDataRecordRead

Registers or unregisters an event handler method.

Table 7- 303 OnDataRecordRead - .NET (C#)

Syntax	`event Delegate_II_EREC_DT_SDRI OnDataRecordRead;`
Parameter	None. See Delegate_II_EREC_DT_SDRI (Page 300).
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- 304 OnDataRecordWrite - .NET (C#)

Syntax	`event Delegate_II_EREC_DT_SDR OnDataRecordWrite;`
Parameter	None. See Delegate_II_EREC_DT_SDR (Page 299).
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- 305 RegisterOnDataRecordReadCallback() - Native C++

Syntax	`void RegisterOnDataRecordReadCallback (`
	`  Event Callback_II_SREC_ST_SDRI in_CallbackFunction`
	`);`

Parameters	• `EventCallback_II_SREC_ST_SDRI in_CallbackFunction:` A callback function that subscribes to the event. See EventCallback_II_SREC_ST_SDRI (Page 288).
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- 306 UnregisterOnDataRecordReadCallback() - Native C++

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

7.6.10.2 OnAlarmNotification events

OnAlarmNotificationDone()

Registers or unregisters an event handler method.

Table 7- 307 OnAlarmNotificationDone() - .NET (C#)

Syntax	`event Delegate_SREC_ST_UINT32_UINT32 OnAlarmNotificationDone;`
Parameters	None. See Delegate_SREC_ST_UINT32_UINT32 (Page 304).
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- 308 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 292).
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- 309 UnregisterOnAlarmNotificationDoneCallback() - Native C++

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

7.6.10.3 OnProcessEvent events

OnProcessEventDone()

Registers or unregisters an event handler method.

Table 7- 310 OnProcessEventDone() - .NET (C#)

Syntax	`event Delegate_SREC_ST_UINT32_UINT32_EPET_UINT32 OnProcessEvent-Done;`
Parameters	None. See Delegate_SREC_ST_UINT32_UINT32_EPET_UINT32. (Page 302)
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- 311 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 290)
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- 312 UnregisterOnProcessEventDoneCallback() - Native C++

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

7.6.10.4 OnPullOrPlugEvent events

OnPullOrPlugEventDone()

Registers or unregisters an event handler method.

Table 7- 313 OnPullOrPlugEventDone() - .NET (C#)

Syntax	`event Delegate_SREC_ST_UINT32_EPPET_UINT32 OnPullOrPlugEventDone;`
Parameters	None. See Delegate_SREC_ST_UINT32_EPPET_UINT32 (Page 301).
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- 314 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 289).
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- 315 UnregisterOnPullOrPlugEventDoneCallback() - Native C++

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

7.6.10.5 OnStatusEvent events

OnStatusEventDone()

Registers or unregisters an event handler method.

Table 7- 316 OnStatusEventDone() - .NET (C#)

Syntax	`event Delegate_SREC_ST_UINT32 OnStatusEventDone;`
Parameters	None. See Delegate_SREC_ST_UINT32 (Page 303).
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- 317 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 291).
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- 318 UnregisterOnStatusEventDoneCallback() - Native C++

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

7.6.10.6 OnProfileEvent events

OnProfileEventDone()

Registers or unregisters an event handler method.

Table 7- 319 OnProfileEventDone() - .NET (C#)

Syntax	`event Delegate_SREC_ST_UINT32 OnProfileEventDone;`
Parameters	None. See Delegate_SREC_ST_UINT32 (Page 303).
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- 320 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 291).
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- 321 UnregisterOnProfileEventDoneCallback() - Native C++

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

7.6.10.7 OnUpdateEvent events

OnUpdateEventDone()

Registers or unregisters an event handler method.

Table 7- 322 OnUpdateEventDone() - .NET (C#)

Syntax	`event Delegate_SREC_ST_UINT32 OnUpdateEventDone;`
Parameters	None. See Delegate_SREC_ST_UINT32 (Page 303).
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- 323 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 291).
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- 324 UnregisterOnUpdateEventDoneCallback() - Native C++

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

7.6.10.8 RackOrStationFault events

OnRackOrStationFaultEvent

Registers or unregisters an event handler method.

Table 7- 325 OnRackOrStationFaultEvent - .NET (C#)

Syntax	`event Delegate_SREC_ST_UINT32_ERSFET OnRackOrStationFault;`
Parameter	None. See Delegate_SREC_ST_UINT32_ERSFET (Page 305).
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- 326 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 293)
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- 327 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- 328 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- 329 GetVersion() - Native C++

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

Table 7- 330 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- 331 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- 332 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- 333 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- 334 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.

Table 7- 335 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_OT_OF_RANGE`	The array is too small to accommodate the computer name.

Table 7- 336 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- 337 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- 338 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.

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- 339 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- 340 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 `GetRegis-teredInstanceCount()`. • `SInstanceInfo* out_InstanceInfo:` The information with name and ID of the instance. See SInstanceInfo (Page 314).	
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- 341 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- 342 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 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 306). • `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 101).

Table 7- 343 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 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 306).	
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
	`ERuntimeError-Code.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- 344 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 306). • `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 101).

Table 7- 345 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 306).	
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
	`ERuntimeError-Code.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- 346 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 101)

Table 7- 347 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
	`ERuntimeError-` `Code.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

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- 348 OnConnectionLost - .NET (C#)

Syntax	`event Delegate_IRRTM OnConnectionLost;`
Parameters	None. See Delegate_IRRTM (Page 297)
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- 349 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 285).
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- 350 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- 351 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- 352 UnregisterOnConnectionLostCallback() - Native C++

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

UnregisterOnConnectionLostEvent()

Unregisters the event object.

Table 7- 353 UnregisterOnConnectionLostEvent() - Native C++

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

Table 7- 354 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- 355 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- 356 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 V2.0, 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 STEP 7 Professional (https://support.industry.siemens.com/cs/ww/en/view/109747136) 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- 357 ApiEntry_Initialize - Native C++

Syntax	`typedef ERuntimeErrorCode(*ApiEntry_Initialize)( ISimulationRuntimeManager** out_RuntimeManagerInterface );`	
Parameters	• `ISimulationRuntimeManager** out_SimulationRuntimeManagerInterface:` Pointer to a Runtime Manager interface pointer. The pointer must be initialized with NULL. The interface is created within the function. • `UINT32 in_InterfaceVersion:` Version of the API interface to be downloaded: `API_DLL_INTERFACE_VERSION`.	
Return values	Runtime error code	Condition
	`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.

7.8.1.2 ApiEntry_DestroyInterface

Description

Type of the entry point for DestroyInterface (Page 101).

Table 7- 358 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- 359 EventCallback_VOID - Native C++

Syntax	`typedef void (*EventCallback_VOID)();`
Parameters	None
Return values	None

7.8.2.2 EventCallback_II_SREC_ST

Description

Table 7- 360 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

7.8.2.3 EventCallback_II_SREC_ST_SROS_SROS

Description

Table 7- 361 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.

7.8.2.4 EventCallback_II_SREC_ST_SRLT_SRLM

Description

Table 7- 362 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

7.8.2.5 EventCallback_II_SREC_ST_UINT32_INT64_INT64_UINT32

Description

Table 7- 363 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 80): The runtime since the last call of the `StartProcessing()` function. • `INT64 in_TimeSinceAnySyncPoint_ns:` The virtual time (in nanoseconds) since the last synchronization point of any process image partition ID was reached. For the time-controlled operating modes: The runtime since the last call of the `StartProcessing()` function. • `UINT32 in_SyncPointCount:` The number of synchronization points since the last event. If the events are triggered faster than they are received, multiple events are combined into one event. In this case, this value contains the number of cycles since the last event was received.
Return values	None

7.8.2.6 EventCallback_IRRTM

Description

Table 7- 364 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

7.8.2.7 EventCallback_SRCC_UINT32_UINT32_INT32

Description

Table 7- 365 EventCallback_SRCC_UINT32_UINT32_INT32 - Native C++

Syntax	`ERuntimeConfigChanged in_RuntimeConfigChanged, UINT32 in_Param1, UINT32 in_Param2, INT32 in_Param3 );`			
Parameters	`ERuntimeConfigChanged in_RuntimeConfigChanged`	`UInt32 in_Param1`	`UInt32 in_Param2`	`Int32 in_Param3`
	`SRCC_INSTANCE_REGISTERED`	-	-	ID of the registered instance
	`SRCC_INSTANCE_UNREGISTERED`	-	-	ID of the unregistered instance
	`SRCC_CONNECTION_OPENED`	IP of the remote Runtime Manager	Port of the remote Runtime Manager	-
	`SRCC_CONNECTION_CLOSED`	IP of the remote Runtime Manager	Port of the remote Runtime Manager	-
	`SRCC_PORT_OPENED`	The open port	-	-
	`SRCC_PORT_CLOSED`	-	-	-
Return values	None			

7.8.2.8 EventCallback_II_SREC_ST_SRICC_UINT32_UINT32_UINT32_UINT32

Description

Table 7- 366 EventCallback_II_SREC_ST_SRICC_UINT32_UINT32_UINT32_UINT32 - Native C++

<table>
<tr><td>Syntax</td><td colspan="5">

```
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
);
```

</td></tr>
<tr><td>Parameters</td><td colspan="5">

- `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.

</td></tr>
<tr><td></td><td>EIn-
stanceCon-
figChanged
in_Instance
Con-
figChanged</td><td>UINT32
in_Param1</td><td>UINT32
in_Param2</td><td>UINT32
in_Param3</td><td>UINT32
in_Param4</td></tr>
<tr><td></td><td>SRICC_HARDW
ARE_SOFTWAR
E_CHANGED</td><td>-</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td></td><td>SRICC_IP_CH
ANGED</td><td>The ID of the interface</td><td>The new IP</td><td>The new sub-net mask</td><td>The new standard gateway</td></tr>
<tr><td>Return values</td><td colspan="5">None</td></tr>
</table>

7.8.2.9 EventCallback_II_SREC_ST_SDRI

Description

Table 7- 367 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

7.8.2.10 EventCallback_II_SREC_ST_SDR

Description

Table 7- 368 EventCallback_II_SREC_ST_SDR - .NET (C#)

Syntax	`typedef void (*EventCallback_II_SREC_ST_SDR)(`
	`IInstance* in_Sender,`
	`ERuntimeErrorCode in_ErrorCode,`
	`SYSTEMTIME in_SystemTime,`
	`SDataRecord in_DataRecord`
	`);`

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. • `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

7.8.2.11 EventCallback_II_SREC_ST_UINT32_EPPET_UINT32

Description

Table 7- 369 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 344). • `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

7.8.2.12 EventCallback_II_SREC_ST_UINT32_UINT32_EPET_UINT32

Description

Table 7- 370 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 344). • `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

Description

Table 7- 371 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

7.8.2.14 EventCallback_II_SREC_ST_UINT32_UINT32

Description

Table 7- 372 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

7.8.2.15 EventCallback_II_SREC_ST_UINT32_ERSFET

Description

Table 7- 373 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 346).
Return values	None

7.8.3 Delegate definitions (managed code)

7.8.3.1 Delegate_Void

Description

Table 7- 374 Delegate_Void - .NET (C#)

Syntax	`delegate void Delegate_Void();`
Parameters	None
Return values	None

7.8.3.2 Delegate_II_EREC_DT

Description

Table 7- 375 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

7.8.3.3 Delegate_II_EREC_DT_EOS_EOS

Description

Table 7- 376 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.

7.8.3.4 Delegate_II_EREC_DT_ELT_ELM

Description

Table 7- 377 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

7.8.3.5 Delegate_II_EREC_DT_UINT32_INT64_INT64_UINT32

Description

Table 7- 378 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 80). • `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 80). • `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

7.8.3.6 Delegate_IRRTM

Description

Table 7- 379 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

7.8.3.7 Delegate_SRCC_UINT32_UINT32_INT32

Description

Table 7- 380 Delegate_SRCC_UINT32_UINT32_INT32 - .NET (C#)

Syntax	`delegate void Delegate_SRCC_UINT32_UINT32_INT32(`
	`  ERuntimeConfigChanged in_RuntimeConfigChanged,`
	`  UInt32 in_Param1,`
	`  UInt32 in_Param2,`
	`  Int32 in_Param3`
	`);`			

Parameters	`ERuntimeConfigChanged in_RuntimeConfigChanged`	`UInt32 in_Param1`	`UInt32 in_Param2`	`Int32 in_Param3`
	`InstanceRegistered`	-	-	ID of the registered instance
	`InstanceUnregistered`	-	-	ID of the unregistered instance
	`ConnectionOpened`	IP of the Remote Runtime Manager	Port of the remote Runtime Manager	-
	`Connection-Closed`	IP of the Remote Runtime Manager	Port of the remote Runtime Manager	-
	`PortOpened`	The open port	-	-
	`PortClosed`	-	-	-
Return values	None			

7.8.3.8 Delegate_II_EREC_DT_SRICC_UINT32_UINT32_UINT32_UINT32

Description

Table 7- 381 Delegate_II_EREC_DT_SRICC_UINT32_UINT32_UINT32_UINT32 - .NET (C#)

<table>
<tr><td>Syntax</td><td colspan="5">

```
delegate void Dele-
gate_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
);
```

</td></tr>
<tr><td>Parameters</td><td colspan="5">

- `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.

</td></tr>
<tr><td></td><td>EIn-
stanceCon-
figChanged
in_Instance
Con-
figChanged</td><td>UInt32
in_Param1</td><td>UInt32
in_Param2</td><td>UInt32
in_Param3</td><td>UInt32
in_Param4</td></tr>
<tr><td></td><td>Hard-
wareSoft-
wareChanged</td><td>-</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td></td><td>IPChanged</td><td>The ID of the interface</td><td>The new IP</td><td>The new sub-net mask</td><td>The new standard gateway</td></tr>
<tr><td>Return values</td><td colspan="5">None</td></tr>
</table>

7.8.3.9 Delegate_II_EREC_DT_SDRI

Description

Table 7- 382 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

7.8.3.10 Delegate_II_EREC_DT_SDR

Description

Table 7- 383 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

7.8.3.11 Delegate_SREC_ST_UINT32_EPPET_UINT32

Description

Table 7- 384 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 344). • `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

7.8.3.12 Delegate_SREC_ST_UINT32_UINT32_EPET_UINT32

Description

Table 7- 385 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 344). • `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

7.8.3.13 Delegate_SREC_ST_UINT32

Description

Table 7- 386 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

7.8.3.14 Delegate_SREC_ST_UINT32_UINT32

Description

Table 7- 387 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

7.8.3.15 Delegate_SREC_ST_UINT32_ERSFET

Description

Table 7- 388 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 346).
Return values	None

7.8.4 Definitions and constants

The following identifiers are used in the API:

Table 7- 389 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- 390 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- 391 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- 392 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

7.8.6.1 SDataValue

Description

The structure contains the value and type of a PLC tag.

Table 7- 393 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- 394 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- 395 SDVBNI - Native C++

Syntax	`struct SDVBNI`

Table 7- 396 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- 397 SDataValueByAddress - Native C++

Syntax	`struct SDataValueByAddress`
	`{`
	`UINT32 Offset;`
	`UINT8 Bit;`
	`SDataValue DataValue;`
	`ERuntimeErrorCode ErrorCode;`
	`};`

Table 7- 398 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- 399 SDataValueByAddressWithCheck - Native C++

Syntax	`struct SDataValueByAddressWithCheck`
	`{`
	` UINT32 Offset;`
	` UINT8 Bit;`
	` SDataValue DataValue;`
	` ERuntimeErrorCode ErrorCode;`
	` bool ValueHasChanged;`
	`};`

Table 7- 400 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- 401 SDataValueByName - Native C++

Syntax	`struct SDataValueByName`
	`{`
	` WCHAR Name[DTAG_NAME_MAX_LENGTH];`
	` SDataValue DataValue;`
	` ERuntimeErrorCode ErrorCode;`
	` SDVBNI Internal;`
	`};`

Table 7- 402 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- 403 SDataValueByNameWithCheck - Native C++

Syntax	`struct SDataValueByNameWithCheck`
	`{`
	` WCHAR Name[DTAG_NAME_MAX_LENGTH];`
	` SDataValue DataValue;`
	` ERuntimeErrorCode ErrorCode;`
	` SDVBNI Internal;`
	` bool ValueHasChanged;`
	`};`

Table 7- 404 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- 405 SConnectionInfo - Native C++

Syntax	`struct SConnectionInfo`
	`{`
	` UIP IP;`
	` UINT16 Port;`
	`};`

Table 7- 406 SConnectionInfo - .NET (C#)

Syntax	`struct SConnectionInfo`
	`{`
	` SIP IP;`
	` UInt16 Port;`
	`}`

7.8.6.8 SInstanceInfo

Description

This structure contains an IPv4 address.

Table 7- 407 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- 408 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- 409 SDimension - Native C++

Syntax	`struct SDimension`
	`{`
	`INT32 StartIndex;`
	`UINT32 Count;`
	`};`

Table 7- 410 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- 411 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 it is not a field for the tag. • `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- 412 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- 413 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- 414 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- 415 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- 416 SOnSyncPointReachedResult - Native C++

Syntax	`struct SOnSyncPointReachedResult`
	`{`
	`ERuntimeErrorCode ErrorCode;`
	`SYSTEMTIME SystemTime;`
	`UINT32 PipId;`
	`INT64 TimeSinceSameSyncPoint_ns;`
	`INT64 TimeSinceAnySyncPoint_ns;`
	`UINT32 SyncPointCount;`
	`};`

Member	• `ERuntimeErrorCode ErrorCode:` A possible error code (Page 327). If no event was triggered during the defined time interval: `SREC_TIMEOUT`. • `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 80): 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 80): 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- 417 SOnSyncPointReachedResult - .NET (C#)

Syntax	`struct SOnSyncPointReachedResult`
	`{`
	` ERuntimeErrorCode ErrorCode;`
	` DateTime SystemTime;`
	` UInt32 PipId;`
	` Int64 TimeSinceSameSyncPoint_ns;`
	` Int64 TimeSinceAnySyncPoint_ns;`
	` UInt32 SyncPointCount;`
	`}`

Member	• `ERuntimeErrorCode ErrorCode:` A possible error code (Page 327). If no event was triggered during the defined time interval: `ERuntimeErrorCode.Timeout`. • `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- 418 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- 419 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- 420 SDataRecord - Native C++

Syntax	`struct SDataRecord`
	`{`
	`SDataRecordInfo Info;`
	`BYTE Data[DDATARECORD_MAX_SIZE];`
	`};`

Member	• `SDataRecordInfo Info:` The data record information, see SDataRecordInfo (Page 321) • `BYTE Data[DDATARECORD_MAX_SIZE]:` The array length of the user data `DDATARECORD_MAX_SIZE`

Table 7- 421 SDataRecord - .NET (C#)

Syntax	`struct SDataRecord`
	`{`
	`UInt32 HardwareId;`
	`byte[] Data`
	`}`

Member	• `SDataRecordInfo Info:` The data record information, see SDataRecordInfo (Page 321) • `byte[] Data:` The array length

7.8.6.16 SConfiguredProcessEvents

Description

This structure contains information about the configured process events.

Table 7- 422 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- 423 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- 424 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 346). • `EDiagProperty Direction:` The value for the incoming/outgoing information, see EDiagProperty (Page 345). • `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- 425 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 346). • `EDiagProperty Direction:` The value for the incoming/outgoing information, see EDiagProperty (Page 345). • `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- 426 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- 427 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

7.8.7 Enumerations

The following enumerations are available:

- ERuntimeErrorCode (Page 327)
- EArea (Page 330)
- EOperatingState (Page 330)
- EOperatingMode (Page 331)
- ECPUType (Page 332)
- ECommunicationInterface (Page 334)
- ELEDType (Page 334)
- ELEDMode (Page 335)
- EPrimitiveDataType (Page 335)
- EDataType (Page 338)
- ETagListDetails (Page 342)
- ERuntimeConfigChanged (Page 343)
- EInstanceConfigChanged (Page 343)
- EPullOrPlugEventType (Page 344)
- EProcessEventType (Page 344)
- EDirection (Page 345)
- EDiagProperty (Page 345)
- EDiagSeverity (Page 346)
- ERackOrStationFaultType (Page 346)

See also

Global functions (Native C++) (Page 106)

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`. Errors are returned with negative values, and alarms with positive values.

Table 7- 428 ERuntimeErrorCode - Native C++

Syntax	

```
enum ERuntimeErrorCode
{
 SREC_ISyntaxenum ERuntimeErrorCode{
 SREC_OK = 0,
 SREC_INVALID_ERROR_CODE = -1,
 SREC_NOT_IMPLEMENTED = -2,
 SREC_INDEX_OUT_OF_RANGE = -3,
 SREC_DOES_NOT_EXIST = -4,
 SREC_ALREADY_EXISTS = -5,
 SREC_UNKNOWN_MESSAGE_TYPE = -6,
 SREC_INVALID_MESSAGE_ID = -7,
 SREC_WRONG_ARGUMENT = -8,
 SREC_WRONG_PIPE = -9,
 SREC_CONNECTION_ERROR = -10,
 SREC_TIMEOUT = -11,
 SREC_MESSAGE_CORRUPT = -12,
 SREC_WRONG_VERSION = -13,
 SREC_INSTANCE_NOT_RUNNING = -14,
 SREC_INTERFACE_REMOVED = -15,
 SREC_SHARED_MEMORY_NOT_INITIALIZED = -16,
 SREC_API_NOT_INITIALIZED = -17,
 SREC_WARNING_ALREADY_EXISTS = 18,
 SREC_NOT_SUPPORTED = -19,
 SREC_WARNING_INVALID_CALL = 20,
 SREC_ERROR_LOADING_DLL = -21,
 SREC_SIGNAL_NAME_DOES_NOT_EXIST = -22,
 SREC_SIGNAL_TYPE_MISMATCH = -23,
 SREC_SIGNAL_CONFIGURATION_ERROR = -24,
 SREC_NO_SIGNAL_CONFIGURATION_LOADED = -25,
 SREC_CONFIGURED_CONNECTION_NOT_FOUND = -26,
 SREC_CONFIGURED_DEVICE_NOT_FOUND = -27,
```

	SREC_INVALID_CONFIGURATION = -28, SREC_TYPE_MISMATCH = -29, SREC_LICENSE_NOT_FOUND = -30, SREC_NO_LICENSE_AVAILABLE = -31, SREC_WRONG_COMMUNICATION_INTERFACE = -32, SREC_LIMIT_REACHED = -33, SREC_NO_STORAGE_PATH_SET = -34, SREC_STORAGE_PATH_ALREADY_IN_USE = -35, SREC_MESSAGE_INCOMPLETE = -36, SREC_ARCHIVE_STORAGE_NOT_CREATED = -37, SREC_RETRIEVE_STORAGE_FAILURE = -38 SREC_INVALID_OPERATING_STATE = -39, SREC_INVALID_ARCHIVE_PATH = -40, SREC_DELETE_EXISTING_STORAGE_FAILED = -41, SREC_CREATE_DIRECTORIES_FAILED = -42, SREC_NOT_ENOUGH_MEMORY = -43, SREC_WARNING_TRIAL_MODE_ACTIVE = 44, SREC_NOT_RUNNING = -45, SREC_NOT_EMPTY = -46, SREC_NOT_UP_TO_DATE = -47, SREC_COMMUNICATION_INTERFACE_NOT_AVAILABLE = -48, SREC_WARNING_NOT_COMPLETE = 49, SREC_VIRTUAL_SWITCH_MISCONFIGURED = -50, SREC_RUNTIME_NOT_AVAILABLE = -51, SREC_IS_EMPTY = -52, SREC_WRONG_MODULE_STATE = -53, SREC_WRONG_MODULE_TYPE = -54, SREC_NOT_SUPPORTED_BY_MODULE = -55, SREC_INTERNAL_ERROR = -56, SREC_STORAGE_TRANSFER_ERROR = -57, SREC_ANOTHER_VARIANT_OF_PLCSIM_RUNNING = -58, SREC_ACCESS_DENIED = -59, SREC_NOT_ALLOWED_DURING_DOWNLOAD = -60 };

Table 7- 429 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,
	MesssageIncomplete = -36,
	ArchiveStorageNotCreated = -37,
	RetrieveStorageFailure = -38,
	InvalidOperatingState = -39,
	InvalidArchivePath = -40,
	DeleteExistingStorageFailed = -41,
	CreateDirectoriesFailed = -42,
	NotEnoughMemory = -43,
	WarningTrialModeActive = 44,
	NotRunning = -45,
	NotEmpty = -46,
	NotUpToData = -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
	}

7.8.7.2 EArea

Description

This enumeration contains all PLC areas that contain the available PLC tags.

Table 7- 430 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- 431 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- 432 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- 433 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- 434 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
	};

Table 7- 435 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
	}

7.8.7.5 ECPUType

Description

This enumeration contains all CPU types that can be loaded in a virtual controller.

Table 7- 436 ECPUType - Native C++

Syntax	enum ECPUType
	{
	SRCT_1500_Unspecified = 0x000005DC,
	SRCT_1511 = 0x000005E7,
	SRCT_1511v2 = 0x010005E7,
	SRCT_1511v3 = 0x020005E7,
	SRCT_1513 = 0x000005E9,
	SRCT_1513v2 = 0x010005E9,
	SRCT_1513v3 = 0x020005E9
	SRCT_1515 = 0x000005EB,
	SRCT_1515v2 = 0x010005EB,
	SRCT_1516 = 0x000005EC,
	SRCT_1516v2 = 0x010005EC,
	SRCT_1517 = 0x000005ED,
	SRCT_1518 = 0x000005EE,
	SRCT_1511C = 0x000405E7,
	SRCT_1511Cv2 = 0x010405E7,
	SRCT_1512C = 0x000405E8,
	SRCT_1512Cv2 = 0x010405E8,
	SRCT_1511F = 0x000105E7,
	SRCT_1511Fv2 = 0x010105E7,
	SRCT_1511Fv3 = 0x020105E7,
	SRCT_1513F = 0x000105E9,
	SRCT_1513Fv2 = 0x010105E9,
	SRCT_1513Fv3 = 0x020105E9,
	SRCT_1515F = 0x000105EB,
	SRCT_1515Fv2 = 0x010105EB,
	SRCT_1516F = 0x000105EC,
	SRCT_1516Fv2 = 0x010105EC,
	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_ET200SP_Unspecified = 0x000205DC,
	SRCT_1510SP = 0x000205E6,
	SRCT_1510SPv2 = 0x010205E6,
	SRCT_1512SP = 0x000205E8,
	SRCT_1512SPv2 = 0x010205E8,
	SRCT_1510SPF = 0x000305E6,
	SRCT_1510SPFv2 = 0x010305E6,
	SRCT_1512SPF = 0x000305E8,
	SRCT_1512SPFv2 = 0x010305E8
	}

Table 7- 437 ECPUType - .NET (C#)

Syntax	enum ECPUType
	{
	CPU1500_Unspecified = 0x000005DC,
	CPU1511 = 0x000005E7,
	CPU1511v2 = 0x010005E7,
	CPU1513 = 0x000005E9,
	CPU1513v2 = 0x010005E9,
	CPU1511v3 = 0x020005E7,
	CPU1515 = 0x000005EB,
	CPU1515v2 = 0x010005EB,
	CPU1513v3 = 0x020005E9,
	CPU1516 = 0x000005EC,
	CPU1516v2 = 0x010005EC,
	CPU1517 = 0x000005ED,
	CPU1518 = 0x000005EE,
	CPU1511C = 0x000405E7,
	CPU1511Cv2 = 0x010405E7,
	CPU1512C = 0x000405E8,
	CPU1512Cv2 = 0x010405E8,
	CPU1511F = 0x000105E7,
	CPU1511Fv2 = 0x010105E7,
	CPU1511Fv3 = 0x020105E7,
	CPU1513F = 0x000105E9,
	CPU1513Fv2 = 0x010105E9,
	CPU1513Fv3 = 0x020105E9,
	CPU1515F = 0x000105EB,
	CPU1515Fv2 = 0x010105EB,
	CPU1516F = 0x000105EC,
	CPU1516Fv2 = 0x010105EC,
	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,
	CPUET200SP_Unspecified = 0x000205DC,
	CPU1510SP = 0x000205E6,
	CPU1510SPv2 = 0x010205E6,
	CPU1512SP = 0x000205E8,
	CPU1512SPv2 = 0x010205E8,
	CPU1510SPF = 0x000305E6,
	CPU1510SPFv2 = 0x010305E6,
	CPU1512SPF = 0x000305E8,
	CPU1512SPFv2 = 0x010305E8
	}

7.8.7.6 ECommunicationInterface

Description

This enumeration contains the available communication interfaces of a virtual controller.

Table 7- 438 ECommunicationInterface - Native C++

Syntax	`enum ECommunicationInterface`
	`{`
	` SRCI_NONE = 0,`
	` SRCI_SOFTBUS = 1,`
	` SRCI_TCPIP = 2,`
	` SRCI_ENUMERATION_SIZE = 3`
	`};`

Table 7- 439 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- 440 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- 441 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- 442 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- 443 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- 444 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- 445 EPrimitiveDataType - .NET (C#)

Syntax	`enum EPrimitiveDataType`
	`{`
	` Unspecific = 0,`
	` Struct = 1,`
	` Bool = 2,`
	` Int8 = 3,`
	` Int16 = 4,`
	` Int32 = 5,`
	` Int64 = 6,`
	` UInt8 = 7,`
	` UInt16 = 8,`
	` UInt32 = 9,`
	` UInt64 = 10,`
	` Float = 11,`
	` Double = 12,`
	` Char = 13,`
	` WChar = 14`
	`}`

Compatible primitive data types

The following tables shows the primitive data types of the user interface (API) and the data types of the 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- 446 Compatible primitive data types - Reading

API	PLCSIM Advanced instance												
	Bool	INT8	INT16	INT32	INT64	UINT8	UINT16	UINT32	UINT64	Float	Double	Char	WChar
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- 447 Compatible primitive data types - Write

API	PLCSIM Advanced instance												
	Bool	INT8	INT16	INT32	INT64	UINT8	UINT16	UINT32	UINT64	Float	Double	Char	WChar
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				
UINT32								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- 448 EDataType - Native C++

Syntax	enum EDataType { SRDT_UNKNOWN = 0, SRDT_BOOL = 1, SRDT_BYTE = 2, SRDT_CHAR = 3, SRDT_WORD = 4, SRDT_INT = 5, SRDT_DWORD = 6, SRDT_DINT = 7, SRDT_REAL = 8, SRDT_DATE = 9, SRDT_TIME_OF_DAY = 10, SRDT_TIME = 11, SRDT_S5TIME = 12, SRDT_DATE_AND_TIME = 14, SRDT_STRUCT = 17, SRDT_STRING = 19, SRDT_COUNTER = 28, SRDT_TIMER = 29, SRDT_IEC_Counter = 30, SRDT_IEC_Timer = 31, SRDT_LREAL = 48, SRDT_ULINT = 49, SRDT_LINT = 50, SRDT_LWORD = 51, SRDT_USINT = 52, SRDT_UINT = 53, SRDT_UDINT = 54, SRDT_SINT = 55, SRDT_WCHAR = 61, SRDT_WSTRING = 62, SRDT_LTIME = 64, SRDT_LTIME_OF_DAY = 65, SRDT_LDT = 66, SRDT_DTL = 67, SRDT_IEC_LTimer = 68, SRDT_IEC_SCounter = 69, SRDT_IEC_DCounter = 70, SRDT_IEC_LCounter = 71, SRDT_IEC_UCounter = 72, SRDT_IEC_USCounter = 73, SRDT_IEC_UDCounter = 74, SRDT_IEC_ULCounter = 75, SRDT_ERROR_STRUCT = 97, SRDT_NREF = 98,

```
SRDT_CREF = 101,
 SRDT_AOM_IDENT = 128,
 SRDT_EVENT_ANY = 129,
 SRDT_EVENT_ATT = 130,
 SRDT_EVENT_HWINT = 131,
 SRDT_HW_ANY = 144,
 SRDT_HW_IOSYSTEM = 145,
 SRDT_HW_DPMASTER = 146,
 SRDT_HW_DEVICE = 147,
 SRDT_HW_DPSLAVE = 148,
 SRDT_HW_IO = 149,
 SRDT_HW_MODULE = 150,
 SRDT_HW_SUBMODULE = 151,
 SRDT_HW_HSC = 152,
 SRDT_HW_PWM = 153,
 SRDT_HW_PTO = 154,
 SRDT_HW_INTERFACE = 155,
 SRDT_HW_IEPORT = 156,
 SRDT_OB_ANY = 160,
 SRDT_OB_DELAY = 161,
SRDT_OB_TOD = 162,
 SRDT_OB_CYCLIC = 163,
 SRDT_OB_ATT = 164,
 SRDT_CONN_ANY = 168,
 SRDT_CONN_PRG = 169,
 SRDT_CONN_OUC = 170,
 SRDT_CONN_R_ID = 171,
 SRDT_PORT = 173,
 SRDT_RTM = 174,
 SRDT_PIP = 175,
 SRDT_OB_PCYCLE = 192,
 SRDT_OB_HWINT = 193,
 SRDT_OB_DIAG = 195,
 SRDT_OB_TIMEERROR = 196,
 SRDT_OB_STARTUP = 197,
 SRDT_DB_ANY = 208,
 SRDT_DB_WWW = 209,
 SRDT_DB_DYN = 210,
 SRDT_DB = 257
};
```

Table 7- 449 EDataType - .NET (C#)

Syntax	`public enum EDataType`
	`{`
	` Unknown = 0,`
	` Bool = 1,`
	` Byte = 2,`
	` Char = 3,`
	` Word = 4,`
	` Int = 5,`
	` DWord = 6,`
	` DInt = 7,`
	` Real = 8,`
	` Date = 9,`
	` TimeOfDay = 10,`
	` Time = 11,`
	` S5Time = 12,`
	` DateAndTime = 14,`
	` Struct = 17,`
	` String = 19,`
	` Counter = 28,`
	` Timer = 29,`
	` IEC_Counter = 30,`
	` IEC_Timer = 31,`
	` LReal = 48,`
	` ULInt = 49,`
	` LInt = 50,`
	` LWord = 51,`
	` USInt = 52,`
	` UInt = 53,`
	` UDInt = 54,`
	` SInt = 55,`
	` WChar = 61,`
	` WString = 62,`
	` LTime = 64,`
	` LTimeOfDay = 65,`
	` LDT = 66,`
	` DTL = 67,`
	` IEC_LTimer = 68,`
	` IEC_SCounter = 69,`
	` IEC_DCounter = 70,`
	` IEC_LCounter = 71,`
	` IEC_UCounter = 72,`
	` IEC_USCounter = 73,`
	` IEC_UDCounte = 74,`
	` IEC_ULCounter = 75,`
	` ErrorStruct = 97,`
	` NREF = 98,`
	` CREF = 101,`

	Aom_Ident = 128, Event_Any = 129, Event_Att = 130, Event_HwInt = 131, Hw_Any = 144, Hw_IoSystem = 145, Hw_DpMaster = 146, Hw_Device = 147, Hw_DpSlave = 148, Hw_Io = 149, Hw_Module = 150, Hw_SubModule = 151, Hw_Hsc = 152, Hw_Pwm = 153, Hw_Pto = 154, Hw_Interface = 155, Hw_IEPort = 156, OB_Any = 160, OB_Delay = 161, OB_Tod = 162, OB_Cyclic = 163, OB_Att = 164, Conn_Any = 168, Conn_Prg = 169, Conn_Ouc = 170, Conn_R_ID = 171, Port = 173, Rtm = 174, Pip = 175, OB_PCycle = 192, OB_HwInt = 193, OB_Diag = 195, OB_TimeError = 196, OB_Startup = 197, DB_Any = 208, DB_WWW = 209, DB_Dyn = 210, DB = 257 }

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- 450 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- 451 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- 452 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- 453 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- 454 EInstanceConfigChanged - Native C++

Syntax	`enum EInstanceConfigChanged`
	`{`
	` SRICC_HARDWARE_SOFTWARE_CHANGED = 0,`
	` SRICC_IP_CHANGED = 1`
	`};`

Table 7- 455 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- 456 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- 457 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- 458 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- 459 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- 460 EDirection - Native C++

Syntax	enum EDirection
	{
	SRD_DIRECTION_INPUT = 0,
	SRD_DIRECTION_OUTPUT = 1
	};

Table 7- 461 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- 462 EDiagProperty - Native C++

Syntax	enum EDiagProperty
	{
	SRP_DIAG_APPEAR = 1,
	SRP_DIAG_DISAPPEAR = 2
	};

Table 7- 463 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- 464 EDiagSeverity - Native C++

Syntax	`enum EDiagSeverity`
	`{`
	` SRDS_SEVERITY_FAILURE = 0,`
	` SRDS_SEVERITY_MAINTENANCE_DEMANDED = 1,`
	` SRDS_SEVERITY_MAINTENANCE_REQUIRED = 2`
	`};`

Table 7- 465 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- 466 ERackOrStationFaultType - Native C++

Syntax	`enum ERackOrStationFaultType`
	`{`
	` SR_RSF_FAULT = 0,`
	` SR_RSF_RETURN = 1`
	`};`

Table 7- 467 ERackOrStationFaultType - .NET (C#)

Syntax	`enum ERackOrStationFaultType`
	`{`
	` Fault = 0,`
	` Return = 1`
	`}`

Restrictions

8

8.1 Overview

Certain actions or events may lead to behavior in PLCSIM Advanced or in STEP 7 which deviates from that of a hardware CPU. Restrictions and possible remedies can be found in the following sections:

- Restrictions with fail-safe CPUs (Page 347)
- OPC UA server (Page 348)
- Web server (Page 349)
- Backing up and restoring the configuration of a PLCSIM Advanced instance (Page 350)
- Restrictions for file paths (Page 350)
- Restrictions for communications services (Page 351)
- Restrictions for instructions (Page 351)
- Restrictions with Motion Control (Page 352)
- Restrictions to local communication via Softbus (Page 354)
- Restrictions of security with VMware vSphere Hypervisor (ESXi) (Page 355)
- Error with overflow of cyclical events and motion OBs (Page 355)
- Deviating I/O values in the STEP 7 user program (Page 356)
- Multiple simulations and possible collision of IP addresses (Page 356)
- Lacking access to an IP address (Page 356)
- Simulation in standby mode (Page 356)
- Error installing the antivirus software from Kaspersky (Page 357)

8.2 Restrictions with fail-safe CPUs

Error downloading program changes

The error message "Corrupt" appears when downloading program changes to an F-CPU. The F-CPU remains in STOP mode when the "Start all" check box is selected.

Remedy

Deselect the "Start all" check box in the "Load results" dialog.

Once the download is complete, set the CPU to RUN mode manually using the RUN button.

8.3 OPC UA server

With OPC UA, data exchange is performed through an open, standardized and manufacturer-independent communication protocol. The CPU acting as the OPC UA server can communicate with OPC UA clients, for example, with HMI panels V14 and SCADA systems.

For technical reasons, the security settings in PLCSIM Advanced differ from a hardware CPU. Some features are disabled for simulations or are available to a limited extent.

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.

OPC UA security settings

Based on the OPC UA security settings, the same settings can be made in STEP 7 for the hardware CPU. However, PLCSIM Advanced does not take these security settings into consideration. This ensures that the user does not have his project change to perform a simulation.

Certificate

Note

Server certificate

PLCSIM Advanced uses its own certificate in the firmware and not that of STEP 7. The certificate need not be changed for simulations. However, it does not have the same security level as a downloaded server certificate and cannot be used for secure connections!

- **Server Security Endpoints**

  PLCSIM Advanced only supports Security Endpoint "none".

- **Client certificate**

  PLCSIM Advanced does not evaluate certificates the imported and configured in STEP 7 . PLCSIM Advanced accepts all client certificates automatically. This setting cannot be changed.

- **User authentication**

  PLCSIM Advanced does not use the user name configured in STEP 7.

  Only one logon is possible as "guest" or "anonymous".

User authorization for OPC UA

The PLCSIM Advanced license also contains the user authorization for OPC UA.

The user authorization applies for three instances.

8.4 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 V2.0.

- 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

- Logon

  "PLCSIM" is preset as the user. There is no logon for users.

  A user configured in STEP 7 and his rights have no effect on the "PLCSIM" user.
- There is no access via the secure transmission protocol "HTTPS".
- 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.

Number of maximum connections

A fixed value of 384 is displayed in the Web server for the maximum possible number of connections.

Remedy

The correct value can be found in the technical data for the loaded CPU.

8.5 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.

8.6 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`.

8.7 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.

Remedy

Set "PLCSIM Virtual Ethernet Adapter" (TCP/IP) as the communication interface in PLCSIM Advanced.

Secure TCP connections

PLCSIM Advanced does not support TLS (Transport Layer Security) and has no certificate management. It is therefore not possible to establish secure TCP connections from the simulation.

This means that the following connections are not supported:

- Secure OUC connections (Secure Open User Communication)
- Secure connections to a mail server using TMAIL_C
- HTTPS connections to the Web server

8.8 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

8.9 Restrictions with Motion Control

8.9.1 MCR and EMCR

There are Motion Control resources (MCR) and Extended Motion Control resources (EMCR) on each CPU that can be distributed to the technology objects.

Note

PLCSIM Advanced does not support the complete quantity structure of MCR und EMCR. The simulation of motion control is therefore restricted for the following CPUs:

Table 8- 1 CPUs with limited Motion Control resources (MCR, EMCR) during the simulation

Type	Motion Control resources	
	MCR	EMCR
CPU 1516T-3 PN/DP CPU 1516TF-3 PN/DP	5120	128
CPU 1517-3 PN/DP CPU 1517F-3 PN/DP	5120	-
CPU 1517T-3 PN/DP CPU 1517TF-3 PN/DP	5120	128
CPU 1518-4 PN/DP CPU 1518F-4 PN/DP CPU 1518-4 PN/DP ODK[1] CPU 1518F-4 PN/DP ODK[1] CPU 1518-4 PN/DP MFP[1] CPU 1518F-4 PN/DP MFP[1]	5120	-

[1] The ODK functionality of this CPU is not simulated.

Exceeding the quantity structure

STEP 7 checks whether the quantity structure for a configured CPU is complied with, and alerts you when it is exceeded. When you load a project into a CPU, another message appears.

Note

A project that uses more MCRs / EMCRs than PLCSIM Advanced supports can be downloaded to a virtual controller. However, there is no message indicating that the quantity structure has been exceeded.

Only when the user program accesses the technology objects do you recognize by the error message at the block and the value 0 of the object that motion control objects cannot be used due to an overrun of the quantity structure.

8.9.2 Technology modules

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

It is possible to download a project with technology modules for operation of motion control. However, the built-in logic of the technology modules is not part of the simulation, which is why the associated Motion Control instructions are not supported.

Additional information

For more information on Motion Control, refer to the following manuals:

- Device manuals to support SIMATIC controllers (https://support.industry.siemens.com/cs/ww/en/view/109744173)
- Function manual S7-1500 Motion Control (https://support.industry.siemens.com/cs/ww/en/view/109739589)
- Function manual S7-1500T Motion Control (https://support.industry.siemens.com/cs/ww/en/view/109481326)

8.9.3 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.

Remedy

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.

8.10 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.

Remedy

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 139)

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 136)

8.11 Restrictions of security with VMware vSphere Hypervisor (ESXi)

When you use the virtualization platform VMware vSphere Hypervisor (ESXi), you must change the policy exception to communicate over TCP/IP.

Remedy

Accept the "Promiscuous mode and "Forged transmit" options for the Virtual Switch of the ESXi.

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.

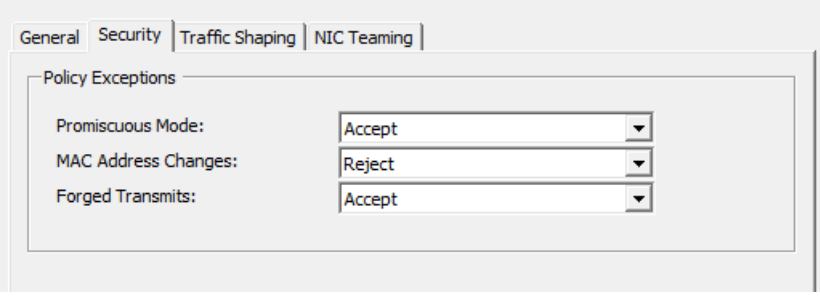


Figure 8-1 Policy exceptions for VMware vSphere Hypervisor (ESXi)

8.12 Error with overflow of cyclical events and motion OBs

If your simulation contains cyclic interrupts, the queue of PLCSIM Advanced may overflow for cyclic events and for motion OBs. 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 75)

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 68)

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 Error installing the antivirus software from 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 in the Control Panel).

Remedy

Check your network settings as described in the sectionEnable distributed communication (Page 56).

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)
EMCR	Extended Motion Control resources
ES	Engineering System
HMI	Human Machine Interface user interface
IE	Industrial Ethernet
GUI	Graphical User Interface
LAN	Local Area Network Computer network that is limited to a local area.
MCR	Motion Control resources
OB	Organization Block
ODK	Open Development Kit
OPC UA	Open Platform Communications Unified Architecture
PG	Programming device
PLC	Programmable Logic Controller
PN	PROFINET
RAM	Random Access Memory
RT	Runtime
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