SIEMENS

SIMATIC

S7/HMI SIMATIC Automation Tool V3.1 User Guide

Manual

V3.1, 10/2017

A5E33042676-AF

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

Ⓟ 09/2017 Subject to change

Preface

Security information

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

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

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

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

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

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

Service and support

In addition to this user guide, Siemens offers technical expertise on the Internet and on the Siemens automation Web site (https://www.siemens.com/automation/) and the Siemens Industry Online Support Web site (http://support.industry.siemens.com). Contact your Siemens distributor or sales office for assistance in answering any technical questions, for training, or for ordering S7 products. Because your sales representatives are technically trained and have the most specific knowledge about your operations, process and industry, as well as about the individual Siemens products that you are using, they can provide the fastest and most efficient answers to any problems you might encounter.

Document source language

The English version of the *SIMATIC Automation Tool user guide* is the authoritative (original) language for SIMATIC Automation Tool information. Siemens identifies the English manual as the authoritative original source in the case of discrepancies between the translated manuals.

Table of contents

Software license and product updates 1

1.1 Software license

Software license requirement

The SIMATIC Automation Tool requires a software license for full feature operation.

Unlicensed operation

Without a license, the SIMATIC Automation Tool enforces the following limitations:

- You can only perform one operation to a single device at a time
- You cannot use the API (Application Programming Interface) for custom application programming.
- You refresh only one device at a time.

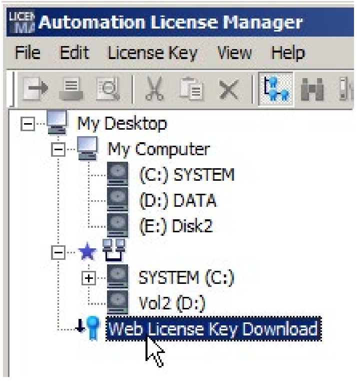
Getting a license

You can purchase a license online through an email account at the Siemens Mall and install it with the Automation License Manager (ALM) software.

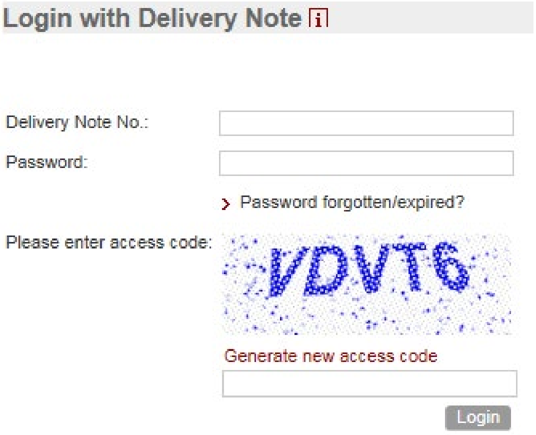
Installing a license

If not already installed, the SIMATIC Automation Tool installation also installs/updates the ALM license manager.

1. Order the product from the Siemens Mall.
2. An email is sent to you that provides a "Delivery Note No." and a temporary password.
3. Start the ALM (Automation License Manager) application on your PG/PC:
4. Double-click "Web License Key Download" from the navigation pane. Wait until the SIEMENS Online Software Delivery page appears in the ALM window.



5. Enter data in SIEMENS Online Software Delivery page:
   - Delivery Note No.
   - Password
   - Access code (read from image)



Login with Delivery Note

Delivery Note No.:

Password:

6. Download software with the Available button.
7. Drag the license rectangle to a directory on your hard drive. Drop the license rectangle on a computer location shown in the right-side tree in the Automation License manger.

1.2 SIMATIC Automation Tool software updates

Getting automatic software updates

The TIA Software updater software is installed during the SIMATIC Automation Tool installation process. If your PG/PC is connected to the Internet, then you can download SIMATIC Automation Tool software updates directly from Siemens through the Internet.

The SIMATIC Automation Tool allows you to automatically search for available updates to installed SIMATIC software products. If updates are available, you have the option to install the updates on your PG/PC.

SIMATIC Automation Tool overview 2

2.1 Managing networks

Managing networks of SIMATIC devices

After you create, verify, and download control programs with the Siemens TIA Portal, you can use the SIMATIC Automation Tool for configuring, operating, maintaining, and documenting devices on automation networks.

Note

You cannot use the SIMATIC Automation Tool to read devices on your automation network if S7-PLCSIM is running on the same PG/PC (programmer/personal computer) at the same time as the SIMATIC Automation Tool. You must close S7-PLCIM for the SIMATIC Automation Tool to be able to read a device network.

Automation Tool operations for configuration, operation, and maintenance

The SIMATIC Automation Tool allows you to perform many useful operations on a network of devices:

Network operations

- Scan the network and create a device table that shows the accessible devices on the network. (Page 46) The device table includes CPUs, modules, HMIs, and other Siemens devices. You can save the device table (Page 89) in a secure *.sat project file or an open text .csv file.
- Identify devices by flashing LEDs or HMI screens (Page 55)
- Update device addresses (IP, subnet, gateway) (Page 51)
- Update the PROFINET name (station name) of a device (Page 53)

System configuration

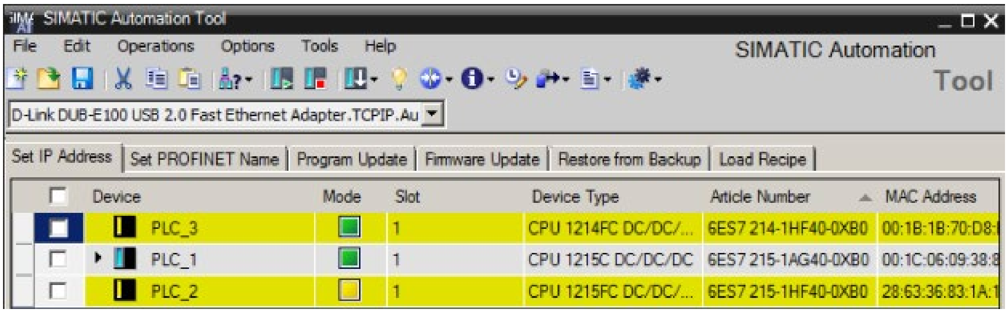
- Set the time in a CPU to the current time in your PG/PC (Programmer/Personal Computer) (Page 85)
- Update a CPU program or HMI operating system and runtime software (Page 56)
- Upload, add, replace, or delete Recipe data from a CPU (Page 64)
- Upload or delete Data Log data from a CPU (Page 67)
- Update module firmware (Page 69)

Operation

- Put a CPU in RUN or STOP mode (Page 54)

System diagnostics and maintenance

- Backup/Restore data to/from a backup file for CPU or HMI device (Page 74)
- Show the diagnostic buffer of a CPU (Page 86)
- Reset CPU memory (Page 78)
- Retrieve Service Data from a CPU (Page 83)
- Format the SIMATIC memory card that is installed in a CPU (Page 81)
- Reset devices to factory default values (Page 79)
- Document and save your network information in a standard text .csv file or a password protected .sat file (Page 242)



☐	Device	Mode	Slot	Device Type	Article Number	MAC Address
☐	PLC_3	🟩	1	CPU 1214FC DC/DC/...	6ES7 214-1HF40-0XB0	00:1B:1B:70:D8:
☐	PLC_1	🟩	1	CPU 1215C DC/DC/DC	6ES7 215-1AG40-0XB0	00:1C:06:09:38:8
☐	PLC_2	🟨	1	CPU 1215FC DC/DC/...	6ES7 215-1HF40-0XB0	28:63:36:83:1A:1

Figure 2-1 SIMATIC Automation Tool device table

Network device support

The SIMATIC Automation Tool supports the following types of Siemens devices:

- Standard CPUs
- Fail-Safe CPUs
- HMIs
- PROFINET devices

The DCP (Discovery and Configuration Protocol) operations can work with supported Siemens PROFINET devices, unsupported Siemens PROFINET devices, and unsupported non-Siemens PROFINET or Ethernet devices.

DCP MAC address based operations list:

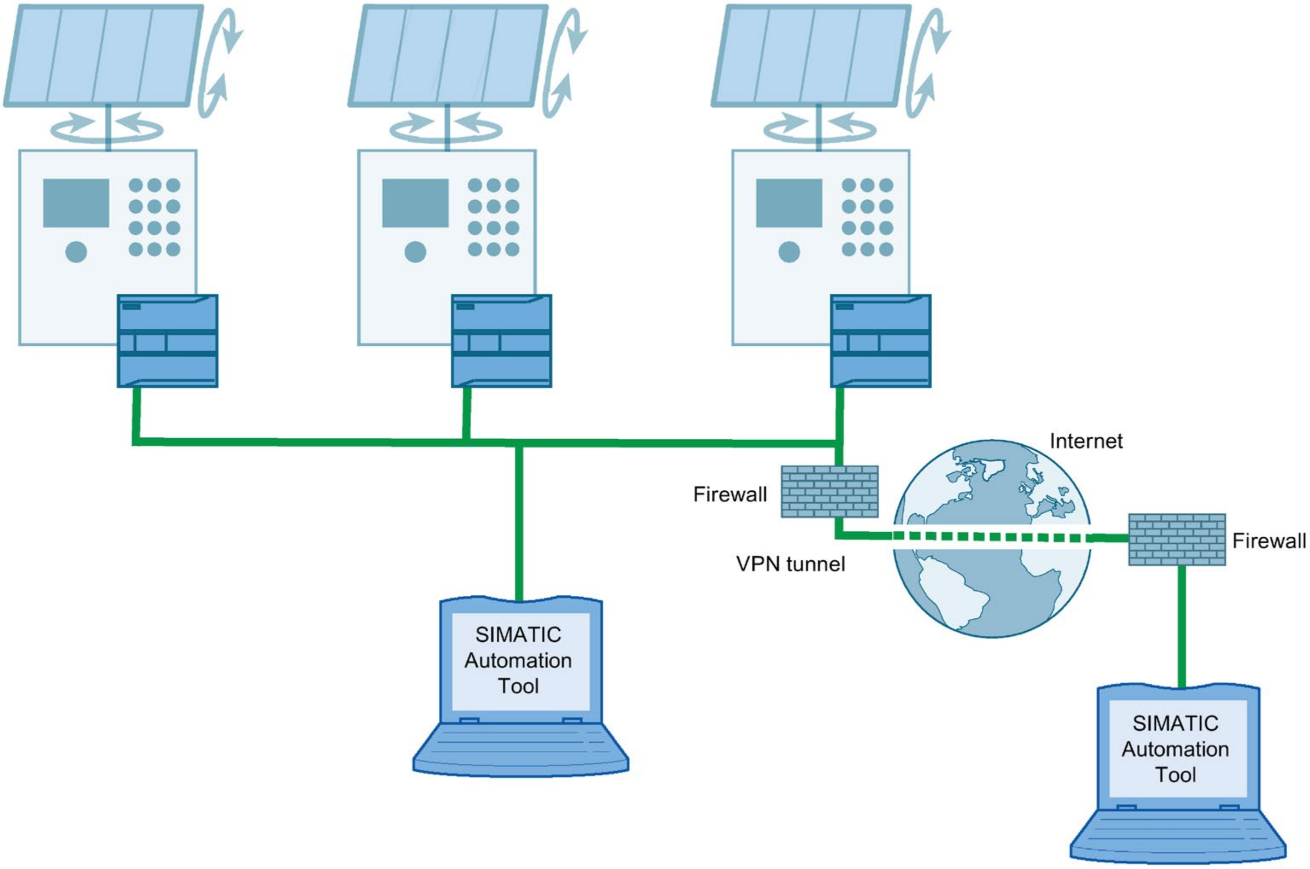
- Scan a network
- Identify devices (Flash device LED/HMI screen)
- Update IP address, subnet mask, and gateway address on devices
- Update PROFINET device name

For information on supported Siemens devices and operations see SIMATIC Automation Tool device support (Page 241)

Example S7-1200 network installation

Panel shop fabrication and initial program load

- Each CPU uses the same hardware configuration and control program.
- The CPU PROFINET configuration uses "Set IP Address on the device" and "Set PROFINET device name on the device" options.
- Each Panel is identical.
- You can use the SIMATIC Automation Tool, a SIMATIC SD memory card, or the TIA Portal to load CPU/HMI devices with project data.



If you are managing a network with many devices, the SIMATIC Automation Tool can simplify operations and save time. You can process an operation on multiple devices at the same time. The SIMATIC Automation Tool uses multi-threading, which means many operations can run concurrently. While the PG/PC is performing an operation on one device, another thread can begin the operation on another device. Multi-threading is the default setting in the Communications settings (Page 99). You can disable multi-threading if you choose.

See also

ET 200AL SM and IO-Link support (Page 242)

ET 200eco support (Page 243)

ET 200M IM support (Page 244)

ET 200MP IM support (Page 244)

ET 200S (Page 245)

ET 200pro IM support (Page 247)

ET 200pro IO-Link, RFID support (Page 247)

ET 200SP CPU support (based on S7-151x) (Page 248)

ET 200SP IM and Server module support (Page 249)

ET 200SP SM, AS-i, CM, CP, TM, IO-Link, Motorstarter support (Page 250)

S7-1200 CPU support (Page 252)

S7-1200 I/O and other module support (Page 253)

S7-1500 CPU support (Page 257)

S7-1500 I/O and other module support (Page 258)

HMI Basic panels support (Page 259)

HMI Comfort panels support (Page 260)

HMI Mobile panels support (Page 261)

SITOP support (Power supply) (Page 262)

RFID (Radio Frequency Identification) (Page 263)

MOBY (DeviceNet interface) (Page 263)

2.2 Network access

Communicating with network devices

The SIMATIC Automation Tool network scan discovers network devices connected directly to a network by using the MAC (Media Access Control) address. A MAC address is unique to each device, cannot be changed, and is printed on the device. Connected devices are discovered whether they are configured with an IP, subnet, and gateway address, or not configured (addresses are 0.0.0.0).

MAC address based operations use the DCP (Discovery and Configuration Protocol). DCP is an Ethernet standard. The SIMATIC Automation Tool can use the DCP-MAC address operations Scan Entire Network, Identify, Update IP address, and Update PROFINET name for all directly connected network devices (supported PROFINET devices, unsupported PROFINET devices, and unsupported non-Siemens Ethernet devices).

The IP address based operations only work for supported Siemens devices.

Simple network with one CPU and local I/O:

- MAC address based operations are possible
- Valid IP address and TIA Portal Device configuration in CPU is necessary, for the network scan to display a CPU's local I/O devices and to enable IP address based operations.

Multiple CPUs with local I/O connected through an Ethernet switch:

- MAC address based operations are possible for all devices
- Valid IP address and TIA Portal Device configuration in CPU is necessary, for the network scan to display the CPU's local I/O devices and to enable IP address based operations.

CPUs in complex networks with decentralized I/O and IP address routers:

- MAC address based operations are possible only for devices that are directly connected to the same subnet as the PG/PC running the SIMATIC Automation Tool.
- Valid IP address and TIA Portal Device configuration in the CPUs is necessary, for the network scan to display devices located behind the CPUs and to enable IP address based operations.

For example, an S7-1500 CPU has a PROFINET network connection to the SIMATIC Automation Tool and uses a local CP module to connect with another PROFINET network where decentralized I/O devices are connected. You must assign a valid IP address to the S7-1500 CPU and successfully compile and download your project's TIA portal device configuration, before the decentralized I/O network is visible in the SIMATIC Automation Tool Device table.

Note that only the Firmware Update operation is possible for devices that are indirectly connected behind a CPU, CM, or CP module.

The type of network access you have depends on the command that you execute, as shown in the following table.

SIMATIC Automation Tool command	Device address used	Must provide CPU password for a protected CPU	PG/PC and device connectivity
Scan (discover CPUs, HMIs, I/O, and other devices)	MAC	No	• **Local network**: You can access network devices through Ethernet switches, but cannot access devices on another network through an IP address router. • **VPN** (Virtual Private Network) connection to the local network
Identify devices (Flash LED/HMI screen)	MAC	No	
Set IP address, subnet mask, and gateway address on devices	MAC	No	
Set PROFINET name on devices	MAC	No	
Reset devices to factory default values (for PROFINET I/O devices only)	MAC	Not applicable	
Put CPUs in RUN or STOP	IP	Yes[1]	• **Local network**: You can access network devices through Ethernet switches. • **Remote network**: You can access devices on another network through an IP address router. • **VPN** connection to the local network
Set CPU time to PG/PC time	IP	Yes[1]	
Perform Program Update: CPU programs	IP	Yes[3]	
Perform Program Update: HMI update for operating system and runtime software	IP	No	
Upload Recipe data from a CPU	IP	Yes[2]	
Add, replace, or delete Recipe data from a CPU	IP	Yes[1]	
Upload Data Log data from a CPU	IP	Yes[2]	
Delete Data Log data from a CPU	IP	Yes[1]	
Backup CPU to a file	IP	Yes[2]	
Backup HMI to a file	IP	No	
Restore CPU from backup file	IP	Yes[3]	
Restore HMI from backup file	IP	No	
Upload Service Data from CPUs	IP	Yes[2]	
Read CPU Diagnostic buffer	IP	Yes[2]	
Reset CPU memory	IP	Yes[3]	
Reset devices to factory default values	IP	Yes[3]	
Update firmware in devices	IP	Yes[3]	
Format memory card	IP	Yes[3]	

[1] Requires the "Full access (no protection)" access level for all CPUs.

[2] Requires the "Read access" access level for all CPUs.

[3] HMI devices: No password required.
Standard CPUs: Requires the "Full access (no protection)".
Fail-Safe CPUs: Requires either the "Full access (no protection)" or the "Full access incl. fail-safe (no protection)" access level, depending on the firmware version in the CPU. You must also reselect and confirm the device or devices for the operation.

Note

IP subnets and network interface protocols

The PG/PC that runs the SIMATIC Automation Tool and the devices connected to your local network must use appropriate IP subnet assignments.

The type of network interface protocol that you select ("TCPIP" or "TCPIP.Auto") can affect whether the SIMATIC Automation Tool Network is able to discover devices with the network scan operation.

See the example in the Communication setup topic (Page 30).

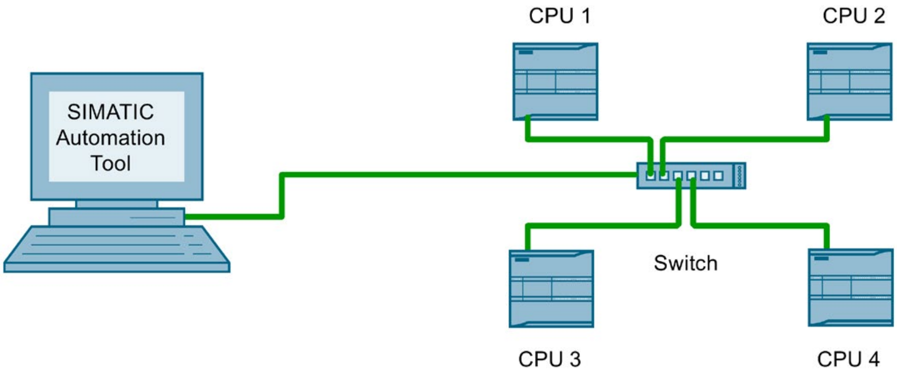
2.3 Network options

Local and remote networks

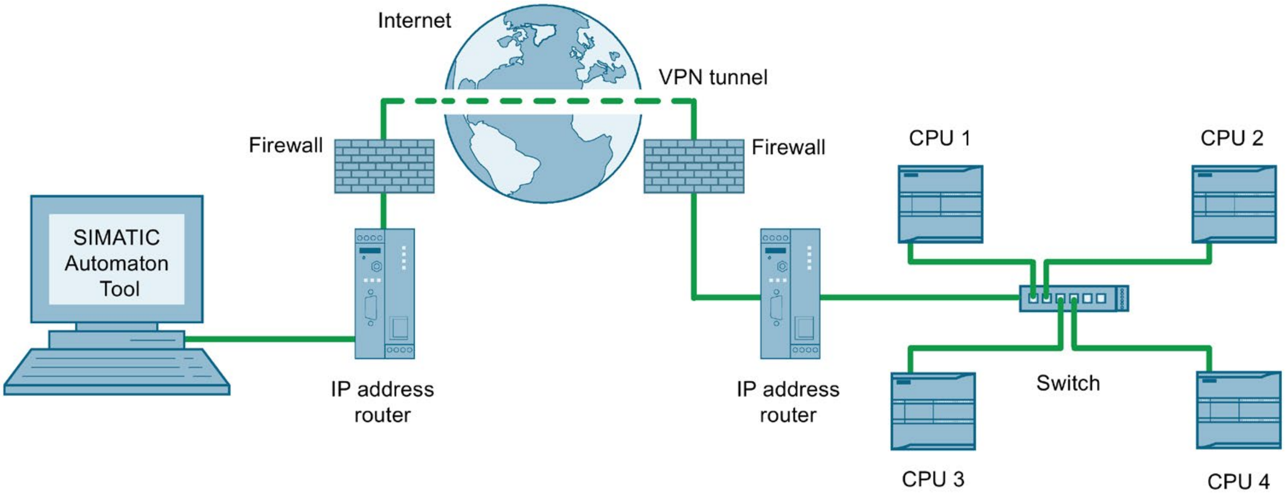
The following examples show local and remote networks that the SIMATIC Automation Tool can use. These simplified diagrams show basic connectivity and do not show HMI devices, local I/O, distributed I/O devices (PROFINET and PROFIBUS), and other devices that are also accessible. Different network topologies are also possible.

You can fill the SIMATIC Automation Tool Device table by scanning a network (Page 46). The device data in an existing Device table can be exported in .csv format. You can modify existing device data in the .csv text and use the import command to bring the new data into the SIMATIC Automation Tool Device table

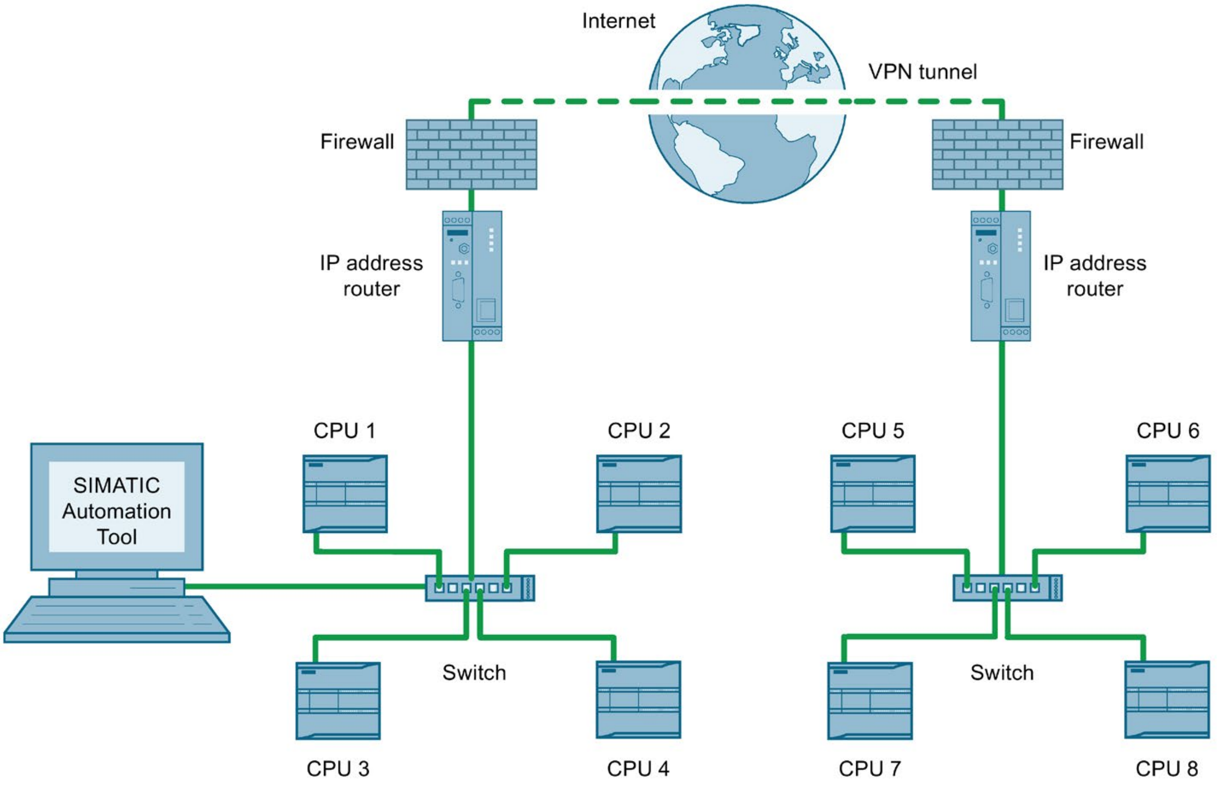
Example 1: S7-1200 local network



Example 2: S7-1200 remote network



Example 3: S7-1200 combined local and remote networks



See also

Import/Export - Device table loaded from/stored in open .csv format (Page 90)

2.4 .NET API (application interface) .dll file

You can create your own application software that uses the SIMATIC Automation Tool Microsoft .NET API (Page 111) to perform the same device operations as the SIMATIC Automation Tool.

The SIMATIC Automation Tool must be installed on any PG/PC that uses this API. The SIMATIC Automation Tool and your application software use the API .dll file and additional S7 communication files.

The SIMATIC Automation Tool installation provides all the files that you need.

The API files are located in the folder where the SIMATIC Automation Tool is installed.

Software license required for V3.0 and later versions

The API is disabled in unlicensed mode when operating V3.0 or later versions.

If you have a valid license for the SIMATIC Automation Tool, the API is enabled when operating V3.0 and later versions.

Prerequisites and communication setup 3

3.1 PG/PC Operating system, VM software, and security software support

Microsoft Windows 64-bit operating systems support

The SIMATIC Automation Tool works with the following 64-bit operating systems.

- Windows 7 Home Premium SP1
- Windows 7 Professional SP1
- Windows 7 Enterprise SP1
- Windows 7 Ultimate SP1
- Windows 10 Home Version 1607 (OS Build 14393)
- Windows 10 Pro Version 1607 (OS Build 14393)
- Windows 10 Enterprise Version 1607 (OS Build 14393)
- Windows 10 Enterprise 2016 LTSB (OS Build 14393)
- Windows 10 IoT Enterprise 2015 LTSB (OS Build 10240)

You can install the SIMATIC Automation Tool and use the unlicensed version to test operations on other Windows 64-bit operating systems. The SIMATIC Automation Tool may install and work correctly with these operating systems. Siemens does not guarantee that the SIMATIC Automation Tools works with other Windows 64-bit operating systems and does not provide technical support in these cases.

Virtual machine software support

The SIMATIC Automation Tool works with the following VM (Virtual Machine) software

- VMware Workstation 12.5
- VMware Player 12.5

Virus and security software support

The SIMATIC Automation Tool works with the following virus and security software

- Symantec Endpoint Protection 14
- McAfee VirusScan Enterprise 8.8
- Trend Micro Office Scan Corporate Edition 12.0
- Kaspersky Anti-Virus 2017
- Windows Defender (as part of Windows operating systems)
- Qihoo "360 Total Security Essential" 8.8 (for Chinese market)
- McAfee Application Control 7.0.1
- Microsoft Bitlocker (part of the Windows operating systems)

The SIMATIC Automation Tool may install and work correctly with other virus and security software. Siemens does not guarantee that the SIMATIC Automation Tools works with other virus and security software and does not provide technical support in these cases.

3.2 Installing the SIMATIC Automation Tool

Save all your work in progress and close all PG/PC applications before installing the SIMATIC Automation Tool.

Installation rules

You can only install one version of the SIMATIC Automation Tool on a PG/PC. If you have installed a previous version, you must uninstall it first. The installation executable checks for a previous installation, and responds as follows:

- If no version of SIMATIC Automation Tool is found, the installation can proceed.
- If a version older than V3.0 of the SIMATIC Automation Tool is found, the setup informs you that you must uninstall the older version. The setup guides you through uninstalling the older version. You cannot proceed with the installation until you close and restart the installation.
- If version 3.0 is found, the installation process uninstalls the V3.0 version and installs V3.1.
- If an existing current version V3.1 of the SIMATIC Automation Tool is found, the setup presents options to modify/upgrade, repair, or uninstall the previous installation.

Note

You can install the SIMATIC Automation Tool on any device that has the required space as indicated by the setup. You must, however, have at least 1.4 GB free on the C:\ drive for system files.

3.3 Starting the SIMATIC Automation Tool

Use one of the following methods to start the SIMATIC Automation Tool:

- Double-click the SIMATIC Automation Tool shortcut icon on your desktop.
- Use the Windows Start button:
  - Click the Windows start button and "All Programs".
  - Click the "Siemens Automation" folder, then the "SIMATIC Automation Tool" folder, and finally "SIMATIC Automation Tool".

3.4 Configuration requirements

If you want the SIMATIC Automation Tool to set the IP address or PROFINET name of a device, then the device's TIA portal project must enable these actions in the project's Device Configuration. Compile the project and download the project to the target device, before attempting to change a device's IP address or PROFINET name.

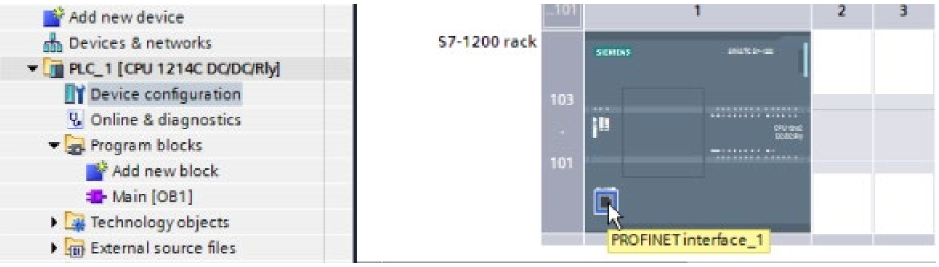
Update IP address and Update PROFINET name are DCP operations that use a MAC address to access the target device. Devices that are indirectly connected must use an IP address for access and cannot use the DCP operations. You must temporarily make a direct connection between the target device and the SIMATIC Automation Tool to change the IP address or PROFINET name.

Update IP address and PROFINET name operations

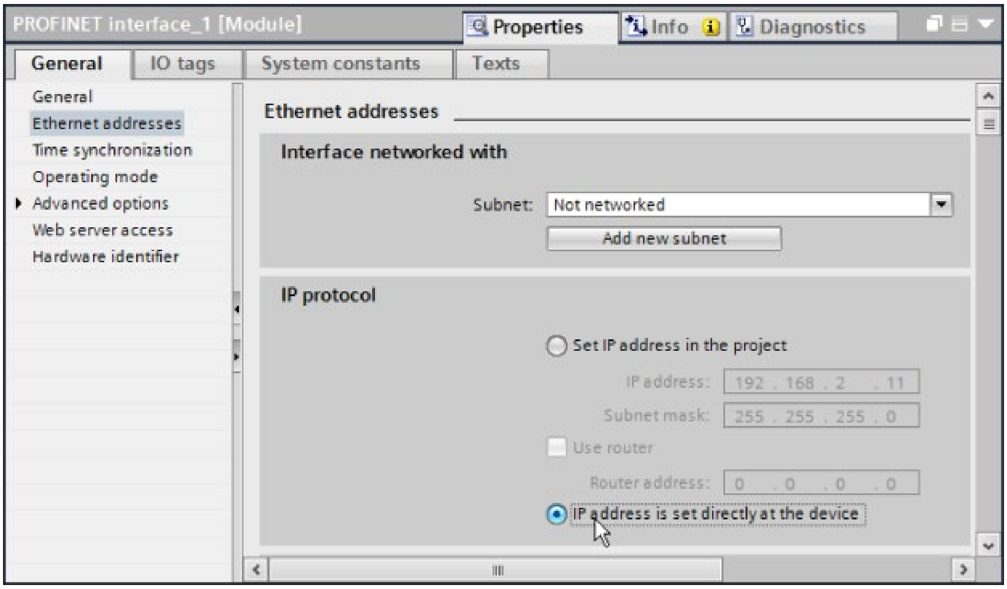
- **Possible** for PROFINET devices (CPUs, HMIs, decentralized I/O, and other devices) directly connected to the network subnet that is connected to the SIMATIC Automation Tool, including connection through an Ethernet switch.
- **Not possible** for PROFINET devices with an indirect connection behind a directly connected CPU, CP/CM module, interface module, or a CPU's second Ethernet port.
- **Not possible** for PROFINET devices on another network with a connection to the SIMATIC Automation Tool that passes through an IP address router.

Example S7-1200 configuration with TIA Portal software

1. Click the PROFINET port on the device configuration CPU image to view the port parameters.



2. On the **Properties** tab, click the **General** tab to view the **Ethernet addresses** options. Click the **IP address is set directly at the device** option. This option may be called "Set IP address on the device" or "Set IP address using a different method", depending on which TIA portal version you are using. For multi-port devices like the S7-1500 CPU, you can similarly configure all ports to enable IP address changes (when connected to the SIMATIC Automation Tool) or you can configure only the port you want to change.



Ethernet addresses

Interface networked with

Subnet: Not networked

IP protocol

○ Set IP address in the project

IP address: 192 . 168 . 2 . 11

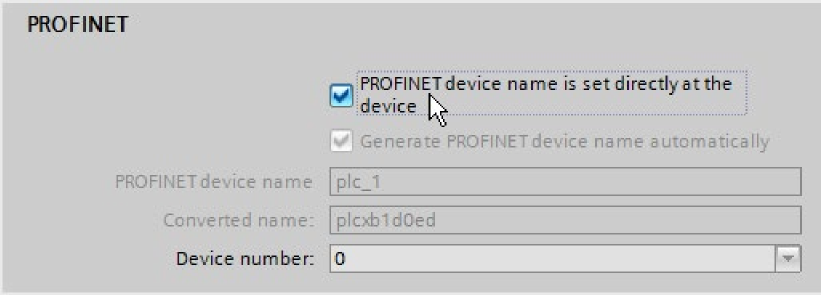
Subnet mask: 255 . 255 . 255 . 0

□ Use router

Router address: 0 . 0 . 0 . 0

● IP address is set directly at the device

3. Also on the **Ethernet addresses** options, click the **PROFINET device name is set directly at the device** option. This option may be called "Set PROFINET device name on the device", depending on which TIA portal version you are using. This selection allows the SIMATIC Automation Tool to assign a PROFINET station name. For multi-port devices like the S7-1500 CPU, you can similarly configure all ports to enable PROFINET name changes (when connected to the SIMATIC Automation Tool) or you can configure only the port you want to change.



PROFINET

☑ PROFINET device name is set directly at the device

☑ Generate PROFINET device name automatically

PROFINET device name: plc_1

Converted name: plcxb1d0ed

Device number: 0

4. Save your project and download the new configuration changes to the CPU.

Note

Default settings of PROFINET IP parameters

When you create a new TIA portal project, the default PROFINET parameter options are set to **"Set IP address in the project"** and **"Generate PROFINET device name automatically"**. With the default options, you cannot set IP addresses or PROFINET device names with the SIMATIC Automation Tool. However, you can use other CPU operations like RUN/STOP control, program/firmware updates, time setting, and service data/diagnostic analysis.

3.5 Communication setup

Identifying the network interface card connected to your device network

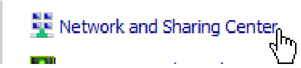
After you connect your PG/PC to a network, you can use the Windows control panel to see the name of the network interface card.

In the following example, The SIMATIC device network is connected by an Ethernet to USB converter to a PC running Windows 7. The network names that you actually see on your PG/PC depend on your network hardware.

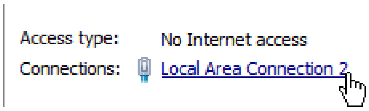
Use the Windows Control Panel to identify the name of the device.

1. Open the Windows Control Panel

2. Click the Network and Sharing center.

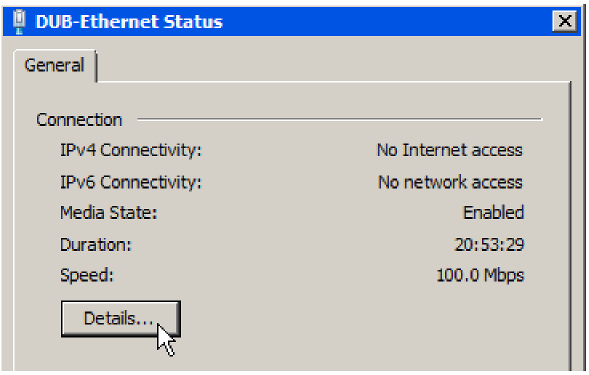


3. View your active networks and click the network that is connected to the S7-1200 CPUs.



Access type: No Internet access
Connections: Local Area Connection 2

4. Click the Details button in the connection status display.



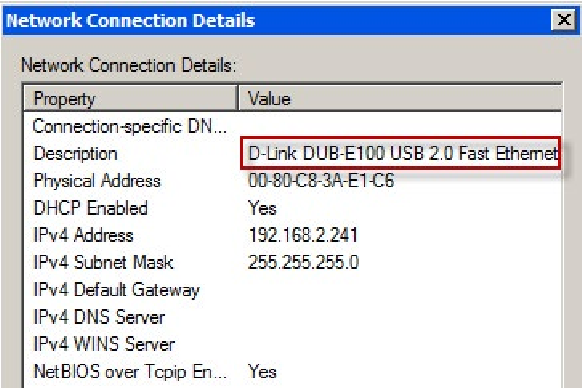
DUB-Ethernet Status

General

Connection

IPv4 Connectivity:	No Internet access
IPv6 Connectivity:	No network access
Media State:	Enabled
Duration:	20:53:29
Speed:	100.0 Mbps

5. View the description of the network interface.



Network Connection Details

Network Connection Details:

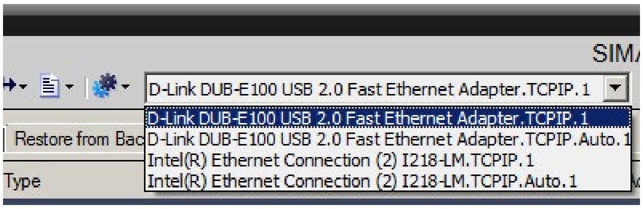
Property	Value
Connection-specific DN...	
Description	D-Link DUB-E100 USB 2.0 Fast Ethernet
Physical Address	00-80-C8-3A-E1-C6
DHCP Enabled	Yes
IPv4 Address	192.168.2.241
IPv4 Subnet Mask	255.255.255.0
IPv4 Default Gateway	
IPv4 DNS Server	
IPv4 WINS Server	
NetBIOS over Tcpip En...	Yes

Assigning the network interface in the SIMATIC Automation Tool

You must assign the network interface to a new project before communication can begin. To set the network interface follow these steps:

1. Start the SIMATIC Automation Tool
2. Click the Network Interface Card drop-down list
3. Select the network interface that is connected to your Siemens device network.

You might see different network interface selections from those shown in the following image, because the list shows the network interfaces that are available in your PG/PC.



D-Link DUB-E100 USB 2.0 Fast Ethernet Adapter.TCPIP.1

D-Link DUB-E100 USB 2.0 Fast Ethernet Adapter.TCPIP.1
D-Link DUB-E100 USB 2.0 Fast Ethernet Adapter.TCPIP.Auto.1
Intel(R) Ethernet Connection (2) I218-LM.TCPIP.1
Intel(R) Ethernet Connection (2) I218-LM.TCPIP.Auto.1

If you have selected a network interface card, but the devices do not have valid IP addresses, then you cannot use the IP address based operations.
You can use the MAC address based operations and set up valid IP addresses for the devices in your network.

MAC address operations

- Scan for network devices (Page 46)
- Identify devices (Page 55)
- Set Ethernet IP addresses (Page 51)
- Set PROFINET names (Page 53)
- Reset to factory default values (Page 79) The MAC address based Reset to factory operation only works for PROFINET I/O devices, other devices use an IP address based Reset to factory operation.

Network interface selection

As seen in the preceding image, there can be two entries for each network card and the difference is the addition of the characters ".Auto".

When you select the Ethernet interface, you have two choices for the type of network protocol:

- TCP/IP
- TCP/IP.Auto

It is recommended that you select **TCPIP** without "Auto" because "virtual" IP addresses are not created automatically in the Windows Ethernet adapter. You must assign a valid IP address in the Windows configuration for your PG/PC Ethernet adapter.

Alternatively, you can also select **TCPIP.Auto**. After you perform a network scan, you must verify that automatically created virtual IP addresses do not conflict with the IP addresses of other devices on the network.

The **TCPIP.Auto** protocol has the following advantages:

- The **TCPIP.Auto** protocol can discover accessible devices that are not discovered by the **TCPIP** protocol.
- You can change the IP addresses of accessible network devices to use a subnet that works with the **TCPIP** protocol.
- After a network scan, the PG/PC network adapter always has valid virtual IP addresses for all your Siemens devices. You do not have to assign new IP addresses explicitly in Windows.

However, the **TCPIP.Auto** protocol may cause network communication problems:

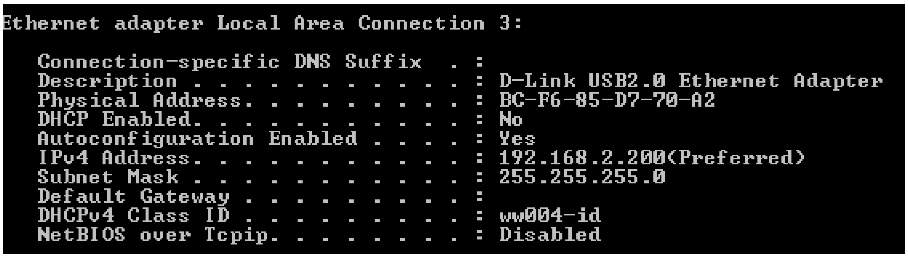
- You cannot assign virtual IP addresses. The Windows operating system automatically assigns virtual IP addresses.
- Virtual addresses are lost after a power cycle or PG/PC reset. New virtual IP addresses are created during the next network scan that uses the **TCPIP.Auto** protocol.
- A virtual IP address might be automatically created that is already used by another node (for example, another PG/PC that is not visible by a SIMATIC Automation Tool Network scan). An address conflict can cause communication errors for some parts of your network that are difficult to diagnose.

Example use of TCPIP and TCPIP.Auto protocols

You can inspect the Windows network adapter IP addresses by entering "`ipconfig /all`" in the command line window.

The "`ipconfig /all`" command was used to obtain the IP addresses shown in the following example.

1. After a PG/PC reset (reboot) and before running a SIMATIC Automation Tool Network scan, execute "`ipconfig /all`" in the command line window. The result for the Ethernet adapter card connected to the Siemens device network is shown below. The Windows Ethernet adapter is configured with the IP address `192.168.2.200`.



```
Ethernet adapter Local Area Connection 3:

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : D-Link USB2.0 Ethernet Adapter
   Physical Address. . . . . . . . . : BC-F6-85-D7-70-A2
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes
   IPv4 Address. . . . . . . . . . . : 192.168.2.200(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . :
   DHCPv4 Class ID . . . . . . . . . : ww004-id
   NetBIOS over Tcpip. . . . . . . . : Disabled
```

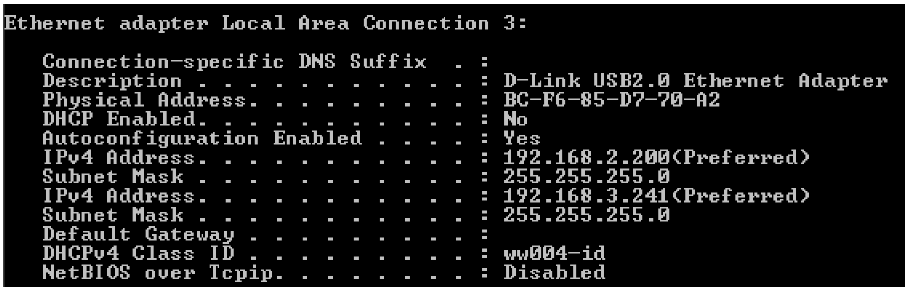
2. Connect a Siemens S7-1200 PLC that is configured with the IP address `192.168.3.1`. The subnet mask is `255.255.255.0`, so the S7-1200 device is actually configured for a different subnet. The third octet is "`3`" and must be "`2`" in order to communicate with the Ethernet adapter's `192.168.2` subnet address.

3. Start the SIMATIC Automation Tool, set the Network interface to the **TCPIP** protocol, and perform a Network scan. In this case, the S7-1200 PLC **is not found** because the S7-1200 PLC is configured with the wrong subnet address.

4. Change the SIMATIC Automation Tool Network interface to the **TCPIP.Auto** protocol and perform a Network scan.

5. The network scan uses the **TCPIP.Auto** protocol and discovers the S7-1200 device. New S7-1200 device information is added to the SIMATIC Automation Tool Device table.

6. Execute "`ipconfig /all`" in the command line window.
   As seen in the following image, an alternate Ethernet adapter virtual IP address `192.168.3.241` was automatically created. The alternate virtual IP address enables access to the `192.168.3` subnet.



```
Ethernet adapter Local Area Connection 3:

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : D-Link USB2.0 Ethernet Adapter
   Physical Address. . . . . . . . . : BC-F6-85-D7-70-A2
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes
   IPv4 Address. . . . . . . . . . . : 192.168.2.200(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   IPv4 Address. . . . . . . . . . . : 192.168.3.241(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . :
   DHCPv4 Class ID . . . . . . . . . : ww004-id
   NetBIOS over Tcpip. . . . . . . . : Disabled
```

   Multiple virtual IP addresses are created when the **TCPIP.Auto** protocol discovers multiple subnets in a complex network.
   The virtual IP addresses are temporary and are deleted if the Windows PG/PC is reset.

7. The SIMATIC Automation Tool can now connect to the S7-1200 device using the `192.168.3.241` virtual IP Address and then change the S7-1200 device's IP address. Use the SIMATIC Automation Tool to update the IP address. Change the IP address from `192.168.3.1` to `192.168.2.1`.
8. Reset the PG/PC and restart Windows. Any virtual IP addresses are deleted after the restart.
9. Start the SIMATIC Automation Tool, set the Network interface to the **TCPIP** protocol, and perform a Network scan.
   The S7-1200 PLC (`192.168.2.1`) **is discovered** and can communicate with the PG/PC Ethernet adapter (`192.168.2.200`).
   No virtual IP addresses are created. Only the IP address configured in the Windows network adapter properties is used.

If the network interface card is selected and the device IP addresses are valid, you can use the SIMATIC Automation Tool operations that use an IP address.

IP address operations

- Put CPUs in RUN or STOP (Page 54)
- Set CPU time to PG/PC time (Page 85)
- Program update for CPU and HMI devices (Page 56)
- Upload, add, replace, or delete Recipe data from a CPU (Page 64)
- Upload or delete Data Log data from a CPU (Page 67)
- Backup/restore CPU and HMI data (Page 74)
- Retrieve Service Data from CPUs (Page 83)
- Show CPU diagnostic buffer (Page 86)
- Reset CPU memory (Page 78)
- Format memory card in a CPU (Page 81)
- Reset devices to factory default values (Page 79)
- Update firmware in devices (Page 69)

Note

Communication problems with the SIMATIC Automation Tool

For example, you send an operation command to multiple devices, but a device does not complete the operation. You see a communication error in the Event Log. Other devices, however, are communicating and executing the operation as you expect. If you have this problem, follow these steps:

1. Reduce the number of simultaneous operations that you allow in the Communications settings (Page 99).
2. Close and restart the SIMATIC Automation Tool.
3. Try the group operation again.

If you send an operation command to a device and the connection has a very slow data transfer rate, then you may get a communication timeout error. If you have this problem, then increase the timeout for communications operations in the Communications settings (Page 99).

Tool operations 4

4.1 CPU passwords

If password protection is configured in a CPU, then you must enter a password for the access level that allows the SIMATIC Automation Tool to perform the operation that you want to use. You provide a password in the device table column titled "Password in CPU".

The password in the "Password in CPU" column refers to the password protection that currently exists in the target CPU.

For example, a new CPU in a packing box from Siemens has no program, no hardware configuration, and no password protection. After a project is loaded in the CPU, you must use the passwords that are configured in that project.

The SIMATIC Automation Tool shows you whether a CPU is password-protected or not. The password cell for a CPU is dark gray and not editable if the CPU has no password.

Using passwords

- If a CPU is password-protected, then you must enter a password in the "Password in CPU" cell for a Program Update or Restore from Backup operation to complete successfully.
- The "Program update" tab has two password entry columns: "Password in CPU" and "Password in Program File".

  A program file might have a password, which can be different from the existing CPU password. When a program file has a password, you must enter the program file password in the "Password in Program File" cell to perform a program update. After a successful Program Update, the SIMATIC Automation Tool copies the program file password to the CPU password cell. Using this password, the SIMATIC Automation Tool then attempts to open a connection to the CPU, using the new password
- The "Restore from Backup" tab has two password entry columns: "Password in CPU" and "Password in Backup File".

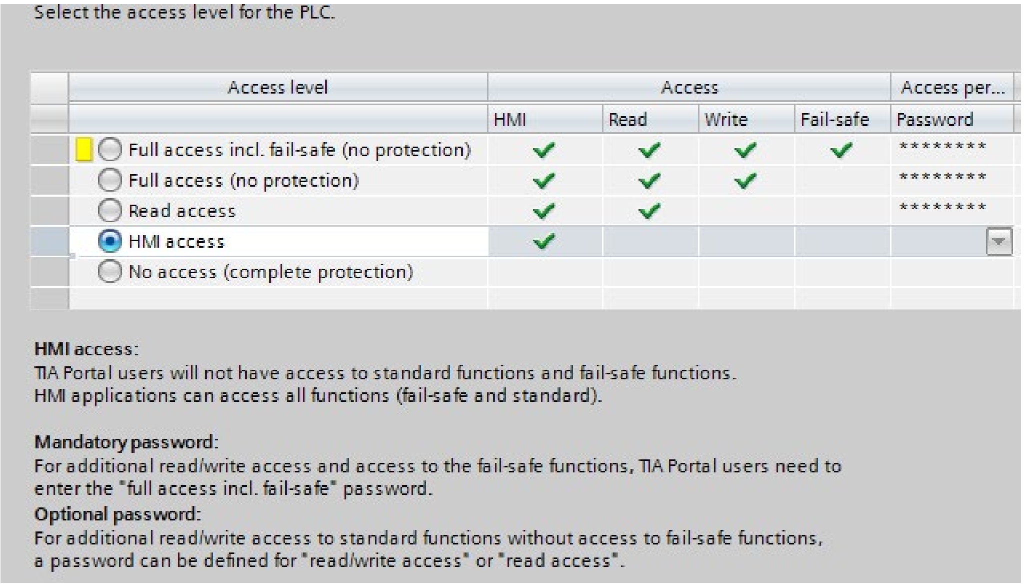
  A backup file might have a password, which can be different from the existing CPU password. When a backup file has a password, you must enter the backup file password in the "Password in Backup File" cell. The backup file password becomes the CPU password after the Restore from Backup operation completes. After a successful Restore from Backup operation, the SIMATIC Automation Tool copies the backup file password to the CPU password cell. Using this password, the SIMATIC Automation Tool then attempts to open a connection to the CPU, using the new password.
- If a CPU password entry is valid, you can hover your mouse cursor over the password field to display a tool tip that shows the access level.

CPU password access levels

A standard CPU has four password access levels and a Fail-Safe CPU has five levels.

The SIMATIC Automation Tool operations that require read or write access cannot work with a CPU that has the "HMI access" or "No access" protection level. You must configure a "Read access" or "Full access" password and then enter that password into the device table row, for the target CPU.

You can see the CPU access levels in the Protection & Security section of the TIA Portal Device configuration:



Select the access level for the PLC.

Access level	Access				Access per...
	HMI	Read	Write	Fail-safe	Password
🟨 ○ Full access incl. fail-safe (no protection)	✓	✓	✓	✓	********
○ Full access (no protection)	✓	✓	✓		********
○ Read access	✓	✓			********
● HMI access	✓				
○ No access (complete protection)					

HMI access:
TIA Portal users will not have access to standard functions and fail-safe functions.
HMI applications can access all functions (fail-safe and standard).

Mandatory password:
For additional read/write access and access to the fail-safe functions, TIA Portal users need to enter the "full access incl. fail-safe" password.
Optional password:
For additional read/write access to standard functions without access to fail-safe functions, a password can be defined for "read/write access" or "read access".

For additional information on access levels and passwords, refer to the *STEP 7 Information System* (online help for the TIA Portal).

Fail-Safe CPUs and passwords

The SIMATIC Automation Tool can work with Fail-Safe CPUs. You can connect to a Fail-Safe CPU and perform some operations using a "Read access" password, or a "Full access (no protection)" password which is described in this guide as the safety F-CPU password. Safety-relevant operations, however, require the safety F-CPU password.

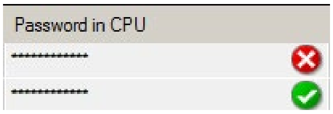
- The SIMATIC Automation Tool displays a device's password cell in yellow when you enter a safety F-CPU password.
- If a Fail-Safe PLC is password-protected, then all safety-relevant operations require that you enter the safety F-CPU password in the "Password in CPU" column.
  If a Fail-Safe CPU does not use password protection, then safety-relevant operations do not require the safety F-CPU password to initiate the operation.
  The safety-relevant operations are:
  - Program Update
  - Restore Device from File
  - Reset to Factory Defaults
  - Format Memory Card

- You must enter the safety F-CPU password in the "Password in Program File" column for a "Program Update" operation, when the program file contains a safety program.
- You must enter the safety F-CPU in the "Password in Backup File" column for a "Restore from Backup File" operation, when the backup file contains a safety program.

Password icons

The SIMATIC Automation Tool provides three password status icons.

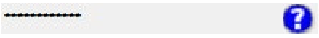
A green checkmark icon means the password is a valid password in the CPU or program file. A red X icon means that password is not a valid password in the CPU or program file.



Password in CPU
************* ✗
************* ✓

When you enter a password, SIMATIC Automation Tool does not know what operations you may start. Therefore, the password is not validated on entry for a specific operation. For example, updating a safety program requires the safety F-CPU password. For many other operations, the standard Full access (read/write) password is sufficient. A green checkmark does not mean the password is validated for any operations. Access level validation occurs when the operation is initiated. If the password is not legitimized to a sufficient level, an event log error is generated, for each device where this error occurs.

The Backup file password cell is the only cell that can display a question mark icon:



************* ?

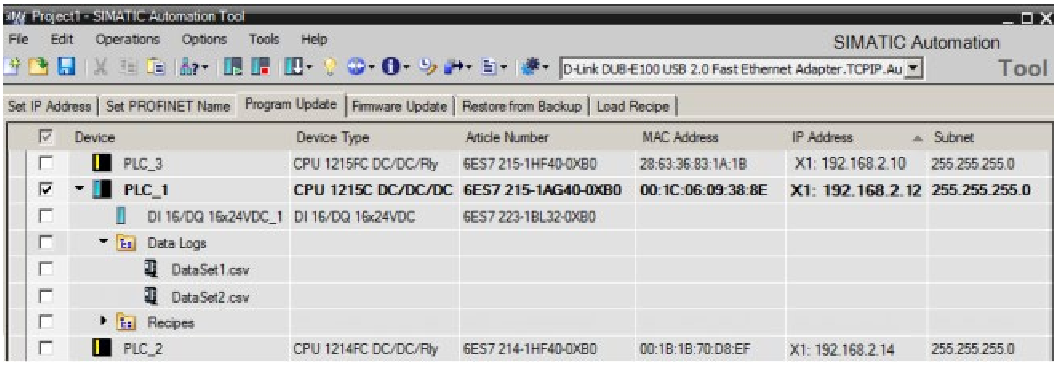
The Backup file password cell has no icon when the field is disabled or empty. When you select a backup file from the list, or enter a CPU password in the backup file password column, the SIMATIC Automation Tool displays the question mark icon. The SIMATIC Automation Tool cannot validate the password at the time you select a file or enter a password.

4.2 Working with the Device table and Event Log

The SIMATIC Automation Tool consists of two parts:

- Device table: The device table is initially empty but after a scan of the network shows the connected devices.
- Event Log: The Event Log shows the results of operations.

For each device, the SIMATIC Automation Tool displays columns with data about the device. Tabs in the device table support a variety of device operations and provide data entry fields.



☑	Device	Device Type	Article Number	MAC Address	IP Address	Subnet
□	PLC_3	CPU 1215FC DC/DC/Rly	6ES7 215-1HF40-0XB0	28:63:36:83:1A:1B	X1: 192.168.2.10	255.255.255.0
☑	**PLC_1**	**CPU 1215C DC/DC/DC**	**6ES7 215-1AG40-0XB0**	**00:1C:06:09:38:8E**	**X1: 192.168.2.12**	**255.255.255.0**
□	DI 16/DQ 16x24VDC_1	DI 16/DQ 16x24VDC	6ES7 223-1BL32-0XB0			
□	Data Logs					
□	DataSet1.csv					
□	DataSet2.csv					
□	Recipes					
□	PLC_2	CPU 1214FC DC/DC/Rly	6ES7 214-1HF40-0XB0	00:1B:1B:70:D8:EF	X1: 192.168.2.14	255.255.255.0

Working with the device table

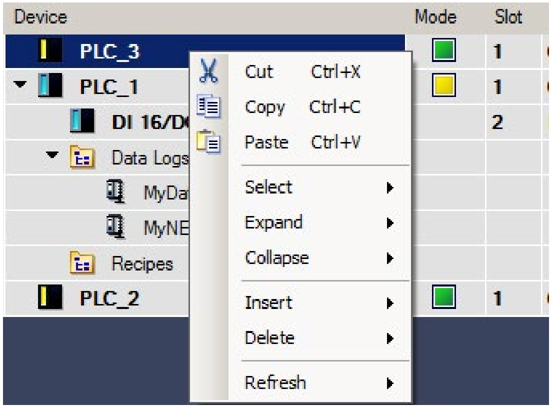
The device table is similar to Microsoft Excel and supports copy and paste operations to or from other applications.

The following tips can help you use the device table:

- Click a column header to sort or reverse sort the rows by that column's data.
- Right-click a column header to show/hide any column.
- Select the Device check box at the top of the device table to select or deselect all devices. Alternatively, you can use the "Edit > Select" menu command to choose either "Select Row(s)" or "Deselect Row(s)". You can also right-click a device row to access the Edit menu.
- Click the upper left corner of the Device table to highlight all rows.
- Select consecutive rows: You can click the cell at the left of a row's check box and drag the cursor up/down to highlight multiple rows.
- When you right-click a row or selection of rows, the shortcut menu shown below lets you use the Select, Expand, Collapse, Insert, Delete, and Refresh commands with a group of rows.
- You can create row filters for the Device, Device Type, and Article Number columns.
- Select one or more devices for operations to perform on multiple devices. The SIMATIC Automation Tool displays selected device rows in bold text.
- You can also export (Page 90) a device table to a .csv file or import (Page 90) a .csv file to the device table.

You can scan the network (Page 46) to fill the device table with devices on your network. You can also insert devices (Page 51) directly.

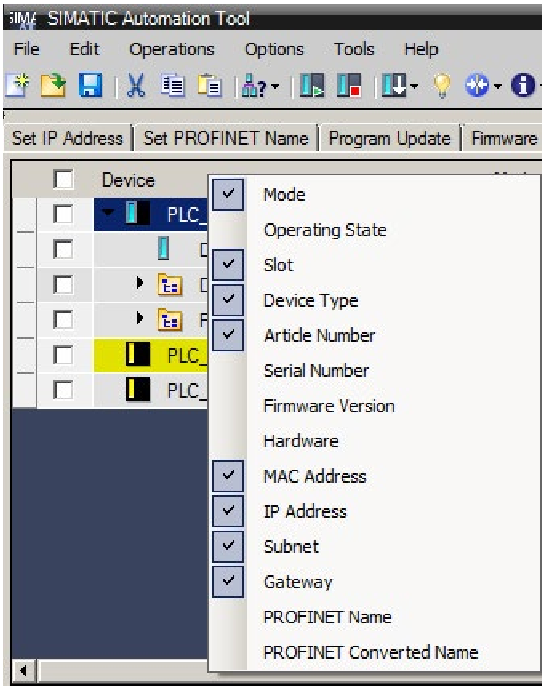
Shortcut menu for table cells



When you right-click a device row in the device table, the shortcut menu is the same as the Edit menu (Page 94).

Right-click menu for column headers

For each tab of the device table, the SIMATIC Automation Tools displays a set of columns by default. You can configure which columns you want to show and which columns you want to hide.



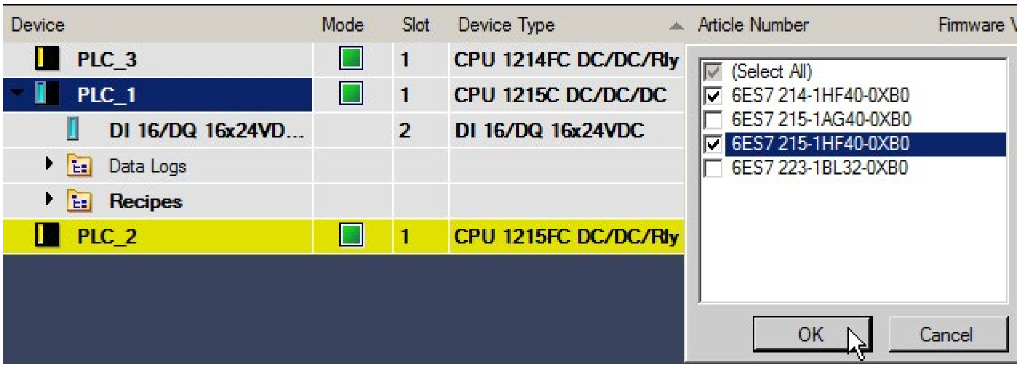
To set the columns to show or hide, follow these steps:

1. Right-click the Device header row to display the shortcut menu for the columns.
2. Select the check boxes to show or hide columns for that tab display.

Filtering the displayed rows

You can filter the Device, Device Type, and Article Number columns. Click one of these three column headers and the filter expand button ▾ appears in the column header. Click this button to open the filter window.

For example, you can select article numbers 6ES7 214-1HF40-0XB0 and 6ES7 215-1HF40-0XB0. When you click the OK button, the device table only displays rows that have these article number values.



Device	Mode	Slot	Device Type
PLC_3	🟩	1	CPU 1214FC DC/DC/Rly
PLC_1	🟩	1	CPU 1215C DC/DC/DC
DI 16/DQ 16x24VD...		2	DI 16/DQ 16x24VDC
Data Logs			
Recipes			
PLC_2	🟩	1	CPU 1215FC DC/DC/Rly

Article Number

- ☑ (Select All)
- ☑ 6ES7 214-1HF40-0XB0
- □ 6ES7 215-1AG40-0XB0
- ☑ 6ES7 215-1HF40-0XB0
- □ 6ES7 223-1BL32-0XB0

Filtering unsupported devices

You can use the General settings (Page 98) to enable/disable the display of unsupported devices.

The SIMATIC Automation Tool displays unsupported devices in gray. You can perform only the following operations on unsupported devices:

- Set IP address
- Set PROFINET name
- Identify device
- Edit the comment for the device
- Perform editing functions such as copy and paste

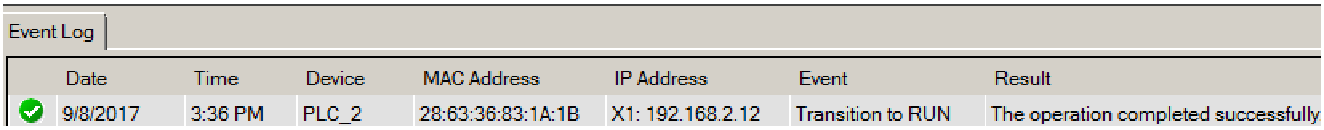
Working with the Event Log

The SIMATIC Automation tool displays the Event Log in the window area below the device table. When you select devices and perform operations, the messages in the Event Log show status information about operational results.

By default, any new device operation clears the Event Log at the start of the operation. You can select whether or not to clear the Event Log at the beginning of an operation in the Event Log settings (Page 105).

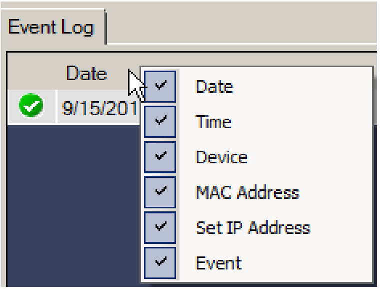
The SIMATIC Automation Tool can also automatically log operation status to a file.

Event Log row showing successful operation:

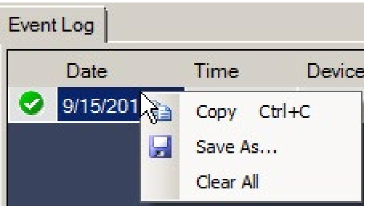


	Date	Time	Device	MAC Address	IP Address	Event	Result
✓	9/8/2017	3:36 PM	PLC_2	28:63:36:83:1A:1B	X1: 192.168.2.12	Transition to RUN	The operation completed successfully

Right-click an Event Log column header to show/hide columns:



Right-click on an event row to open the menu below:



The icons in the Event Log have the following meanings:

✓	Operation is successful
✗	Operation has failed. The Result column describes the reason for failure. If you save the Event Log, these entries begin with "ERROR:".
!	Operation is successful but includes a warning message. The Result column describes the warning information. If you save the Event Log, these entries begin with "WARNING:".

Note

Event Log and user interface language change

When you change the SIMATIC Automation Tool user interface language, the SIMATIC Automation Tool clears the Event Log. Information about previous events is deleted.

4.3 Multi-thread processing options

Multi-thread processing

If you are managing a network with many devices, the SIMATIC Automation Tool can simplify operations and save time by automatically processing a group of devices with multiple processing threads. While a PG/PC communication processing thread is waiting for a SIMATIC device's task complete message, other threads can use this time to communicate with other devices in the group.

For versions 3.0 or later, you must have a SIMATIC Automation Tool software license installed before you can use more than one processing thread.

Devices in a star topology network

If your network has a star topology where each device has a direct connection to the PG/PC through an Ethernet switch, then you can safely enable the multiple threads option.

Devices in a chain topology network

If your network has a chain topology, you should disable the multi-thread option to prevent one device from disrupting the communication to other devices, as described in the Communications settings (Page 99) topic.

Multi-thread processing options

On the Options>Settings>Communications dialog, you can change the following settings:

- Disable or enable multiple threads when processing these operations: Update firmware, Reset to factory defaults, Memory reset, Restore data from backup file, Reset CPU memory, Format SIMATIC memory card, and Reset devices to factory default values operations.
- Set the maximum number of threads allowed (one to five threads).
- Set the timeout which is the maximum time a communications thread waits for a response (180 to 999 seconds).

Table of restrictions to multi-thread processing

✓ Multi-thread processing always used

O Multi-thread processing possible, if enabled in the Communications settings (Page 99).

X Fail-Safe device safety-relevant operation: Only single-thread processing is possible.

SIMATIC Automation Tool operation	Standard device multi-threading	Fail-Safe device multi-threading
Scan network	✓	✓
Identify devices	✓	✓
Update device addresses	✓	✓
Update PROFINET name of a device	✓	✓
Set time in CPU to time in PG/PC	✓	✓
Program update for CPU and HMI devices	✓	X
Upload, add, replace, or delete Recipe data from a CPU	✓	✓
Upload or delete Data Log data from a CPU	✓	✓
Update the firmware in a device	O	O
Put a CPU in RUN or STOP mode	✓	✓
Backup data to a backup file, for CPU or HMI device	✓	✓
Restore from backup file, for CPU or HMI device	O	X
Show the diagnostic buffer of a CPU	✓	✓
Reset CPU memory	O	O
Retrieve Service Data from a CPU	✓	✓
Format memory card (SIMATIC memory card inserted in a CPU)	O	X
Reset to factory defaults	O	X

Processing queues

You can select a group of standard and Fail-Safe devices in the Device table rows and then initiate group processing so the SIMATIC Automation Tool performs the same operation on all devices in the group. SIMATIC Automation Tool uses two processing queues, a priority one queue for Fail-Safe devices to process safety-relevant operations (single thread processing only) and a priority two queue that uses from 1 to 5 threads for standard devices and Fail-Safe devices that allow multi-thread processing.

The SIMATIC Automation Tool processes the Fail-Safe safety-relevant operation device queue first using a single processing thread. The process performs each operation on one Fail-Safe device at a time.

After the SIMATIC Automation Tool completes the Fail-Safe device safety-relevant operation queue, it processes the second queue by multi-thread processing. Some operations always use multi-threading. Others use multi-threading if enabled in the Communications settings (Page 99).

Thread processing status

When you start a group operation, the SIMATIC Automation Tool displays a progress message that shows how many devices in each queue it has completed and the processing progress (% completion) for the active processing threads.

4.4 Scan a network

The device table for a new SIMATIC Automation Tool project is empty. To begin work with the SIMATIC Automation Tool, you scan the communications network to fill the device table. You can also manually insert a device (Page 51).

Scan your network

To scan the network, select the "Operations>Scan Network>Scan Entire Network" menu command. Alternatively, you can click the Scan button on the toolbar and select "Scan Entire Network" from the button drop-down menu.

Note

SIMATIC Automation Tool is an offline tool

Note that the SIMATIC Automation Tool does not update device data continuously. The SIMATIC Automation Tool displays device information at the point of time that you scanned or refreshed your communications network or at the point of time when you inserted devices. The TIA Portal or Web server, for example, could change device data since your last scan. Before performing device operations, scan the network or refresh the devices for which you want to perform device operations.

Device table conventions

Row icons help you identify the device table rows:

Device is unknown or not fully supported. The row's address text is grayed.

PROFINET device

PROFINET Fail-Safe device

PROFINET HMI device

PROFINET Fail-Safe HMI device

Folder containing PROFINET master devices

Folder containing PROFIBUS master devices

Folder containing PROFINET AS-i master devices

Folder containing Data Log or Recipe data

Data Log data

Recipe data

192.168.2.22 Duplicate IP addresses and PROFINET station names appear in red text.

Standard device identity problem

Fail-Safe device identity problem

Note

TIA Portal online connections to devices

If a device has an online connection in the TIA Portal, the SIMATIC Automation Tool cannot read information from the device. The device table displays the icon for unknown device and reports one of the following messages in the Event Log:

- SIMATIC Automation Tool does not support this device.
- Could not establish a connection to the device.

To be able to read the device, go offline in the TIA Portal.

Click the check box next to a device to select it. The SIMATIC Automation Tool displays device text in black for devices you have not selected and in **bold black** when you have selected them.

You can enter text in cells with a light gray background. You cannot enter text in cells with a dark gray background. A dark gray cell indicates that the SIMATIC Automation Tool does not support the operation for that device type/firmware version.

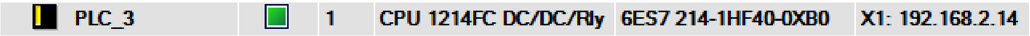
Understanding the device table

> **Note**
>
> **Scan your network again to resolve device identity problems**
>
> A device identity problem can occur, for example, if the TIA Portal modifies the device program or configuration since the last time the SIMATIC Automation Tool completed a network scan. A device with an identity problem continues to fail on all operations and on a refresh command. You must scan the network to resolve the device identity problem.

Fail-Safe devices

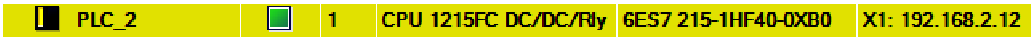
When you change a device's identity or safety program status from the SIMATIC Automation Tool, the tool makes the changes without a new network scan. For example, if you download a new firmware version from the SIMATIC Automation Tool, the SIMATIC Automation Tool updates the device variables to the new values. The SIMATIC Automation Tool is an approved tool for operating on safety devices and is able to handle safety state changes.

If you have a Fail-Safe CPU in your network, but you have not downloaded a safety program (Page 110) to it, the device row cells appear in the color gray.



PLC_3	🟩	1	CPU 1214FC DC/DC/Rly	6ES7 214-1HF40-0XB0	X1: 192.168.2.14

If you have downloaded a safety program to a Fail-Safe CPU, the information fields for the device appear in yellow.

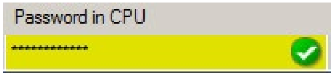


PLC_2	🟩	1	CPU 1215FC DC/DC/Rly	6ES7 215-1HF40-0XB0	X1: 192.168.2.12

The following user-entry fields for Fail-Safe CPUs that have a safety program initially appear in gray. The SIMATIC Automation Tool displays these user-entry fields in yellow after you enter valid values.

- Password in CPU
- Program Update Folder
- Password in Program File
- Backup File

Example:



Password in CPU

************* ✓

Devices connected through CPUs and IP address routers

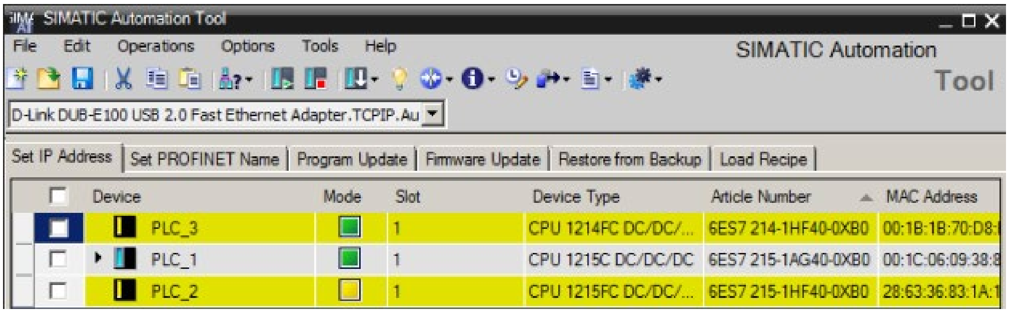
After you enter valid IP addresses in the device table on the IP address tab, you can use the "Operations>Update>IP Address" menu command to transfer the address assignments into selected directly connected devices. You can also click the Update button and choose the "IP Address" command from the button drop-down menu

When supported network devices have valid IP addresses, a network scan shows devices located behind CPUs and IP address routers.

Directly connected devices (including connection through an Ethernet switch)

A directly connected device can use all MAC address operations (with IP address unconfigured or configured) and all IP addressed operations (with IP address configured).

Example initial scan result:



□	Device	Mode	Slot	Device Type	Article Number	MAC Address
□	PLC_3	🟩	1	CPU 1214FC DC/DC/...	6ES7 214-1HF40-0XB0	00:1B:1B:70:D8:
□	PLC_1	🟩	1	CPU 1215C DC/DC/DC	6ES7 215-1AG40-0XB0	00:1C:06:09:38:8
□	PLC_2	🟨	1	CPU 1215FC DC/DC/...	6ES7 215-1HF40-0XB0	28:63:36:83:1A:1

Scan rules for existing table entries

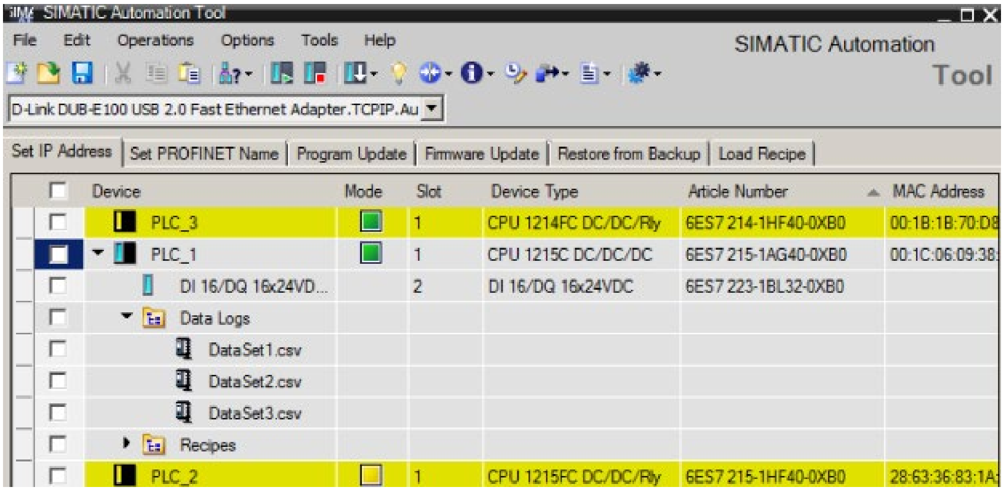
- If a MAC address already exists in the table, then scanning the network updates the IP address, Subnet, and Gateway fields for that device table row. The data in all other fields remains the same.
- If a MAC address is new, then the SIMATIC Automation Tool creates a new row with the MAC address, IP Address, Subnet, and Gateway. All other fields are empty.

PROFINET I/O

PROFINET I/O devices can appear twice in the device table. The device is shown once on a top level row, where direct connection with the tool allows all supported SIMATIC Automation Tool operations. The device is also shown in a lower level row behind a CPU (with valid IP address and hardware configuration), where an indirect tool connection restricts the device row to firmware update only. The two device table rows result from the two different connection paths that are possible on the Ethernet network.

Expand the device rows and show local modules, decentralized I/O devices, HMI panels, and CPU files (Recipes and Data Logs).

Click the expand icon ▸ to expand a device row. Use the right-click shortcut menu or Edit menu to expand/collapse all levels.



□	Device	Mode	Slot	Device Type	Article Number	MAC Address
□	PLC_3	🟩	1	CPU 1214FC DC/DC/Rly	6ES7 214-1HF40-0XB0	00:1B:1B:70:D8
□	PLC_1	🟩	1	CPU 1215C DC/DC/DC	6ES7 215-1AG40-0XB0	00:1C:06:09:38:
□	DI 16/DQ 16x24VD...		2	DI 16/DQ 16x24VDC	6ES7 223-1BL32-0XB0	
□	Data Logs					
□	DataSet1.csv					
□	DataSet2.csv					
□	DataSet3.csv					
□	Recipes					
□	PLC_2	🟨	1	CPU 1215FC DC/DC/Rly	6ES7 215-1HF40-0XB0	28:63:36:83:1A:

Only the firmware update operation is possible for indirectly connected devices.

Devices on the lower levels represent devices and CPU data files that are indirectly connected to the SIMATIC Automation Tool through a directly connected CPU. A valid IP address and hardware configuration is necessary in a CPU before devices connecting through that CPU are visible in the device table.

Devices on the third and fourth levels can represent decentralized I/O devices (PROFINET and PROFIBUS devices). An IP configuration is necessary in a level two decentralized I/O controller, before the decentralized I/O devices (for example, head module and I/O modules) are visible in the device table.

Password identification

If the device is password-protected (Page 37) at any protection level, then the SIMATIC Automation Tool enables the password field.

Refreshing device table data

The SIMATIC Automation Tool refreshes device table row data in the following situations:

- On a network scan
- After an operation completes
- When you refresh one or more devices

To refresh device table data, choose one of these methods:

- Select "Scan Entire Network" from either the toolbar button drop down menu or the "Operations > Scan Network" menu to refresh all devices.
- Select devices and select "Refresh Status of All Selected Devices" from either the toolbar button drop down menu or the "Operations > Scan Network" menu.
- Select the Edit>Refresh menu command or right-click a device row and select Refresh from the shortcut menu. Then choose one of the following options from the Refresh menu command:
  - Device
  - All Selected Devices F5
  - All Devices
- Press the F5 key to refresh "All Selected Devices"

The SIMATIC Automation Tool refreshes the device data that it reads from the devices and retains all user-entered data fields.

If you refresh devices that are no longer present on the network, the SIMATIC Automation Tool displays the device row data in italics.

See also

Event Log settings (Page 105)

General settings (Page 98)

4.5 Inserting a device

You can insert a device into the device table. You can only add a device that has a unique MAC address and unique IP address from any other devices in the device table.

To insert a device, follow these steps:

1. Select the "Insert > Device" menu command from either the Edit menu or the device table right-click shortcut menu.
2. From the "Insert Device" dialog, enter either an IP address or MAC address for the device. The address you enter must not correspond to the address for an existing device.

The SIMATIC Automation Tool rejects an attempt to insert a device that does not have a unique address and generates an event log message.

When you enter a unique IP address or unique MAC address, the SIMATIC Automation Tool attempts to communicate with the address you provided. If communication is successful, the SIMATIC Automation Tool inserts the device into the device table. If communication is not successful, the SIMATIC Automation Tool informs you that the device does not exist on the network.

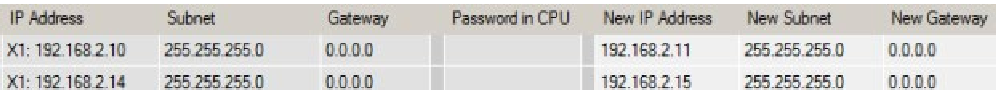
If the device is behind a router, the SIMATIC Automation Tool displays the device name in blue.

4.6 Update IP, subnet, and gateway addresses

Change IP addresses

To update the IP address for a device, follow these steps:

1. Click the "Set IP Address" tab.
2. Select one or more devices to include in the operation. You can use the Devices check box at the top of the device table to select or deselect all devices. Alternatively, you can use the right-click shortcut menu or the "Edit > Select" menu command to access the "Select All" and "Deselect All" commands.
3. Enter address changes in the "New IP Address", "New Subnet", and "New Gateway" columns. Note that the device table shows the communication interface, such as "X1". You do not, however, enter the communication interface when you enter the New IP Address. If you enter invalid syntax, the SIMATIC Automation Tool displays the field in red text.



IP Address	Subnet	Gateway	Password in CPU	New IP Address	New Subnet	New Gateway
X1: 192.168.2.10	255.255.255.0	0.0.0.0		192.168.2.11	255.255.255.0	0.0.0.0
X1: 192.168.2.14	255.255.255.0	0.0.0.0		192.168.2.15	255.255.255.0	0.0.0.0

4. Select Update from the Operations menu or click the Update button on the toolbar and select "Set IP address" from the button drop-down menu.

The Update operation sets the IP, subnet, and gateway addresses in the selected devices.

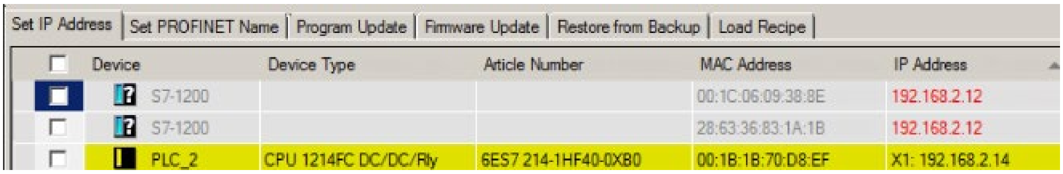
The Event log below the device table shows the results of this operation.

Duplicate IP addresses

When two or more devices have the same IP address, the addresses appear in red text as shown in the following image. You can select devices with duplicate IP addresses, update the IP addresses and correct the network problem. Only the following operations on devices with duplicate IP addresses are possible:

- Delete
- Set IP address
- Set PROFINET name
- Identify devices

No other tool operations are possible for devices that have duplicate IP addresses.



Set IP Address | Set PROFINET Name | Program Update | Firmware Update | Restore from Backup | Load Recipe

□	Device	Device Type	Article Number	MAC Address	IP Address
□	S7-1200			00:1C:06:09:38:8E	192.168.2.12
□	S7-1200			28:63:36:83:1A:1B	192.168.2.12
□	PLC_2	CPU 1214FC DC/DC/Rly	6ES7 214-1HF40-0XB0	00:1B:1B:70:D8:EF	X1: 192.168.2.14

Setting the IP address on unsupported devices

MAC address based operations use the DCP (Discovery and Configuration Protocol). DCP is an Ethernet standard. The SIMATIC Automation Tool can use the DCP-MAC address operations Scan Entire Network, Identify, Update IP address, and Update PROFINET name for all directly connected network devices (CPUs, HMIs, decentralized I/O, and other devices).

Select the unsupported device row, enter new data in the appropriate column, and update the unsupported device IP address, in the same way that you update supported devices.

Unsupported devices might not accept a change based on the hardware configuration of the device.

By default, the SIMATIC Automation Tool displays unsupported devices. You can disable this option in the General settings (Page 98)

Note

Some of the information that SIMATIC Automation Tool displays cannot be gathered from devices that are connected behind a IP router.

For example, the Default Gateway address is gathered from devices using DCP.
DCP is not a routable protocol and therefore information cannot be read from devices connected behind a router. In this situation, the related Device table fields are empty.

4.7 Update PROFINET device names

PROFINET name rules

Valid names follow the standard DNS (Domain Name System) naming conventions.

The maximum number of characters for the device name is 63. Valid characters are the lower case letters "a" through "z", the digits 0 through 9, the hyphen character (minus sign), and the period character.

Invalid names

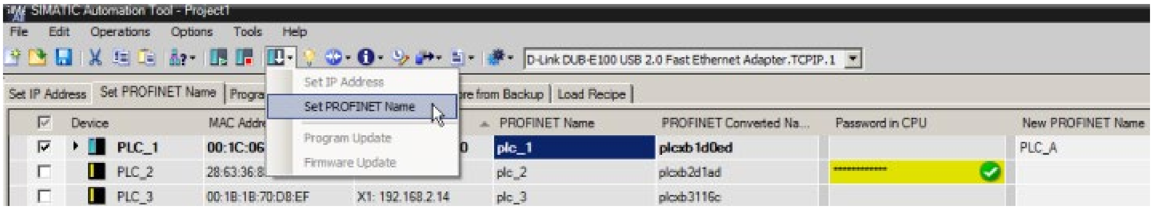
- The name must not have the format n.n.n.n where n is a value of 0 through 999.
- You cannot begin the name with the string port-nnn or the string port-nnnnnnnn, where n is a digit 0 through 9. For example, "port-123" and "port-123-45678" are illegal names.
- A name cannot start or end with a hyphen "-" or period "." character.

Change PROFINET name

Click the "Set PROFINET Name" tab.

Select one or more devices to include in the operation. You can use the Devices check box at the top of the device table to select or deselect all devices. Alternatively, you can use the right-click shortcut menu or the "Edit > Select" menu command to access the "Select All" and "Deselect All" commands.

1. Enter a new PROFINET name in the "New PROFINET Name" column.
2. Select Update from the Operations menu or click the Update button on the toolbar and select "PROFINET Name" from the button drop-down menu.



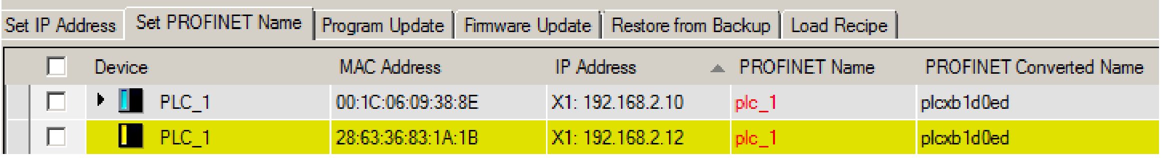
☑	Device	MAC Addr		PROFINET Name	PROFINET Converted Na...	Password in CPU	New PROFINET Name
☑	PLC_1	00:1C:06	0	plc_1	plcxb1d0ed		PLC_A
☐	PLC_2	28:63:36:8		plc_2	plcxb2d1ad	************ ✓	
☐	PLC_3	00:1B:1B:70:D8:EF	X1: 192.168.2.14	plc_3	plcxb3116c		

The Update operation sets new PROFINET names in the selected devices. If you enter an invalid PROFINET name according to the PROFINET name rules, the SIMATIC Automation Tool corrects the name to a valid name. The column "PROFINET Converted Name" shows the converted name.

The Event log below the device table shows the results of this operation.

Duplicate PROFINET names

When two or more devices have duplicate PROFINET names, the SIMATIC Automation Tool indicates the duplicates with red text. The SIMATIC Automation Tool supports full functionality for these devices and displays all other information.



☐	Device	MAC Address	IP Address	PROFINET Name	PROFINET Converted Name
☐	PLC_1	00:1C:06:09:38:8E	X1: 192.168.2.10	plc_1	plcxb1d0ed
☐	PLC_1	28:63:36:83:1A:1B	X1: 192.168.2.12	plc_1	plcxb1d0ed

Setting PROFINET name on unsupported devices

MAC address based operations use the DCP (Discovery and Configuration Protocol). DCP is an Ethernet standard. The SIMATIC Automation Tool can use the DCP-MAC address operations Scan Entire Network, Identify, Update IP address, and Update PROFINET name for all directly connected network devices (CPUs, HMIs, decentralized I/O, and other devices).

Select the unsupported device row, enter new data in the appropriate column, and update unsupported device PROFINET names, in the same way that you update supported devices.

Unsupported devices might not accept a change based on the hardware configuration of the device.

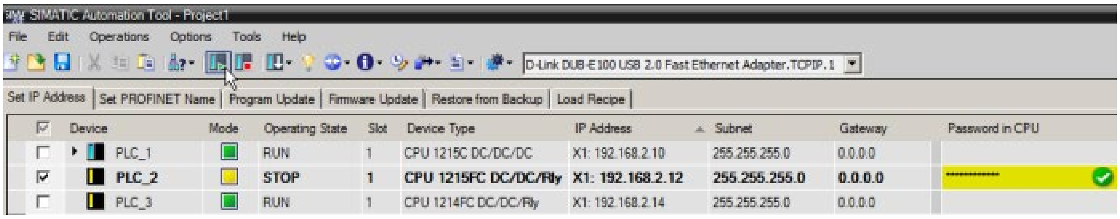
By default, the SIMATIC Automation Tool displays unsupported devices. You can disable this option in the General settings (Page 98)

4.8 Set CPUs to RUN or STOP mode

Change CPUs to RUN or STOP mode

To change the operating mode for a device, follow these steps:

1. Select one or more devices to include in the operation. You can use the Devices check box at the top of the device table to select or deselect all devices. Alternatively, you can use the right-click shortcut menu or the "Edit > Select" menu command to access the "Select All" and "Deselect All" commands.
2. For each selected CPU, enter a password, if used, in the "Password in CPU" column of the currently open tab.
3. Set the operating mode to either RUN mode or STOP mode:
   - Select RUN from the Operations menu or click the RUN toolbar button. A valid program must exist in the CPU before it can enter RUN mode.
   - Select STOP from the Operations menu or click the STOP toolbar button.



☑	Device	Mode	Operating State	Slot	Device Type	IP Address	Subnet	Gateway	Password in CPU
☐	PLC_1	🟩	RUN	1	CPU 1215C DC/DC/DC	X1: 192.168.2.10	255.255.255.0	0.0.0.0	
☑	**PLC_2**	🟨	**STOP**	**1**	**CPU 1215FC DC/DC/Rly**	**X1: 192.168.2.12**	**255.255.255.0**	**0.0.0.0**	************* ✓
☐	PLC_3	🟩	RUN	1	CPU 1214FC DC/DC/Rly	X1: 192.168.2.14	255.255.255.0	0.0.0.0	

The SIMATIC Automation Tool sets the selected CPUs to RUN or STOP mode.

The Mode and Operating state columns in the device table indicate the current CPU state. Yellow means STOP mode. Green means RUN mode. RED means CPU fault.

The Event log below the device table shows the results of the operation.

4.9 Identify devices

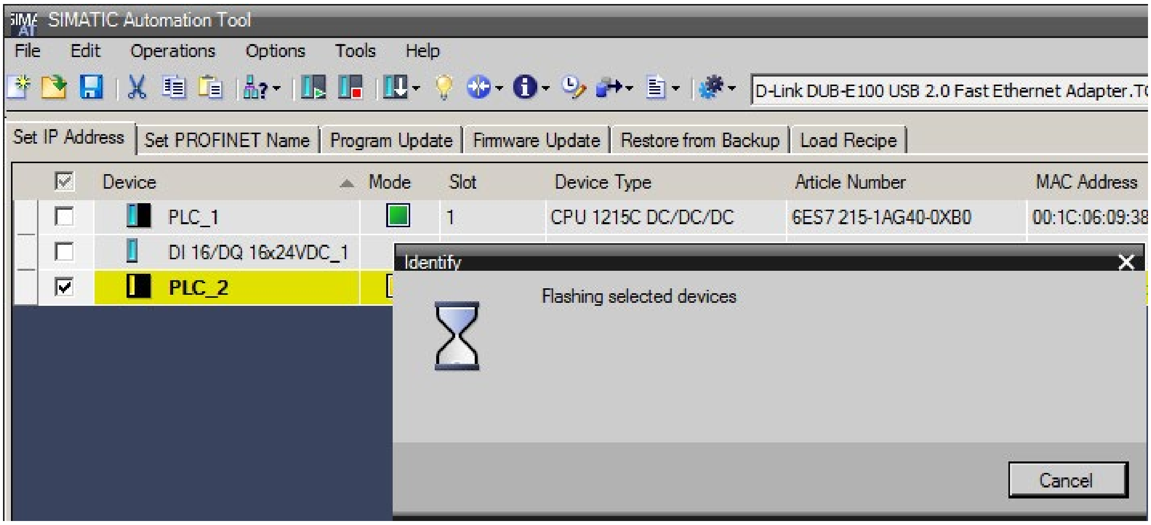
Locate a device by flashing an LED or HMI display

The Identify operation helps you physically locate devices in the device table. You can use the Identify operation in RUN mode and STOP mode. To identify devices, follow these steps:

1. Select one or more devices to include in the operation. You can use the Devices check box at the top of the device table to select or deselect all devices. Alternatively, you can use the right-click shortcut menu or the "Edit > Select" menu command to access the "Select All" and "Deselect All" commands.
2. Select the "Operations > Identify" menu command or click the toolbar button 💡 for identifying selected devices.

Selected CPU devices flash their LEDs to show their location. Selected HMI devices perform a screen flash.

Flashing continues until you click the cancel button.



☑	Device	Mode	Slot	Device Type	Article Number	MAC Address
□	PLC_1	🟩	1	CPU 1215C DC/DC/DC	6ES7 215-1AG40-0XB0	00:1C:06:09:38
□	DI 16/DQ 16x24VDC_1					
☑	PLC_2					

Identify

Flashing selected devices

Identifying unsupported devices (flashing LED / HMI screen)

MAC address based operations use the DCP (Discovery and Configuration Protocol). DCP is an Ethernet standard. The SIMATIC Automation Tool can use the DCP-MAC address operations Scan Entire Network, Identify, Update IP address, and Update PROFINET name for all directly connected network devices (supported PROFINET devices, unsupported PROFINET devices, and unsupported non-Siemens Ethernet devices).

Select the unsupported device row, and identify an unsupported device, in the same way that you identify supported devices.

Unsupported devices might not flash LEDs based on the hardware configuration of the device.

By default, the SIMATIC Automation Tool displays unsupported devices. You can disable this option in the General settings (Page 98)

4.10 Update device programs from the SIMATIC Automation Tool

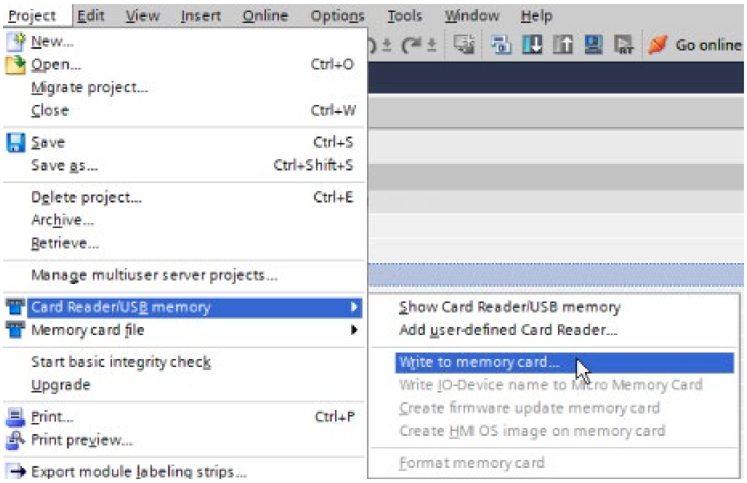
Before you can transfer a program to a CPU using the SIMATIC Automation Tool, you must have access to the program on one of the following forms of media:

- SIMATIC memory card
- USB flash drive
- hard drive of your PG/PC

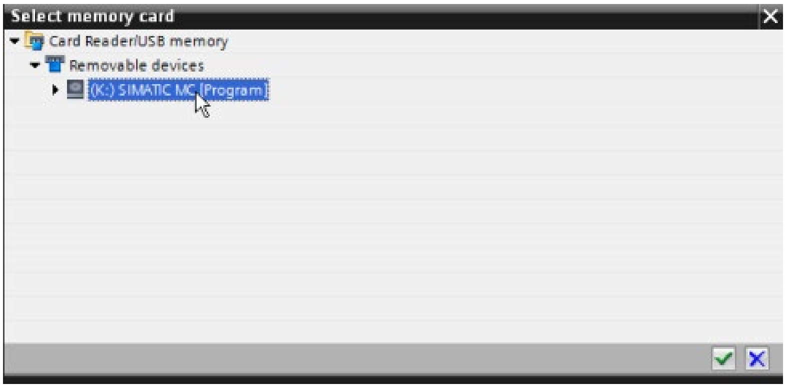
Preparing a CPU program for use with the SIMATIC Automation Tool

To transfer a TIA Portal CPU project to a SIMATIC memory card, follow these steps:

1. Insert a SIMATIC memory card into the card reader for your PG/PC
2. From STEP 7, select the CPU in the Project tree
3. Select the "Project > Card Reader/USB memory > Write to memory card" menu command.



4. Select your memory card from the dialog.



STEP 7 saves a SIMATIC.S7S folder on your SIMATIC memory card that contains your CPU project. You can also copy the STEP 7 project to the memory card by dragging the project to the memory card in the project tree.

Refer to the STEP 7 Information System (online help) for additional information.

After the TIA portal transfers program data to a storage device, you can use Windows file Explorer to transfer the program to the folder that is used by the SIMATIC Automation Tool.

Copy the "SIMATIC.S7S" folder for each CPU program

Follow these steps to make a CPU program accessible to the SIMATIC Automation Tool

1. Create subfolders under the Program Update folder (Page 102). Create one folder for each program and create a folder name that identifies the program. The folder names that you create will appear in the SIMATIC Automation Tool program drop-down list.
2. Use Windows Explorer to copy the "SIMATIC.S7S" folder (including all subfolders and files) to each subfolder for each program. You can put a TIA portal program (a "SIMATIC.S7S" folder) in a zip file archive and extract it to your subfolder location. Note that you update recipes in a separate recipe operation (Page 64).

See the "Example CPU program update" section later in this topic.

Note

TIA portal program data

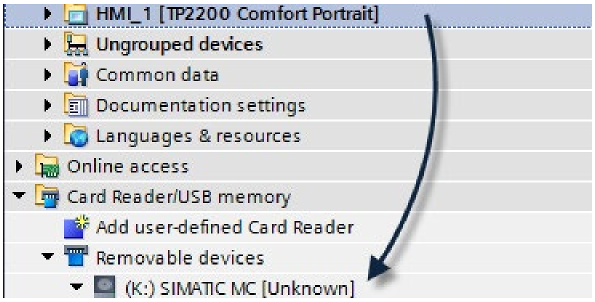
The program data is protected. You cannot discover details like the project name or target CPU of a TIA portal program from the data that is stored in a SIMATIC.S7S folder. You cannot identify one program's SIMATIC.S7S folder from another program's SIMATIC.S7S folder.
You must create and name subfolders under the SIMATIC Automation Tool program update folder (Page 102) that identify a program's function or target CPU. Copy a program's SIMATIC.S7S folder into the subfolder that you named. The subfolder names that you create appear in the SIMATIC Automation Tool "Program" column drop-down list and provide the path to the correct SIMATIC.S7S folder.

Preparing an HMI operating system and runtime software for use with the SIMATIC Automation Tool

HMI devices from Version 14 and higher support saving the operating system and runtime from STEP 7.

To copy the operating system and runtime files for an HMI to a SIMATIC memory card, follow these steps:

1. Insert a SIMATIC memory card into the card reader for your PG/PC.
2. Expand "Card Reader/USB memory" in the Project tree to show the drive corresponding to your card reader.
3. Select your HMI in the Project tree and drag it to the drive letter of your card reader.



STEP 7 saves a SIMATIC.HMI folder on your SIMATIC memory card that contains your HMI runtime and HMI operating system. HMI updates include the operating system and runtime data. You do not have the option to select a partial update.

After the TIA portal transfers the SIMATIC.HMI folder to a storage device, use the Windows file explorer to make the SIMATIC.HMI folder accessible to the SIMATIC Automation Tool:

- Create a subfolder for the HMI program in the Program Update (Page 102) folder.
- Copy the SIMATIC.HMI folder to the subfolder.

Update CPU programs or HMI operating system and runtime software

If you have a chain communication topology and the Communications settings (Page 99) enable multi-threading, be aware of the risk of communication disruption with this operation.

Note

Fail-Safe devices

If the Fail-Safe CPU is protected, you must enter the safety F-CPU password in the "Password in CPU" column to update the program in a Fail-Safe device.

You must confirm an additional prompt for program updates to F-CPUs and reselect your device under the following conditions:

- You are updating a safety program (Page 110) with another safety program
- You are updating a safety program with a standard program
- You are loading a safety program for the first time
- You are updating a standard program that requires the CPU password for access level "Full access incl. fail-safe (no protection)".

The SIMATIC Automation Tool places "Program Update" requests for Fail-Safe devices in the F-CPU safety-relevant operation queue. The SIMATIC Automation Tool uses only single-thread sequential processing for the safety-relevant operation queue.

The destination device for a safety program must be a Fail-Safe CPU.

⚠ **WARNING**

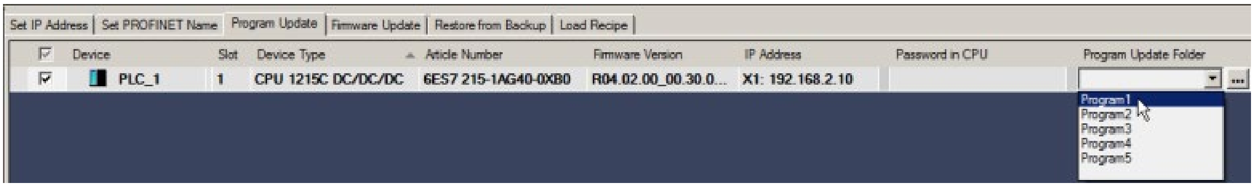
Verify that the device is not actively running a process before updating the program

Installing a new program causes CPUs to go to STOP mode, which could affect the operation of an online process or machine. Unexpected operation of a process or machine could result in death or injury to personnel and/or property damage.

After you have stored programs in the program update folder, you can use the SIMATIC Automation Tool to load new programs in one or more devices. To perform a program update, follow these steps:

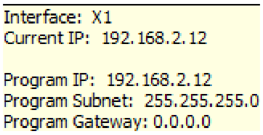
1. Click the "Program Update" tab.
2. Select one or more devices to include in the operation. You can use the Devices check box at the top of the device table to select or deselect all devices. Alternatively, you can use the right-click shortcut menu or the "Edit > Select" menu command to access the "Select All" and "Deselect All" commands.

3. For each selected device, use the "Program Update Folder" column drop-down list to select a folder name. The drop-down list shows the folders that you created in the program update path.



☑	Device	Slot	Device Type	Article Number	Firmware Version	IP Address	Password in CPU	Program Update Folder
☑	PLC_1	1	CPU 1215C DC/DC/DC	6ES7 215-1AG40-0XB0	R04.02.00_00.30.0...	X1: 192.168.2.10		

You can also use the browse button ... and navigate to the folder where you have stored a program on your PG/PC. When you select a program, the SIMATIC Automation Tool adds it to the drop-down list. If the selected file has the same name as one of the files already listed, the SIMATIC Automation Tool adds a number to the new file name to make the names unique. To help you identify files, when you hover over the program name in the Program Update field, a tooltip displays the following information about the device and selected program:

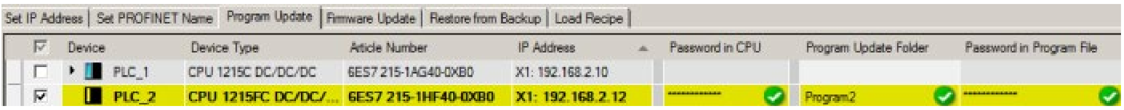


```
Interface: X1
Current IP: 192.168.2.12

Program IP: 192.168.2.12
Program Subnet: 255.255.255.0
Program Gateway: 0.0.0.0
```

If the program file does not contain an IP address, the tooltip displays "Set directly at device" for all IP address fields.

4. Enter passwords, if used, in the "Password in CPU" and "Password in Program File" columns. Program update is a safety-relevant operation. If the device is a Fail-Safe device, you must enter the safety F-CPU password.



☑	Device	Device Type	Article Number	IP Address	Password in CPU	Program Update Folder	Password in Program File
☐	PLC_1	CPU 1215C DC/DC/DC	6ES7 215-1AG40-0XB0	X1: 192.168.2.10			
☑	PLC_2	CPU 1215FC DC/DC/...	6ES7 215-1HF40-0XB0	X1: 192.168.2.12	************ ✓	Program2 ✓	************ ✓

5. Select the "Operations > Update > Program Update" menu command to start the operation. Alternatively select the Update toolbar button and select "Program Update" from the button drop-down menu.

The Event Log below the device table shows the results of this operation.

Program validation

The SIMATIC Automation Tool verifies the program data, before updating the program in a CPU.

If there is an error in the program data, then a red "X" icon is displayed in the "Program Update Folder" cell. Additional error information is available in a tooltip, when you hover over the cell.

Password handling after Program Update operation

A program file might have a password, which might be different from the existing CPU password. When a program file has a password, you must enter the program password in the "Password in Program File" cell to perform a program update. The program password becomes the CPU password after the Program Update operation completes.

After a successful Program Update operation, the SIMATIC Automation Tool automatically copies the Password in Program File to the CPU password field and attempts a connection using the new password. The SIMATIC Automation Tool then clears the Password in Program File field and the Program Update Folder field.

If the password you enter in the "Password in Program File" column is not the password configured for the project in the TIA Portal, then the Event Log shows a warning after the operation completes. In this case the CPU password shows a red ‘X’ icon that indicates an invalid password.

F-signature validation

A TIA Portal project that contains a safety program has an F-signature that is used to verify the data in a copied program and provides an additional level of security for safety programs. After a Program Update operation, the F-Signature in the project is compared to the F-Signature now loaded on the CPU device.

A successful comparison is reported in the Event Log as: "Result of CRC comparison, online and offline collective F-signatures match"

The SIMATIC Automation Tool reports an unsuccessful comparison in the Event Log as:" Result of CRC comparison, online and offline collective F-signatures do not match" In the event of an unsuccessful comparison, reset the device to factory defaults (Page 79) and reattempt the program update.

⚠WARNING

Be sure you load the correct safety program.

Running the wrong program on an F-CPU can affect the operation of a process or machine. Unexpected operation of a process or machine could result in death or injury to personnel and/or property damage.

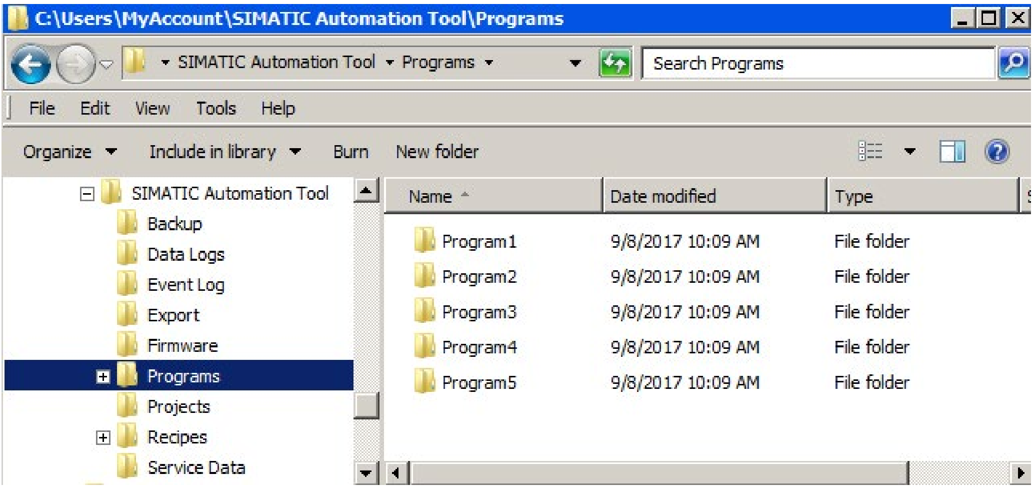
Do not attempt to go to RUN mode if you are not sure that you have loaded the correct safety program.

Example: Program update

If you want five different CPU programs available for Program Update, then you must create and name five folders in the Program Update folder (Page 100). Copy the entire "SIMATIC.S7S" folders to the five corresponding folders.

In this example, the folder names "Program1", "Program2", "Program3", "Program4", and "Program5" identify the available programs. You can use any folder name you want. The folder name could refer to a program function, or CPU location.

The following image shows the Windows Explorer view of the subfolders under the Programs folder. You copy the corresponding SIMATIC.S7S program folders to these folders.

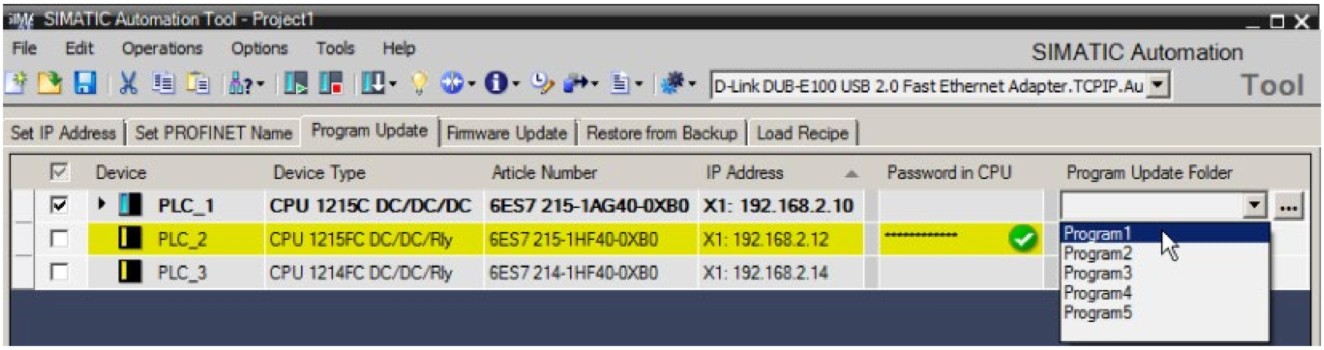


C:\Users\MyAccount\SIMATIC Automation Tool\Programs

Name	Date modified	Type
Program1	9/8/2017 10:09 AM	File folder
Program2	9/8/2017 10:09 AM	File folder
Program3	9/8/2017 10:09 AM	File folder
Program4	9/8/2017 10:09 AM	File folder
Program5	9/8/2017 10:09 AM	File folder

The following image shows the SIMATIC Automation Tool Program Update tab with the example folder names in the "New Program Update" column drop-down list. You must use the drop-down list in the "New Program Update" column to assign which program to use. If you select more than one CPU row, then you must repeat the process and assign the correct program for each CPU that you selected.

Select the "Operations > Update > Program Update" menu command to start the program update. Alternatively, click the toolbar Update button, and select "Program Update" from the button drop-down menu.



SIMATIC Automation Tool - Project1

☑	Device	Device Type	Article Number	IP Address	Password in CPU	Program Update Folder
☑	PLC_1	CPU 1215C DC/DC/DC	6ES7 215-1AG40-0XB0	X1: 192.168.2.10		
□	PLC_2	CPU 1215FC DC/DC/Rly	6ES7 215-1HF40-0XB0	X1: 192.168.2.12	************ ✓	
□	PLC_3	CPU 1214FC DC/DC/Rly	6ES7 214-1HF40-0XB0	X1: 192.168.2.14		

The process is similar for HMI data. The folder name within a project folder is "SIMATIC.HMI" instead of "SIMATIC.S7S". The procedure is the same.

CPU program update rules

The SIMATIC Automation Tool supports the program update operation for standard CPUs and Fail-Safe CPUs.

Program update rules:

- The firmware version of the CPU hardware must be greater than or equal to the firmware version in the project that you want to load.
  You can work around this restriction by updating the firmware in the CPU, if possible.
- For the S7-1200, S7-1500, and ET 200SP (S7-1500) CPUs, the SIMATIC Automation Tool supports the program update operation, if the project's assigned CPU firmware version is supported as shown in the following tables.

Program update support tables

- Program update is possible where ✓ is displayed.
- Program update is not possible where an empty cell is displayed.

S7-1500 **ET 200SP CPU**		**CPU firmware version configured in project for CPU update**						
		1.0	1.1	1.5	1.6	1.7	1.8	2.0
Target CPU firmware version	1.0	✓						
	1.1	✓	✓					
	1.5	✓	✓	✓				
	1.6	✓	✓	✓	✓			
	1.7	✓	✓	✓	✓	✓		
	1.8	✓	✓	✓	✓	✓	✓	
	2.0	✓	✓	✓	✓	✓	✓	✓
	2.1	✓	✓	✓	✓	✓	✓	✓

Use the TIA Portal to change an S7-1200 project's CPU version to a supported version for a successful program update.

S7-1200		**CPU firmware version configured in project for CPU update**						
		2.0	2.1	2.2	3.0	4.0	4.1	4.2
Target CPU firmware version	2.0	✓						
	2.1	✓	✓					
	2.2	✓	✓	✓				
	3.0	✓	✓	✓	✓			
	4.0					✓		
	4.1					✓	✓	
	4.2					✓	✓	✓

4.11 Upload, add, replace, and delete Recipes in CPUs

Recipe operations work for CPUs that have recipes in external load memory (SIMATIC memory card). The CPU can be in either RUN or STOP mode.

Recipe data is formatted as .CSV (comma-separated values) text files.

You can add or replace multiple recipes in a single operation if the recipes are all on different CPUs. You cannot add or replace more than one recipe into one CPU in a single operation.

You can select multiple recipes for upload or deletion in one CPU in a single operation.

When you select a recipe file or recipe folder, the SIMATIC Automation Tool displays the device table row in **bold text**.

For Recipe downloads, click the Recipe tab and select a CPU's Recipe folder row.

The SIMATIC Automation Tool creates a unique folder name for each CPU to store uploaded recipe files on your PG/PC. The folder name includes the CPU name combined with the MAC address. If you select and upload the same recipe file twice, the SIMATIC Automation Tool appends a number to the filename to make all filenames unique.

The SIMATIC Automation Tool must have Read access to upload recipe files. To delete, add, or replace recipe files, the SIMATIC Automation Tool must have Full access (read and write) authorization. You might have to enter a password to successfully perform a delete or add/replace operation. If you do not enter a password, or if the password does not authorize the CPU write access, the operation for that CPU fails and puts an error message in the Event Log.

Toolbar actions

From the drop-down menu of the "File Operations" toolbar button 📄▾, you have the following choices:

Upload Recipes Copies selected recipe files from a CPU to the recipe folder (Page 104) of your PG/PC.

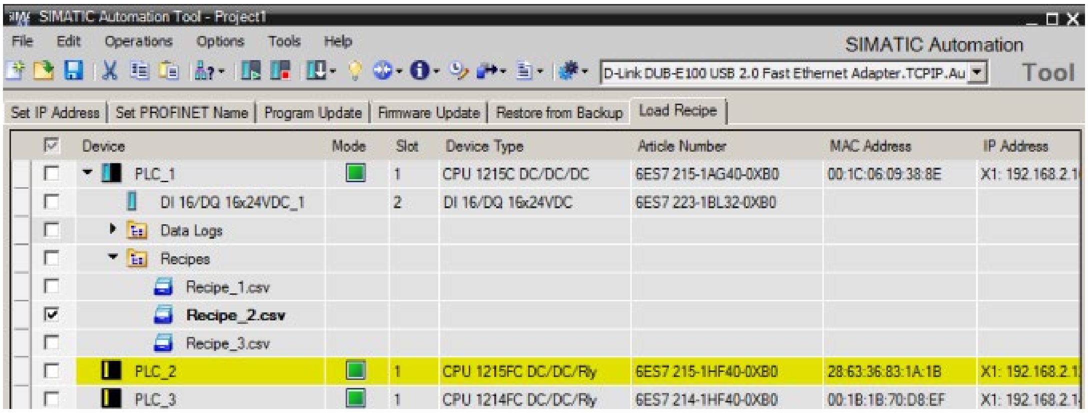
Add/Replace Recipe (available for selection when you are displaying the Load Recipe tab) Copies selected recipe files from your PG/PC to a CPU.

Delete Recipes Deletes selected recipe files from a CPU.

Uploading or deleting recipe files

To upload or delete recipe files from a CPU, follow these steps:

1. Click the "Load Recipe" tab on the Device table.
2. Expand a CPU device and make a recipe folder visible
3. Expand a recipe folder and select the recipe files to include in the operation.
4. For each selected CPU, enter a password, if used, in the "Password in CPU" column.
5. Select the "Operations > File Operations" menu command or click the "File Operations" toolbar button:
6. Select either "Upload Recipes" or "Delete Recipe" from the File Operations menu.



☑	Device	Mode	Slot	Device Type	Article Number	MAC Address	IP Address
□	PLC_1	🟩	1	CPU 1215C DC/DC/DC	6ES7 215-1AG40-0XB0	00:1C:06:09:38:8E	X1: 192.168.2.1
□	DI 16/DQ 16x24VDC_1		2	DI 16/DQ 16x24VDC	6ES7 223-1BL32-0XB0		
□	Data Logs						
□	Recipes						
□	Recipe_1.csv						
☑	**Recipe_2.csv**						
□	Recipe_3.csv						
□	PLC_2	🟩	1	CPU 1215FC DC/DC/Rly	6ES7 215-1HF40-0XB0	28:63:36:83:1A:1B	X1: 192.168.2.1
□	PLC_3	🟩	1	CPU 1214FC DC/DC/Rly	6ES7 214-1HF40-0XB0	00:1B:1B:70:D8:EF	X1: 192.168.2.1

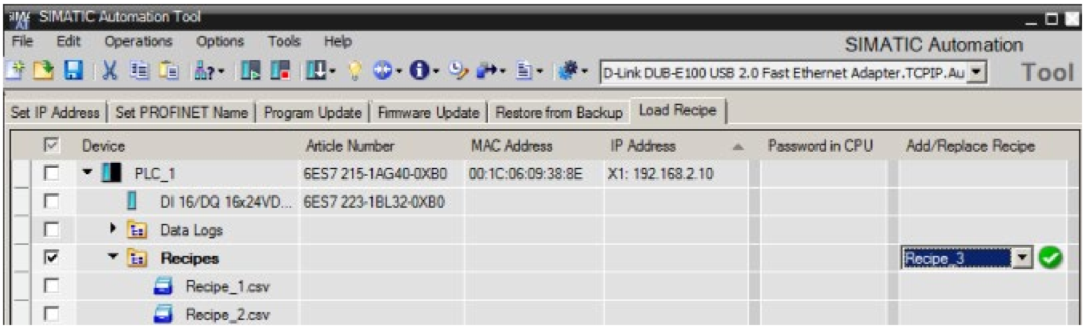
The Event Log below the device table shows the results of the operation.

Adding or replacing (downloading) recipe files

To add or replace CPU recipe files from files on your PG/PC, follow these steps:

1. Copy Recipe data .csv files that you want to add to or replace in a CPU into the Recipes folder. Recipes folder. (Page 104)
2. Click the "Load Recipe" tab in the device table.
3. Expand a CPU device and make a recipe folder visible.
4. Select the recipe folder that you want to load.

5. For each recipe folder that you selected, click the "Add/Replace Recipe" column drop-down list and select a recipe file name. The drop-down list shows the names of .csv files that exist in the directory path assigned in the Recipes section of the "Options > Settings" dialog.
   You can also use the browse button ... and navigate to the folder where you store recipe files on your PG/PC. The SIMATIC Automation Tool adds the file that you browsed to and selected to the drop-down list. If the selected file has the same name as one of the files already listed, the SIMATIC Automation Tool adds a number to the new file name to make the names unique. To help you identify files, a tooltip displays the entire path and filename.



☑	Device	Article Number	MAC Address	IP Address	Password in CPU	Add/Replace Recipe
□	PLC_1	6ES7 215-1AG40-0XB0	00:1C:06:09:38:8E	X1: 192.168.2.10		
□	DI 16/DQ 16x24VD...	6ES7 223-1BL32-0XB0				
□	Data Logs					
☑	Recipes					Recipe_3 ✓
□	Recipe_1.csv					
□	Recipe_2.csv					

6. Select the "Operations > File Operations" menu command or click the "File Operations" toolbar button:
7. Select "Add/Replace Recipe" from the File Operations menu.

If the recipe existed, the SIMATIC Automation Tool replaces it. If the recipe did not exist, the SIMATIC Automation Tool adds it.

The Event Log below the device table shows the results of this operation.

After a successful Recipe download operation, the recipe file path is deleted.

WARNING

Security note

Operating a process or machine with compromised data could affect the operation of an online process or machine. Unexpected operation of a process or machine could result in death or injury to personnel and/or property damage.

Make sure that you protect Recipe .csv files from being compromised through the use of different methods, for example, by limiting network access and using firewalls.

4.12 Upload and delete Data Logs in CPUs

The Data Log upload operation works for CPUs that have Data Logs in external load memory (SIMATIC memory card). The CPU can be in either RUN or STOP mode.

The Data Log delete operation only works for CPUs in STOP mode. If you select to delete one or more Data Logs (from one or more CPUs) and any of the CPUs are found to be in RUN mode, then you are prompted that all CPUs must be placed in STOP mode before attempting the operation. If you choose not to switch to STOP mode, the entire delete operation is stopped.

Data Logs are uploaded as .CSV (comma-separated values) text files.

You can select multiple data files from one or more CPUs and process all the selected files in a single operation.

SIMATIC Automation Tool creates a unique folder name for each CPU, to store uploaded Data Log files on your PG/PC. A folder name is created from the CPU name combined with the MAC address. If you select and upload the same Data Log file twice, a number is appended to the filename, to make all filenames unique.

SIMATIC Automation Tool must have Read access to upload Data Log files and Full access (read and write) authorization to delete Data Log files from a CPU. Therefore, you may have to enter a password to successfully perform a delete operation. If you do not enter a password, or if the password does not authorize the CPU write access, the delete operation(s) for that CPU will fail and an error message is put in the operations log.

Data Log actions

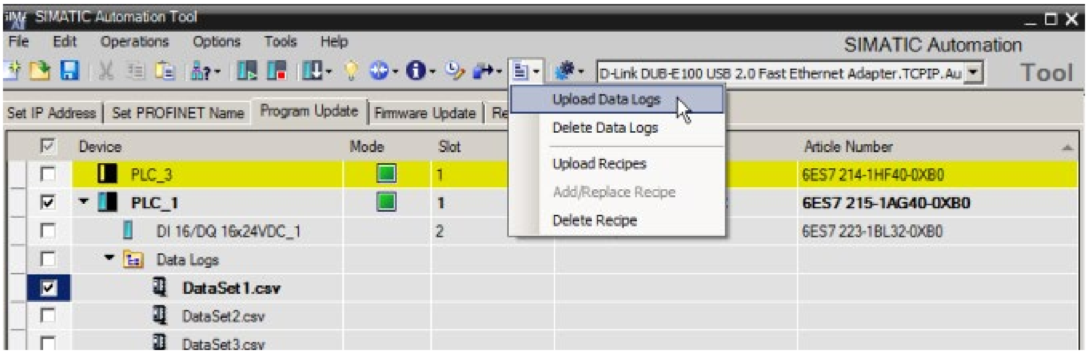
The File Operations toolbar button 📄▾ and File Operations menu provide the following menu commands:

- **Upload Data Logs:** Uploads a copy of selected Data Log file(s) from the CPU to PG/PC. The SIMATIC Automation Tool copies the files to the directory assigned in the Data Logs settings (Page 104).
- **Delete Data Logs:** Deletes selected Data Log files that are stored in a CPU.

Upload or Delete Data Log files

To upload or delete data log files, follow these steps:

1. Expand a CPU row and make any Data Log folders visible.
2. Expand a Data Log folder and select Data Log files:
3. For each CPU, enter a password, if used, in the "Password in CPU" column of the currently open tab.
4. Select the "Operations > File Operations > Upload Data Logs" menu command or the "Operations > File Operations > Delete Data Logs" menu command. Alternatively, click the "File Operations" toolbar button and select the "Upload Data Logs" or "Delete Data Logs" command from the button drop-down menu.



☑	Device	Mode	Slot		Article Number
□	PLC_3	🟩	1		6ES7 214-1HF40-0XB0
☑	PLC_1	🟩	1		6ES7 215-1AG40-0XB0
□	DI 16/DQ 16x24VDC_1		2		6ES7 223-1BL32-0XB0
□	Data Logs				
☑	DataSet1.csv				
□	DataSet2.csv				
□	DataSet3.csv				

The Event log below the device table shows the results of your operation.

WARNING

Security note

Operating a process or machine with compromised data could affect the operation of an online process or machine. Unexpected operation of a process or machine could result in death or injury to personnel and/or property damage.

Make sure that you protect Data Log .csv files from being compromised through the use of different methods, for example, by limiting network access and using firewalls.

4.13 Install new firmware in devices

TIA Portal firmware update

You can use a SIMATIC memory card to install firmware updates in devices. Alternative firmware update methods include using the Module Information page of a CPU's built-in Web server, or using the TIA portal online and diagnostic functions.

Note

HMI operating system, and runtime software updates

You must use the SIMATIC Automation Tool Program Update operation (Page 56) to update the HMI operating system, and runtime software. You do not have the option to select a partial update. The Program Update operation updates all data components as necessary, for a consistent download.

SIMATIC Automation Tool firmware update

The SIMATIC Automation Tool can perform firmware updates on a group of devices. You can use the new format single .upd file and the older (classic) format which uses three or more separate .upd files.

If you have a chain communication topology and the Communications settings (Page 99) enable multi-threading, be aware of the risk of communication disruption with this operation.

Note

S7-1200 CM communication modules must be configured before a firmware update

You can use the SIMATIC Automation Tool to update the firmware in unconfigured and configured SM and CM modules, except for left-side S7-1200 CM devices. For an S7-1200 CM module, you must configure the CM module in the TIA Portal and download the configuration to the module before you can use the SIMATIC Automation Tool to update the CM firmware.

Note

CPU firmware downgrade

You can use the SIMATIC Automation Tool to downgrade CPU firmware (load a previous firmware version), but the IP address and program might be erased. In this case, the IP address is reset to 0.0.0.0 and a new network scan is required to communicate with this device. You must set the IP address to restore your previous network address.

You cannot downgrade the firmware for some devices. Check your device documentation.

Note that programs for one CPU firmware version might not run on another firmware version. The CPU cannot go to RUN mode if the program is incompatible with the firmware version.

Preparing firmware update files for use with the SIMATIC Automation Tool

- You can obtain firmware update software from the customer support (https://www.siemens.com/automation/) web site.
- Another option is to select a device row and then select "Check for Firmware Updates" from either the Tools menu or the Tools toolbar icon. The SIMATIC Automation Tool launches the device's customer support web page. The Siemens support web page selection is controlled by the article number displayed in a device table row. For example, a "Check for updates" command on article number 6ES7 215-1HG31-0XB0 links to the corresponding CPU 1215C web support page (https://support.industry.siemens.com/cs/products/6es7215-1hg31-0xb0/cpu-1215c-dcdcrly-14di10do2ai2ao?pid=79072&dtp=Download&mlfb=6ES7215-1HG31-0XB0&lc=en-WW)

For a CPU example, the firmware update file named **6ES7 211-1AE40-0XB0**_V04.00.02**.exe** is only for the **CPU 1211C DC/DC/DC** model. If you use the .upd file within this package for any other S7-1200 CPU model, the update process will fail.

When you execute the update file and extract the files, you see the following set of files and folders.

- file: S7-JOB.SYS
- folder: FWUPDATE.SYS contains the .upd file.
  - file: **6ES7 211-1AE40-0XB0** V04.00.02**.upd** (.upd file used by the SIMATIC Automation Tool)

For an I/O module example, the firmware update file named **232-4HD32-0XB0**_V203**.exe** is only for the **SM 1232 ANALOG OUTPUT 4AO** module. The self-extracting .exe file contains the file **6ES7 232-4HD32-0XB0** V02.00.03_00.00.00.00**.upd** that is used by the SIMATIC Automation Tool**.**

Note

New format firmware update files

- The self-extracting .exe update package name must refer to the article number of the device that you want to update.
- The extracted .upd file name must match the article number of the device and the firmware version that you want to load.

Note

Old format firmware update files

- The self-extracting .exe update package name must refer to the article number of the device that you want to update.
- Contains three or more files depending on the firmware size.
- Create a folder with any name in the Firmware Update folder (Page 101). You can name the folder with the article number and version number so it will be easier to identify, but you can use any name. The SIMATIC Automation Tool parses all firmware files at startup to confirm exact firmware version numbers.

Copy .upd files to the firmware update folder

The new format firmware update single .upd files have the target module model and version numbers in their file names. You can copy multiple .upd files to a single firmware folder and then identify the target module by the .upd file name. Copy all the .upd files you need to the Firmware Update folder (Page 101).

> ⚠ WARNING
>
> **Verify that the CPU is not actively running a process before installing firmware updates**
>
> Installing a firmware update for a CPU or module causes the CPU to go to STOP mode, which could affect the operation of an online process or machine. Unexpected operation of a process or machine could result in death or injury to personnel and/or property damage.

Perform firmware updates for CPUs and modules

After .upd files are present in the firmware update folder, you can use the SIMATIC Automation Tool to update device firmware. Follow these steps to perform the update:

1. Click the "Firmware Update" tab.
2. Select one or more devices to include in the operation. You can use the Devices check box at the top of the device table to select or deselect all devices. Alternatively, you can use the right-click shortcut menu or the "Edit > Select" menu command to access the "Select All" and "Deselect All" commands.
3. For each device row that you selected, click the "New Firmware Version" column drop-down list and select a firmware version for either a CPU or module. The drop-down list shows the names of the .upd files that you copied to the firmware update folder (Page 101). If new firmware versions (.upd files) are available in the firmware update folder, then these files are available from the "New Firmware Version" drop-down list.

   You can also use the browse ... button and navigate to a folder on your PG/PC that contains firmware update files. Select a file to add it to the drop-down list. If the selected file has the same name as one of the files already listed, the SIMATIC Automation Tool adds a number to the new file to make the names unique. To help you identify files, a tooltip displays the entire path and filename.
4. For each selected device, enter a password, if used, in the "Password in CPU" column.
5. Select the "Operations > Update > Firmware Update" menu command to start the operation. Alternatively, click the Update toolbar button ⬇ and the "Firmware Update" command from the button drop-down menu.

The Event Log below the device table shows the results of the operation. After a successful Firmware update operation, the SIMATIC Automation Tool clears the "New Firmware Version" field.

Fail-Safe device CPU passwords for firmware update

For firmware versions earlier than S7-1200 V4.2 and S7-1500 V2.0, a password protected Fail-Safe CPU requires the safety F-CPU password (top level 5) for the firmware update operation. Since a firmware update is not a safety-relevant operation, the password level requirement was relaxed. Later versions of F-CPU firmware require only read-write access (level 4 Full access).

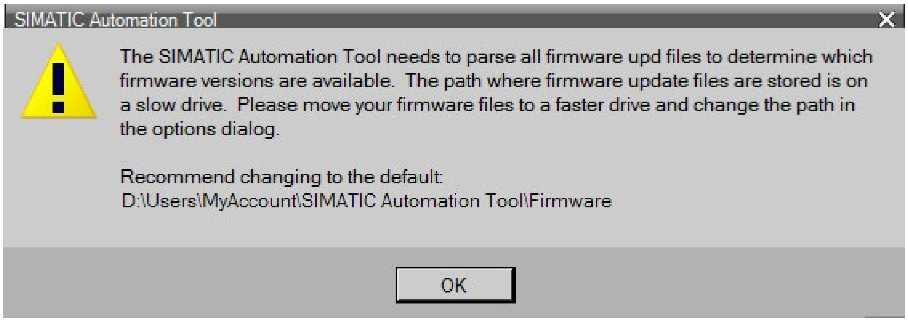
The SIMATIC Automation Tool does not check the legitimization level for different firmware versions. The operation is initiated and the device will reject a password based on the implementation in the firmware and an Event Log error is provided as follows:

"The password entered is not sufficient to complete the operation."

S7-1200 V4.2 or later and S7-1500 V2.0 or later require only read-write "Full access (no protection)". If the safety F-CPU password is used then this also gives you read-write access, so the operation will always succeed if the Fail-Safe password is entered.

Timeout error message due to slow communication with .upd file storage device

If you see the following error message box, then more than ten seconds has elapsed and the SIMATIC Automation Tool has not completed processing all the .upd files in the firmware storage folder. The time required to open and scan all the .upd files depends on data access time and the number of .upd files in the folder.



The SIMATIC Automation Tool needs to parse all firmware upd files to determine which firmware versions are available. The path where firmware update files are stored is on a slow drive. Please move your firmware files to a faster drive and change the path in the options dialog.

Recommend changing to the default:
D:\Users\MyAccount\SIMATIC Automation Tool\Firmware

This timeout error can occur when communication with a remote storage device is too slow.

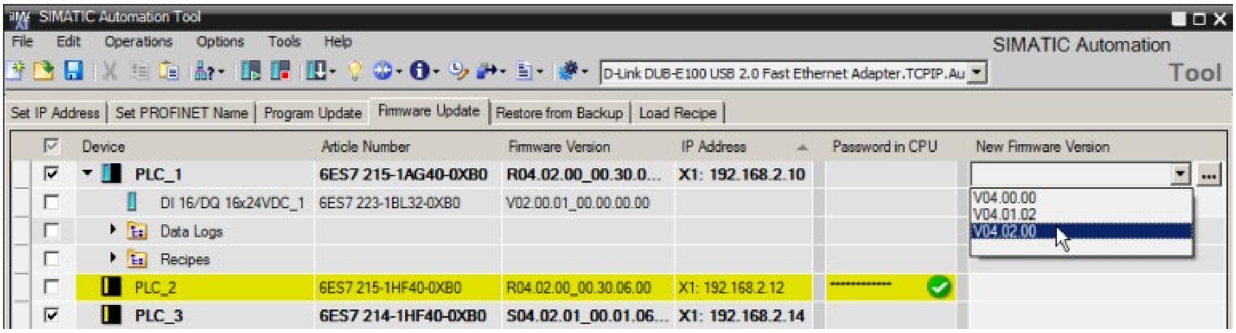
To prevent this problem, assign a faster firmware data storage path from the Firmware Update section of the "Options > Settings" menu command dialog. Copy the .upd files you need to a faster local storage device and try the operation again.

Example firmware update

This example shows how to update the firmware for a single CPU.

To perform the firmware update, follow these steps:

1. Open the drop-down list of available versions from the "New Firmware Version" column. The drop-down list shows all of the available firmware update files in the firmware update folder (Page 101).
2. Select the firmware update version to use. (If you had selected more than one device, then you would choose an update file for each selected device.)
3. Select the "Operations > Update > Firmware Update" menu command to start the operation. Alternatively, click the Update toolbar button followed by the Firmware Update menu command from the button drop-down menu.



D-Link DUB-E100 USB 2.0 Fast Ethernet Adapter.TCPIP.Au

☑	Device	Article Number	Firmware Version	IP Address	Password in CPU	New Firmware Version
☑	PLC_1	6ES7 215-1AG40-0XB0	R04.02.00_00.30.0...	X1: 192.168.2.10		V04.00.00 V04.01.02 V04.02.00
□	DI 16/DQ 16x24VDC_1	6ES7 223-1BL32-0XB0	V02.00.01_00.00.00.00			
□	Data Logs					
□	Recipes					
□	PLC_2	6ES7 215-1HF40-0XB0	R04.02.00_00.30.06.00	X1: 192.168.2.12	************* ✓	
☑	PLC_3	6ES7 214-1HF40-0XB0	S04.02.01_00.01.06...	X1: 192.168.2.14		

Note

You cannot update the firmware of some S7-1200 modules with the SIMATIC Automation Tool

If you see the error message "The device requires both the CPU and module to support firmware update. This device can only be updated via SD card", then you cannot update the module firmware with the SIMATIC Automation Tool.

4.14 Backup and Restore CPU or HMI data

Backing up a device

The "Backup Device to File" command creates new data backup files and copies the files to the backup and restore folder (Page 103).

You can use these files in the SIMATIC Automation Tool Restore Device operation.

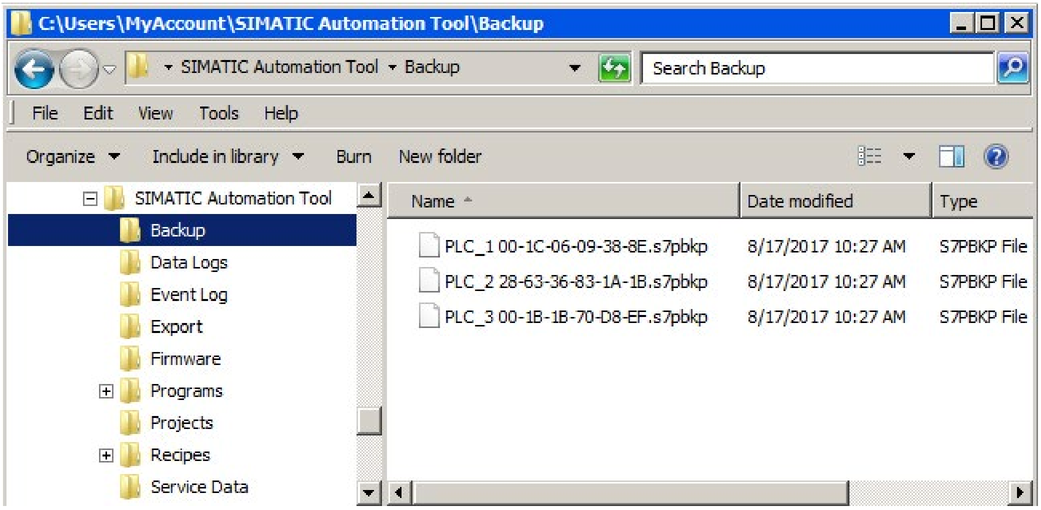
You can start the backup operation from any tab selection.

To create a backup file, follow these steps:

1. Select one or more devices to include in the operation. You can use the "Devices" check box at the top of the device table to select or deselect all devices. Alternatively, you can use the right-click shortcut menu or the "Edit > Select" menu command to access the "Select All" and "Deselect All" commands.
2. Select the "Backup/Restore > Backup Device to File" menu command from the Operations menu. Alternatively, click the Backup/Restore toolbar button and select "Backup Device to File" from the button drop-down menu.

The Event Log below the device table shows the results of the operation.

For a successful operation, the SIMATIC Automation Tool creates a backup file name for S7 and HMI devices. The file name combines the project name, MAC address, and .s7pbkp. The SIMATIC Automation Tool copies the files to the backup and restore folder (Page 103).



Name	Date modified	Type
PLC_1 00-1C-06-09-38-8E.s7pbkp	8/17/2017 10:27 AM	S7PBKP File
PLC_2 28-63-36-83-1A-1B.s7pbkp	8/17/2017 10:27 AM	S7PBKP File
PLC_3 00-1B-1B-70-D8-EF.s7pbkp	8/17/2017 10:27 AM	S7PBKP File

Restoring devices from backup files

You use the "Restore Device from File" command to restore backup files to the corresponding devices. S7 and HMI backup files that you created with the "Backup Device to File" command have the extension name "s7pbkp". You can restore files from the backup and restore folder (Page 103) or browse to another location.

If you have a chain communication topology and the Communications settings (Page 99) enable multi-threading, be aware of the risk of communication disruption with this operation.

Note

Fail-Safe devices

If a Fail-Safe CPU is protected, you must enter the safety F-CPU password in the "Password in CPU" column to restore a Fail-Safe device from a backup file.

You must confirm an additional prompt and reselect your device if the program in an F-CPU is a safety program (Page 110).

The SIMATIC Automation Tool places "Restore from backup" requests for Fail-Safe devices in the F-CPU safety-relevant operation queue. The SIMATIC Automation Tool uses only single-thread sequential processing for the safety-relevant operation queue.

The destination device for a safety program must be a Fail-Safe CPU.

WARNING

Verify that the device is not actively running a process before restoring a device from a backup file

Restoring a device causes the CPU to go to STOP mode, which could affect the operation of an online process or machine. Unexpected operation of a process or machine could result in death or injury to personnel and/or property damage.

To restore selected devices from a backup file, follow these steps:

1. Click the "Restore from Backup" tab in the device table.
2. Select one or more devices to include in the operation. You can use the Devices check box at the top of the device table to select or deselect all devices. Alternatively, you can use the right-click shortcut menu or the "Edit > Select" menu command to access the "Select All" and "Deselect All" commands.
3. For each device, select a backup file name from the "Backup File" drop-down list. The drop-down list shows the names of the .s7pbkp files that exist in the backup and restore folder (Page 103).

   You can also use the browse ... button and navigate to the folder on your PG/PC that contains backup files. Select a file to add it to the drop-down list. If the file that you selected has the same name as an existing file, the SIMATIC Automation Tool adds a number to the new file name to make the names unique.

   For valid files, a tooltip displays the path and filename. You also see a green check mark by the file name.

   For invalid files, the tooltip displays the file error, which the Event Log also displays. You see a red X by the file name.

4. Enter passwords, if used, in the "Password in CPU" and "Password in Backup File" columns. Restore from Backup is a safety-relevant operation and the safety F-CPU password is required for a password protected Fail-Safe device.
5. Select the "Operations > Backup/Restore > Restore Device from File" menu command to start the operation. Alternatively, click the Backup/Restore toolbar button and select "Restore Device from File" from the button drop-down menu.

Backup file validation

Before starting the restore operation, the SIMATIC Automation Tool performs limited data checks on the backup file data.

- The file extension name and header data are validated
- You cannot restore from a backup file that contains a safety program when the target device is not a Fail-Safe CPU.

If a backup file is not valid, the SIMATIC Automation Tool displays a red "X" in the "Backup File" field. Additional error information is available in a tooltip when you hover over the cell.

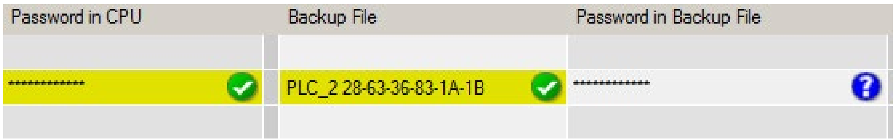
Password handling after the restore operation

If a CPU is password-protected, then you must supply a password for the Restore from Backup operation to complete successfully.

After you restore a backup file to a CPU, the new file might have a password. The password you restored might be different from the previous password, if the CPU had a password. You must therefore enter a second password in the "Password in Backup File" column. The second password becomes the CPU password after the restore operation completes.

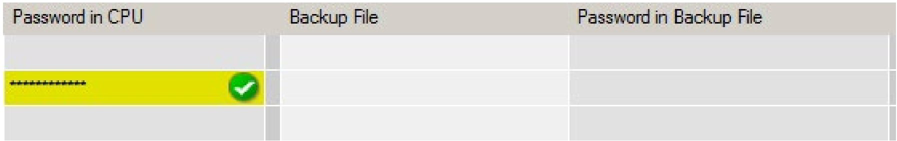
After a successful Restore operation, the SIMATIC Automation Tool automatically copies the second password (the "Password in Backup File" that you entered) to the CPU password field and attempts a connection using the new password. The SIMATIC Automation Tool then deletes the second password and backup file path.

Before you restore a backup file to a CPU:



Password in CPU	Backup File	Password in Backup File
************* ✓	PLC_2 28-63-36-83-1A-1B ✓	************* ?

After you restore a backup file to a CPU:



Password in CPU	Backup File	Password in Backup File
************* ✓		

If the password that you entered in the "Password in Backup File" column is incorrect and is not actually a password configured in the restored CPU data, then the Event Log shows a warning after the operation completes. In this case the CPU password shows a red ‘X’ icon to indicate an invalid password.

F-signature validation

A TIA Portal project that contains a safety program has an F-signature. The SIMATIC Automation Tool uses the F-signature to verify the data in a program file, which provides an additional level of security for safety programs. After a Restore from backup file operation, the SIMATIC Automation Tool compares the F-Signature in the project file to the F-Signature that is now in the CPU device program.

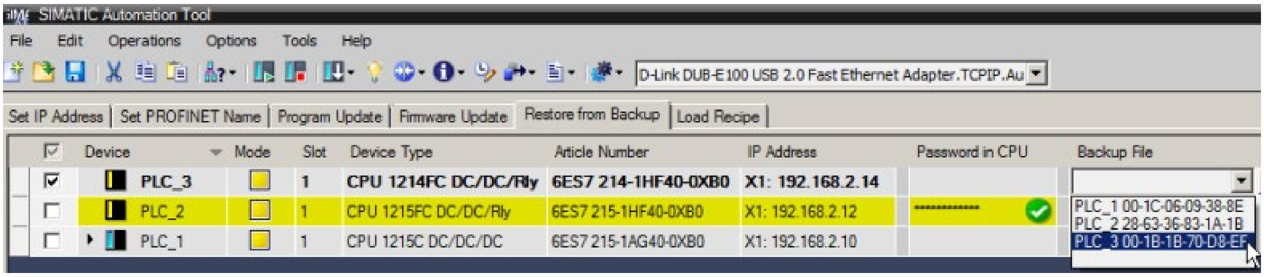
The Event Log reports a successful comparison is reported as: "Result of CRC comparison, online and offline collective F-signatures match"

The SIMATIC Automation Tool reports an unsuccessful comparison in the Event Log as:" Result of CRC comparison, online and offline collective F-signatures do not match" In the event of an unsuccessful comparison, reset the device to factory defaults (Page 79) and reattempt the program update. Do not attempt to go to RUN mode if you are not sure that you have loaded the correct safety program.

Restore from backup example

This example shows one selected device and the selection of one backup file for the "Backup File" field. For multiple devices, you would select a backup file to restore for each device.

The Event Log below the device table shows the results of this operation.



☑	Device	Mode	Slot	Device Type	Article Number	IP Address	Password in CPU	Backup File
☑	PLC_3	🟨	1	CPU 1214FC DC/DC/Rly	6ES7 214-1HF40-0XB0	X1: 192.168.2.14		
☐	PLC_2	🟨	1	CPU 1215FC DC/DC/Rly	6ES7 215-1HF40-0XB0	X1: 192.168.2.12	************ ✓	
☐	PLC_1	🟨	1	CPU 1215C DC/DC/DC	6ES7 215-1AG40-0XB0	X1: 192.168.2.10		

PLC_1 00-1C-06-09-38-8E
PLC_2 28-63-36-83-1A-1B
PLC_3 00-1B-1B-70-D8-EF

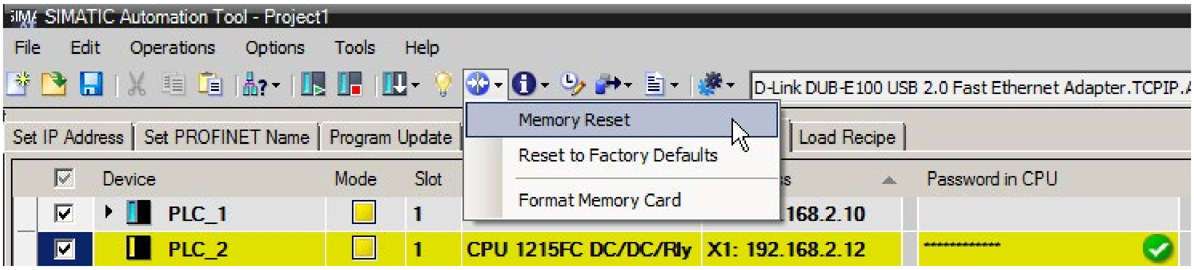
4.15 Reset CPU memory

Reset memory on selected CPUs

To reset CPU memory on selected devices, follow these steps:

1. Select one or more devices to include in the operation. You can use the Devices check box at the top of the device table to select or deselect all devices. Alternatively, you can use the right-click shortcut menu or the "Edit > Select" menu command to access the "Select All" and "Deselect All" commands.
2. For each selected CPU, enter a password, if used, in the "Password in CPU" column.
3. Select the "Operations > Reset > Memory Reset" command to start the operation. Alternatively, click the "Reset" toolbar button and select "Memory Reset" from the button drop-down menu.
4. Click the "Continue" button on the "Memory Reset" dialog box.

The SIMATIC Automation Tool performs a memory reset on the selected devices.



☑	Device	Mode	Slot		...s	Password in CPU
☑	PLC_1	🟨	1		168.2.10	
☑	PLC_2	🟨	1	CPU 1215FC DC/DC/Rly	X1: 192.168.2.12	************* ✓

The Event log below the device table shows the results of this operation.

4.16 Reset CPUs and modules to factory default values

Reset selected devices to factory default values

You can reset selected devices to factory default values, except for the IP address. The device retains the existing IP address so your network IP assignments are preserved.

If you have a chain communication topology and the Communications settings (Page 99) enable multi-threading, be aware of the risk of communication disruption with this operation.

Note

Fail-Safe devices

If a Fail-Safe CPU is protected, you must enter the safety F-CPU password in the "Password in CPU" column to reset a Fail-Safe device to factory default values

You must confirm an additional prompt and reselect your device if the program in the F-CPU is a safety program (Page 110).

Reset to factory defaults requests for Fail-Safe devices are placed in the safety-relevant operation queue and only single-thread sequential processing is allowed.

⚠ WARNING

Verify that the device is not actively running a process before a Reset to factory defaults operation

A Reset to factory defaults operation causes the CPU to go to STOP mode, which could affect the operation of an online process or machine. Unexpected operation of a process or machine could result in death or injury to personnel and/or property damage.

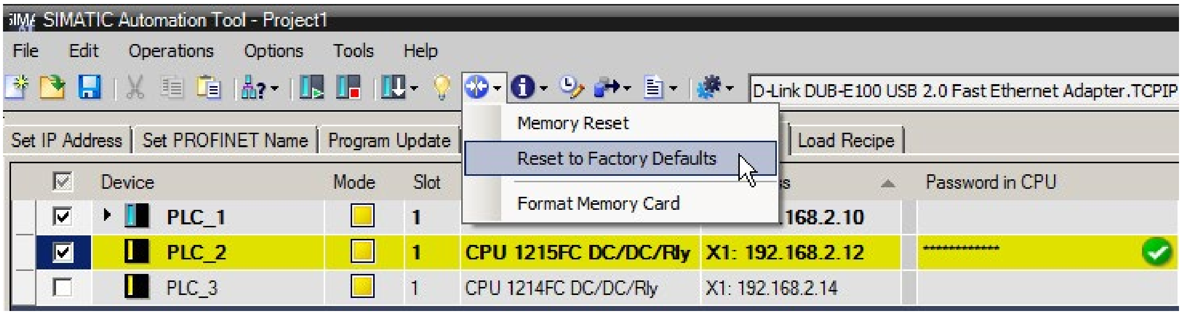
To reset selected devices to factory default values, follow these steps:

1. Select one or more devices to include in the operation. You can use the Devices check box at the top of the device table to select or deselect all devices. Alternatively, you can use the right-click shortcut menu or the "Edit > Select" menu command to access the "Select All" and "Deselect All" commands.

   For each selected CPU, enter a password, if used, in the "Password in CPU" column of the currently open tab. Reset to factory defaults is a safety-relevant operation and the safety F-CPU password is required for a protected Fail-Safe device.

2. Select the "Operations > Reset > Reset to Factory Defaults" menu command to start the operation. Alternatively, click the "Reset" toolbar button and select "Reset to Factory Defaults" from the button drop-down menu.

   For Fail-Safe CPUs, the SIMATIC Automation Tool displays the "Program Update" dialog for additional confirmation. Select the device, devices or all devices that you want to reset to factory defaults.



☑	Device	Mode	Slot		...s	Password in CPU
☑	PLC_1	🟨	1		...168.2.10	
☑	PLC_2	🟨	1	CPU 1215FC DC/DC/Rly	X1: 192.168.2.12	************* ✓
□	PLC_3	🟨	1	CPU 1214FC DC/DC/Rly	X1: 192.168.2.14	

3. Click the "Continue" button on the "Reset to Factory" dialog.

   The SIMATIC Automation Tool resets the selected devices to factory default values.

4. Allow time for the reset to complete. Wait until the device lights stop flashing before attempting another operation.

Note

Reset to Factory operation does not clear SIMATIC memory card

If you have a SIMATIC memory card in a CPU, a "Reset to Factory" operation does not clear the contents. If you do not have a SIMATIC memory card in a CPU, "Reset to Factory" clears the program in the internal load memory of the CPU.

The Event log below the device table shows the results of the operation.

4.17 Format memory card

SIMATIC memory cards plug into SIMATIC devices and support a variety of purposes. Depending on the device type or device family, you can use memory cards for the following purposes:

- Load memory of a CPU
- Storage medium for projects
- Firmware backups and updates
- Storage medium for the PROFINET device name
- Project transfer from one device to another
- Other files

Note

Use only Siemens software to format SIMATIC memory cards

If you use a SIMATIC memory card for non-SIMATIC purposes or you format it incorrectly, the internal structure of the SIMATIC memory card is overwritten. The structure is not recoverable and the SIMATIC memory card becomes unusable for SIMATIC devices.

Do not use SIMATIC memory cards for non-SIMATIC-related purposes and do not format SIMATIC memory cards with third-party devices or Windows tools.

If you have a chain communication topology and the Communications settings (Page 99) enable multi-threading, be aware of the risk of communication disruption with this operation.

⚠WARNING

Verify that the device is not actively running a process before formatting a memory card

Formatting a memory card causes a CPU to go to STOP mode, which could affect the operation of an online process or machine. Unexpected operation of a process or machine could result in death or injury to personnel and/or property damage.

To format SIMATIC memory cards on selected devices, follow these steps:

1. Select one or more devices to include in the operation. You can use the Devices check box at the top of the device table to select or deselect all devices. Alternatively, you can use the right-click shortcut menu or the "Edit > Select" menu command to access the "Select All" and "Deselect All" commands.
2. For each selected CPU, enter a password, if used, in the "Password in CPU" column of the currently open tab.
3. Select the "Operations > Reset > Format Memory Card" command to start the operation. Alternatively, click the "Reset" toolbar button and select "Format Memory Card" from the button drop-down menu.
4. Click the "Continue" button on the "Format Memory Card" dialog box.

The SIMATIC Automation Tool formats the memory card of the selected devices.

Note

Fail-Safe devices

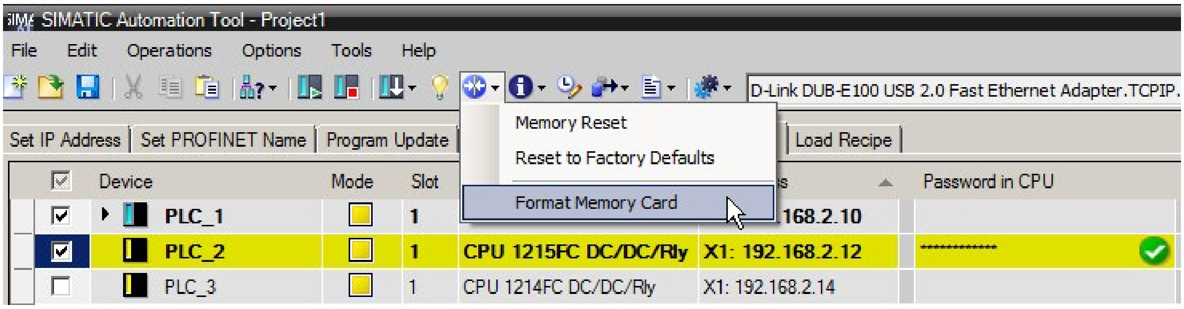
If a Fail-Safe CPU is protected, then you must enter the safety F-CPU in the "Password in CPU" column to format a SIMATIC memory card in a Fail-Safe device.

You must confirm an additional prompt and reselect your device if the program in the F-CPU is a safety program (Page 110).

Format memory card requests for Fail-Safe devices are placed in the safety-relevant operation queue and only single-thread sequential processing is allowed.

In this example, the SIMATIC Automation Tool formats the memory cards of the selected devices when you click the "Continue" button.

The Event log below the device table shows the results of this operation.



D-Link DUB-E100 USB 2.0 Fast Ethernet Adapter.TCPIP.

☑	Device	Mode	Slot		s	Password in CPU
☑	PLC_1	🟨	1		168.2.10	
☑	PLC_2	🟨	1	CPU 1215FC DC/DC/Rly	X1: 192.168.2.12	************* ✓
□	PLC_3	🟨	1	CPU 1214FC DC/DC/Rly	X1: 192.168.2.14	

4.18 Retrieve Service Data from CPUs

When a CPU enters a defective state, the CPU saves fault information that you can upload to your PG/PC. You can send this Service Data to Siemens customer support and help find the cause of a fault.

You can retrieve Service Data when the CPU is in STOP or RUN mode. The Service Data contains multiple files that are compressed into a single .zip file with a file name based on the PLC name, date, and time. A unique number in parentheses is appended to the file name to avoid duplicate file names.

You configure or accept the default Service Data path from the Service Data settings (Page 102).

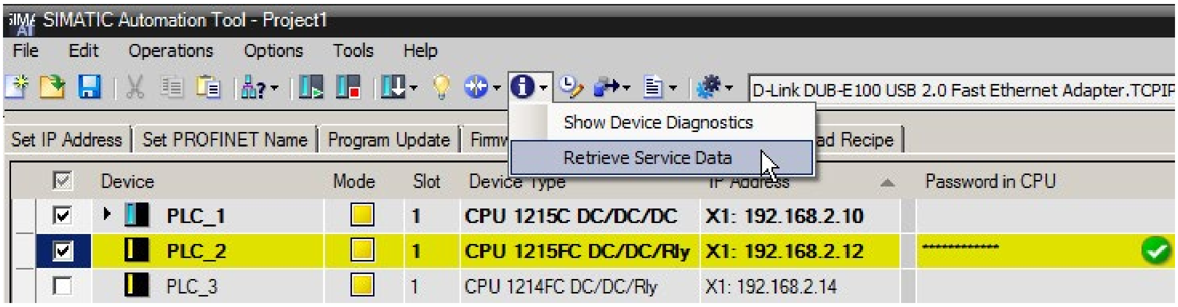
To retrieve Service Data from selected CPUs, follow these steps:

1. Select one or more devices to include in the operation. You can use the Devices check box at the top of the device table to select or deselect all devices. Alternatively, you can use the right-click shortcut menu or the "Edit > Select" menu command to access the "Select All" and "Deselect All" commands.
2. For each selected CPU, enter a password, if used, in the "Password in CPU" column of the currently open tab.
3. Select the "Operations > Diagnostics > Retrieve Service Data" command to start the operation. Alternatively, click the "Show device diagnostics" toolbar button ⓘ and select "Retrieve Service Data" from the button drop-down menu.
4. Click the "Continue" button on the "Upload Service Data" dialog box.

The SIMATIC Automation Tool retrieves Service Data from the selected CPUs and stores the files in the Service Data folder (Page 102).

Example:

After you click the "Continue" button, the SIMATIC Automation Tool retrieves the Service Data from the selected devices.



☑	Device	Mode	Slot	Device Type	IP Address	Password in CPU
☑	PLC_1	🟨	1	CPU 1215C DC/DC/DC	X1: 192.168.2.10	
☑	PLC_2	🟨	1	CPU 1215FC DC/DC/Rly	X1: 192.168.2.12	************** ✓
□	PLC_3	🟨	1	CPU 1214FC DC/DC/Rly	X1: 192.168.2.14	

The Event log below the device table shows the results of the operation.

Service Data files

Note

Retrieving Service Data files from password-protected CPUs

If a CPU is password protected, then you must provide a password with read access or full access to retrieve the Service Data files. Enter CPU passwords in the SIMATIC Automation Tool's "Password in CPU" column before you execute the "Retrieve Service Data" command.

Example S7-1200 service data file: PLC_1 00-1C-06-13-58-10.zip

Contents of .zip file:

ResourceStats.txt

RAM.img

PLCInformation.txt

NAND.img

General.txt

Fault.bin

DNN.txt

CommBuffers.txt

ASLog.txt

Alarms.txt

⚠ **WARNING**

Service Data is clear text

A malicious user could read the service Data files to obtain status and configuration details about the control system. The CPU stores the Service Data files in clear text (binary encoding), which is unencrypted. A CPU password can control access to this information.

Operating a process or machine with compromised data could affect the operation of an online process or machine. Unexpected operation of a process or machine could result in death or injury to personnel and/or property damage.

Use the TIA portal device configuration to set up CPU protection with a strong password. Strong passwords are at least ten characters in length, mixed letters, numbers, and special characters, are not words that can be found in a dictionary, and are not names or identifiers that can be derived from personal information. Keep the password secret and change it frequently.

4.19 Set time in CPUs

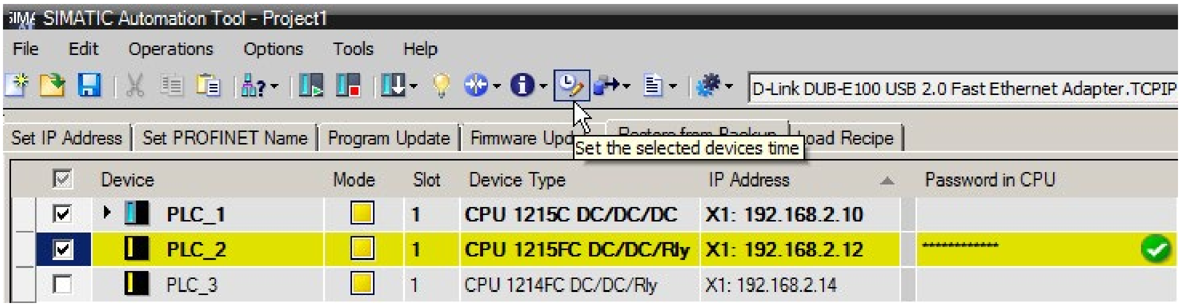
Set time in CPUs to current PG/PC time

The Time button sets the time for selected CPUs to your current PG/PC time. Time transformation information for time zone and daylight saving time is not changed and must be modified in the TIA Portal Project.

> ⚠ **WARNING**
>
> **Changing the CPU time of day could disrupt process operation**
>
> Changing the CPU time of day could cause process disruption to STEP 7 programs that execute program logic based on the time of day.
>
> Unexpected operation of a process or machine could result in death or injury to personnel and/or property damage.
>
> Ensure that changing the time of day does not cause unwanted effects in the STEP 7 program.

To set the CPU time to the PG/PC time, follow these steps:

1. Select one or more devices to include in the operation. You can use the Devices check box at the top of the device table to select or deselect all devices. Alternatively, you can use the right-click shortcut menu or the "Edit > Select" menu command to access the "Select All" and "Deselect All" commands.
2. For each selected CPU, enter a password, if used, in the "Password in CPU" column of the currently open tab.
3. Select the "Operations > Set Time" menu command or click the "Set the selected devices time" toolbar button:



☑	Device	Mode	Slot	Device Type	IP Address	Password in CPU
☑	PLC_1	🟨	1	CPU 1215C DC/DC/DC	X1: 192.168.2.10	
☑	PLC_2	🟨	1	CPU 1215FC DC/DC/Rly	X1: 192.168.2.12	************* ✓
□	PLC_3	🟨	1	CPU 1214FC DC/DC/Rly	X1: 192.168.2.14	

4. Click the "Continue" button on the "Set Time" dialog box.

The SIMATIC Automation Tool sets the system time on the selected devices to your current PG/PC time.

The Event log below the device table shows the results of this operation.

4.20 Show CPU diagnostic buffer

CPU diagnostic buffer

A CPU diagnostics buffer contains an entry for each diagnostic event. Each entry includes the date and time the event occurred, an event category, and an event description. The diagnostic buffer displays the entries in chronological order with the most recent event at the top. When the log is full, a new event replaces the oldest event in the log. When power is lost, the events are saved.

To show device diagnostics, follow these steps:

1. Select one or more CPUs in the device table. If you want to deselect all devices first, you can deselect the Devices check box at the top of the device table. Alternatively, you can also use the right-click shortcut menu or the "Edit > Select" menu to access the "Deselect All" command.
2. For each selected CPU, enter a password, if used, in the "Password in CPU" column of the currently open tab.
3. Select the "Operations > Diagnostics > Show CPU Diagnostics" menu command. Alternatively, click the "Show device diagnostics" toolbar button ℹ and select "Show CPU Diagnostics" from the button drop-down menu.

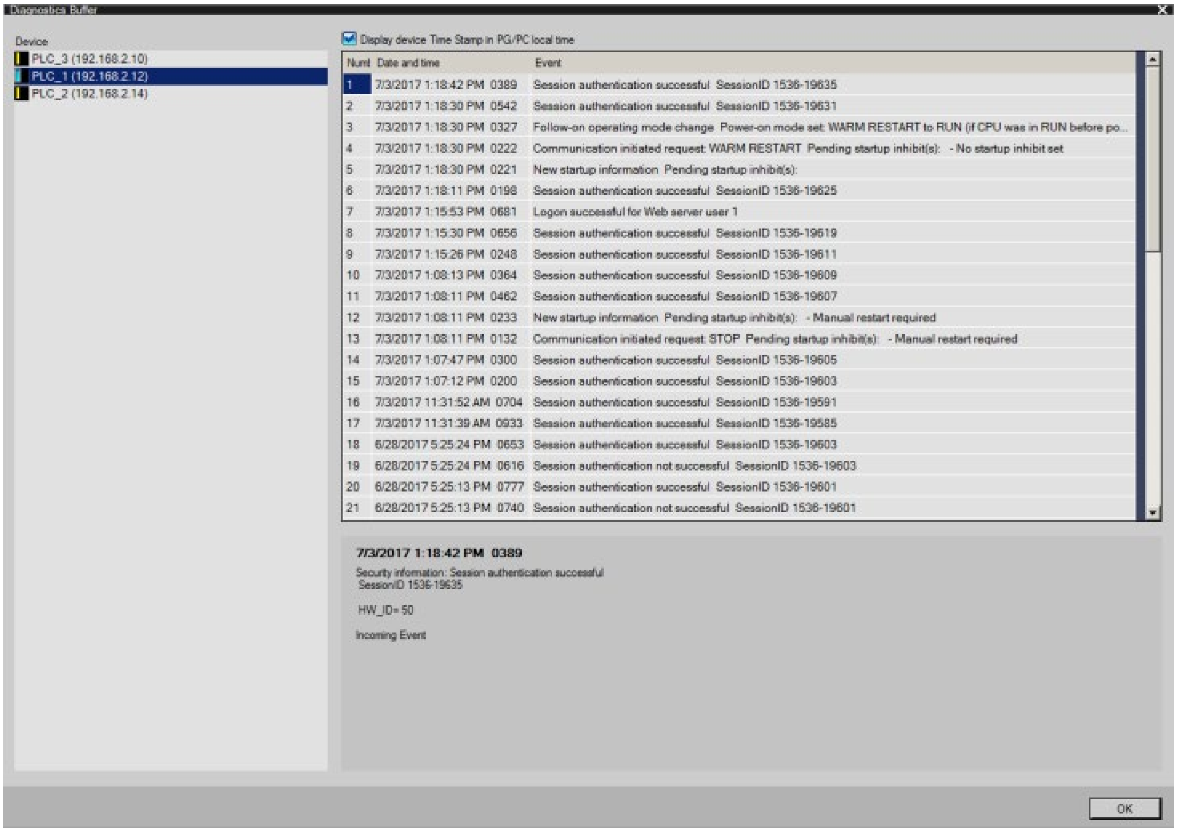
The SIMATIC Automation Tool then displays a dialog that includes the diagnostic buffers of the selected CPUs. You can select a CPU from the device list on the left to see the diagnostic buffer for that CPU.

Example diagnostic log

The diagnostics buffer contains the following types of entries:

- System diagnostic event (each CPU error and module error)
- CPU state changes (each power up, each transition to STOP, each transition to RUN)

You can use the "Display CPU Time Stamp in PG/PC local time" check box to view time stamps in local time or UTC time (Coordinated Universal Time).



☑ Display device Time Stamp in PG/PC local time

Num	Date and time	Event
1	7/3/2017 1:18:42 PM 0389	Session authentication successful SessionID 1536-19635
2	7/3/2017 1:18:30 PM 0542	Session authentication successful SessionID 1536-19631
3	7/3/2017 1:18:30 PM 0327	Follow-on operating mode change Power-on mode set: WARM RESTART to RUN (if CPU was in RUN before po...
4	7/3/2017 1:18:30 PM 0222	Communication initiated request: WARM RESTART Pending startup inhibit(s): - No startup inhibit set
5	7/3/2017 1:18:30 PM 0221	New startup information Pending startup inhibit(s):
6	7/3/2017 1:18:11 PM 0198	Session authentication successful SessionID 1536-19625
7	7/3/2017 1:15:53 PM 0681	Logon successful for Web server user 1
8	7/3/2017 1:15:30 PM 0656	Session authentication successful SessionID 1536-19619
9	7/3/2017 1:15:26 PM 0248	Session authentication successful SessionID 1536-19611
10	7/3/2017 1:08:13 PM 0364	Session authentication successful SessionID 1536-19609
11	7/3/2017 1:08:11 PM 0462	Session authentication successful SessionID 1536-19607
12	7/3/2017 1:08:11 PM 0233	New startup information Pending startup inhibit(s): - Manual restart required
13	7/3/2017 1:08:11 PM 0132	Communication initiated request: STOP Pending startup inhibit(s): - Manual restart required
14	7/3/2017 1:07:47 PM 0300	Session authentication successful SessionID 1536-19605
15	7/3/2017 1:07:12 PM 0200	Session authentication successful SessionID 1536-19603
16	7/3/2017 11:31:52 AM 0704	Session authentication successful SessionID 1536-19591
17	7/3/2017 11:31:39 AM 0933	Session authentication successful SessionID 1536-19585
18	6/28/2017 5:25:24 PM 0653	Session authentication successful SessionID 1536-19603
19	6/28/2017 5:25:24 PM 0616	Session authentication not successful SessionID 1536-19603
20	6/28/2017 5:25:13 PM 0777	Session authentication successful SessionID 1536-19601
21	6/28/2017 5:25:13 PM 0740	Session authentication not successful SessionID 1536-19601

7/3/2017 1:18:42 PM 0389

Security information: Session authentication successful
SessionID 1536-19635

HW_ID= 50

Incoming Event

4.21 Execution order of operations

Diagnostic buffer entries correspond to toolbar button operations or menu commands. For each toolbar button command, the SIMATIC Automation Tools adds a single operation to the operations queue for each selected device row. For example, if you select 20 different CPUs and click the RUN button, then the SIMATIC Automation Tool adds 20 RUN operations to the queue.

For better performance, separate threads can run independently to initiate and execute the operations contained in the queue. You assign the number of simultaneous operations in the Communications settings (Page 99). Separate threads cannot simultaneously start jobs on one CPU to avoid race conditions. For example, one job cannot put the CPU in STOP mode while another job tries to place the same CPU in RUN mode.

Execution examples

Example 1:

If the operations queue contains 10 go to RUN jobs for different CPUs, then multiple threads work in parallel to put all the CPUs in RUN mode. Since the threads execute in parallel, there is no guarantee of the order that CPUs complete the transition to RUN mode. Communication speeds can be different and how fast the job completes can be different, for each CPU.

Example 2:

You can queue as many jobs of the same type as you want. For example, you can place 100 CPUs in STOP mode by selecting all 100 CPUs and clicking the STOP button. However, a dialog box with a progress bar is displayed until all 100 jobs are complete. This dialog box will block the start of another operation, until all the STOP operations are complete.

Saving your Device table information 5

5.1 Save/Save as - Device table stored in protected .sat format

Use the "File > Save/Save as" menu commands or click the Save button 💾 to store your device table information in an encrypted .sat file. After you save the SIMATIC Automation Tool project, you can use the "File > Open" menu command or Open button 📂 to restore this project's device table information. The project file does not save device operating mode, selection state, or confirmation of selection state data. You can use the Refresh command to read the operating mode states.

- You assign the folder for saving projects in the Projects settings (Page 100).
- You must provide a valid password to save a SIMATIC Automation Tool .sat project file.
- You must enter the correct password to reopen an existing SIMATIC Automation Tool .sat project file.

SIMATIC Automation Tool .sat file security

Protect your SIMATIC Automation Tool project with a strong password. Strong passwords follow these rules:

- Are at least ten characters in length
- Mix letters, numbers, and special characters
- Are not words that can be found in a dictionary
- Are not names or identifiers that can be derived from personal information

Keep the password secret and change it frequently.

Minimum SIMATIC Automation Tool password requirements

The SIMATIC Automation Tool enforces the following minimum password requirements:

- At least ten characters in length
- Mix of letters, numbers, and special characters

Your .sat files are protected

A valid password is required to decrypt and reopen a .sat file.

Project file compatibility with previous versions

The SIMATIC Automation Tool V3.1 supports safety-relevant operations that were prohibited in previous versions and older .sat project files do not contain necessary safety data in the project file.

Opening V1.x project files is not possible.

When you open a V2.x - V3.0 project file, you are notified a network scan must be performed before opening the project file. After the scan is complete, the file is opened and data from the opened file will be applied to devices which were found on the network scan.

5.2 Import/Export - Device table loaded from/stored in open .csv format

The SIMATIC Automation Tool provides the following menu commands for exporting and importing device tables:

- The **File>Export** menu command saves the device table in .csv (comma separated values) text format.
- The **File>Import** menu command reads a .csv text file and puts that data in the SIMATIC Automation Tool device table.

The first text line is a description header followed by one or more data lines. Data text must match the expected format, with 15 "," comma characters on each line of text. 15 comma characters separate the 16 data columns that you see in the export example.

The device table in the SIMATIC Automation Tool configures communication with a device group. If you put incorrect information in the cells of a device table or in an imported .csv file, then the affected device operation can fail. Correct the device data and try the operation again.

For security reasons, CPU passwords are not exported.

Note

Copying all rows and columns displayed in the Device table

If you want to create a document that shows your complete network as displayed in the Device table, then you can use the Device table and MS Excel.

1. Expand all rows in the Device table with the Edit>Expand All Devices command.
2. Enter Ctrl-A to highlight all rows and columns.
3. Enter Ctrl-C to copy the Device table data to the Windows clipboard.
4. Enter Ctrl-V and paste the clipboard data into a MS Excel worksheet.

Note

Importing .csv file from older SIMATIC Automation Tool versions

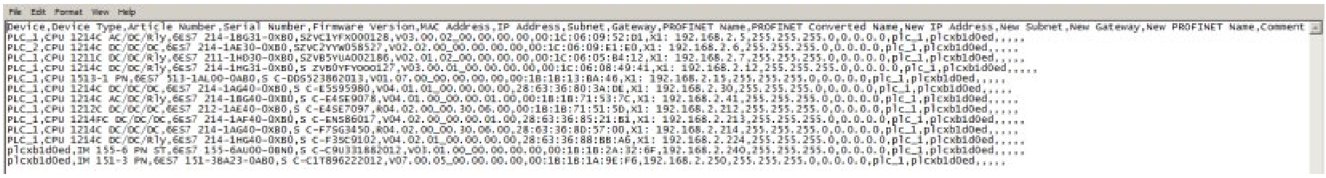
The number of device table columns and the names of the columns changed in the SIMATIC Automation Tool V3.1 compared to previous versions. Therefore, you cannot import .csv files created or formatted for previous versions.

The Import / Export settings (Page 106) provide the file path for import and export operations.

Export example

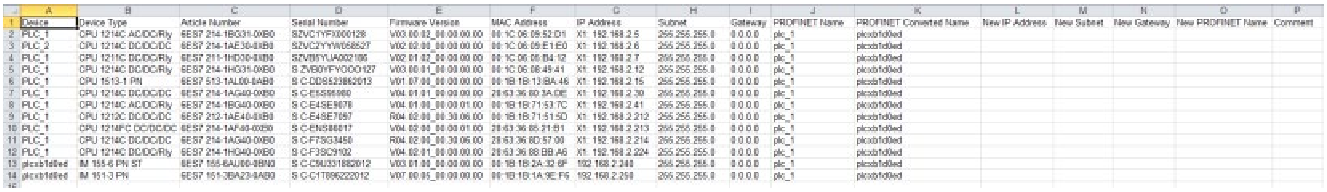
The following image shows the text format of a .csv file exported from the SIMATIC Automation Tool.

Note that the Export command only provides a list of the devices that are directly connected to the same subnet as the PG/PC running the SIMATIC Automation Tool and does not show CPU files or devices that are connected behind CPUs or IMs.



```
Device,Device Type,Article Number,Serial Number,Firmware Version,MAC Address,IP Address,Subnet,Gateway,PROFINET Name,PROFINET Converted Name,New IP Address,New Subnet,New Gateway,New PROFINET Name,Comment
PLC_1,CPU 1214C AC/DC/Rly,6ES7 214-1BG31-0XB0,SZVC1YFX000128,V03.00.02_00.00.00.00,00:1C:06:09:52:D1,X1: 192.168.2.5,255.255.255.0,0.0.0.0,plc_1,plcxb1d0ed,,,,,
PLC_2,CPU 1214C DC/DC/DC,6ES7 214-1AE30-0XB0,SZVC2YYW058527,V02.02.00_00.00.00.00,00:1C:06:09:E1:E0,X1: 192.168.2.6,255.255.255.0,0.0.0.0,plc_1,plcxb1d0ed,,,,,
PLC_1,CPU 1211C DC/DC/Rly,6ES7 211-1HD30-0XB0,SZVB5YUA002186,V02.01.02_00.00.00.00,00:1C:06:05:B4:12,X1: 192.168.2.7,255.255.255.0,0.0.0.0,plc_1,plcxb1d0ed,,,,,
PLC_1,CPU 1214C DC/DC/Rly,6ES7 214-1HG31-0XB0,S ZVB0YFYOOO127,V03.00.01_00.00.00.00,00:1C:06:08:49:41,X1: 192.168.2.12,255.255.255.0,0.0.0.0,plc_1,plcxb1d0ed,,,,,
PLC_1,CPU 1513-1 PN,6ES7 513-1AL00-0AB0,S C-DDS523862013,V01.07.00_00.00.00.00,00:1B:1B:13:BA:46,X1: 192.168.2.15,255.255.255.0,0.0.0.0,plc_1,plcxb1d0ed,,,,,
PLC_1,CPU 1214C DC/DC/DC,6ES7 214-1AG40-0XB0,S C-E5595980,V04.01.01_00.00.00.00,28:63:36:80:3A:DE,X1: 192.168.2.30,255.255.255.0,0.0.0.0,plc_1,plcxb1d0ed,,,,,
PLC_1,CPU 1214C AC/DC/Rly,6ES7 214-1BG40-0XB0,S C-E4SE9078,V04.01.00_00.00.01.00,00:1B:1B:71:53:7C,X1: 192.168.2.41,255.255.255.0,0.0.0.0,plc_1,plcxb1d0ed,,,,,
PLC_1,CPU 1212C DC/DC/DC,6ES7 212-1AE40-0XB0,S C-E4SE7097,R04.02.00_00.30.06.00,00:1B:1B:71:51:5D,X1: 192.168.2.212,255.255.255.0,0.0.0.0,plc_1,plcxb1d0ed,,,,,
PLC_1,CPU 1214FC DC/DC/DC,6ES7 214-1AF40-0XB0,S C-ENS86017,V04.02.00_00.00.01.00,28:63:36:85:21:B1,X1: 192.168.2.213,255.255.255.0,0.0.0.0,plc_1,plcxb1d0ed,,,,,
PLC_1,CPU 1214C DC/DC/DC,6ES7 214-1AG40-0XB0,S C-F7SG3450,R04.02.00_00.30.06.00,28:63:36:8D:57:00,X1: 192.168.2.214,255.255.255.0,0.0.0.0,plc_1,plcxb1d0ed,,,,,
PLC_1,CPU 1214C DC/DC/Rly,6ES7 214-1HG40-0XB0,S C-F3SC9102,V04.02.01_00.00.00.00,28:63:36:88:BB:A6,X1: 192.168.2.224,255.255.255.0,0.0.0.0,plc_1,plcxb1d0ed,,,,,
plcxb1d0ed,IM 155-6 PN ST,6ES7 155-6AU00-0BN0,S C-C9U331882012,V03.01.00_00.00.00.00,00:1B:1B:2A:32:6F,192.168.2.240,255.255.255.0,0.0.0.0,plc_1,plcxb1d0ed,,,,,
plcxb1d0ed,IM 151-3 PN,6ES7 151-3BA23-0AB0,S C-C1T896222012,V07.00.05_00.00.00.00,00:1B:1B:1A:9E:F6,192.168.2.250,255.255.255.0,0.0.0.0,plc_1,plcxb1d0ed,,,,,
```

The following image shows the same text file opened in Microsoft Excel.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	Device	Device Type	Article Number	Serial Number	Firmware Version	MAC Address	IP Address	Subnet	Gateway	PROFINET Name	PROFINET Converted Name	New IP Address	New Subnet	New Gateway	New PROFINET Name	Comment
2	PLC_1	CPU 1214C AC/DC/Rly	6ES7 214-1BG31-0XB0	SZVC1YFX000128	V03.00.02_00.00.00.00	00:1C:06:09:52:D1	X1: 192.168.2.5	255.255.255.0	0.0.0.0	plc_1	plcxb1d0ed					
3	PLC_2	CPU 1214C DC/DC/DC	6ES7 214-1AE30-0XB0	SZVC2YYW058527	V02.02.00_00.00.00.00	00:1C:06:09:E1:E0	X1: 192.168.2.6	255.255.255.0	0.0.0.0	plc_1	plcxb1d0ed					
4	PLC_1	CPU 1211C DC/DC/Rly	6ES7 211-1HD30-0XB0	SZVB5YUA002186	V02.01.02_00.00.00.00	00:1C:06:05:B4:12	X1: 192.168.2.7	255.255.255.0	0.0.0.0	plc_1	plcxb1d0ed					
5	PLC_1	CPU 1214C DC/DC/Rly	6ES7 214-1HG31-0XB0	S ZVB0YFYOOO127	V03.00.01_00.00.00.00	00:1C:06:08:49:41	X1: 192.168.2.12	255.255.255.0	0.0.0.0	plc_1	plcxb1d0ed					
6	PLC_1	CPU 1513-1 PN	6ES7 513-1AL00-0AB0	S C-DDS523862013	V01.07.00_00.00.00.00	00:1B:1B:13:BA:46	X1: 192.168.2.15	255.255.255.0	0.0.0.0	plc_1	plcxb1d0ed					
7	PLC_1	CPU 1214C DC/DC/DC	6ES7 214-1AG40-0XB0	S C-E5595980	V04.01.01_00.00.00.00	28:63:36:80:3A:DE	X1: 192.168.2.30	255.255.255.0	0.0.0.0	plc_1	plcxb1d0ed					
8	PLC_1	CPU 1214C AC/DC/Rly	6ES7 214-1BG40-0XB0	S C-E4SE9078	V04.01.00_00.00.01.00	00:1B:1B:71:53:7C	X1: 192.168.2.41	255.255.255.0	0.0.0.0	plc_1	plcxb1d0ed					
9	PLC_1	CPU 1212C DC/DC/DC	6ES7 212-1AE40-0XB0	S C-E4SE7097	R04.02.00_00.30.06.00	00:1B:1B:71:51:5D	X1: 192.168.2.212	255.255.255.0	0.0.0.0	plc_1	plcxb1d0ed					
10	PLC_1	CPU 1214FC DC/DC/DC	6ES7 214-1AF40-0XB0	S C-ENS86017	V04.02.00_00.00.01.00	28:63:36:85:21:B1	X1: 192.168.2.213	255.255.255.0	0.0.0.0	plc_1	plcxb1d0ed					
11	PLC_1	CPU 1214C DC/DC/DC	6ES7 214-1AG40-0XB0	S C-F7SG3450	R04.02.00_00.30.06.00	28:63:36:8D:57:00	X1: 192.168.2.214	255.255.255.0	0.0.0.0	plc_1	plcxb1d0ed					
12	PLC_1	CPU 1214C DC/DC/Rly	6ES7 214-1HG40-0XB0	S C-F3SC9102	V04.02.01_00.00.00.00	28:63:36:88:BB:A6	X1: 192.168.2.224	255.255.255.0	0.0.0.0	plc_1	plcxb1d0ed					
13	plcxb1d0ed	IM 155-6 PN ST	6ES7 155-6AU00-0BN0	S C-C9U331882012	V03.01.00_00.00.00.00	00:1B:1B:2A:32:6F	192.168.2.240	255.255.255.0	0.0.0.0	plc_1	plcxb1d0ed					
14	plcxb1d0ed	IM 151-3 PN	6ES7 151-3BA23-0AB0	S C-C1T896222012	V07.00.05_00.00.00.00	00:1B:1B:1A:9E:F6	192.168.2.250	255.255.255.0	0.0.0.0	plc_1	plcxb1d0ed					

When you use the SIMATIC Automation Tool and import an exported .csv file, you must do the following to display all CPU files and devices:

1. Import the file with the FileI>Import command.
2. Reenter any protected CPU passwords.
3. Select all devices in the Device table.
4. Refresh the Device table using the Operations>Scan Network>Refresh Status of Selected Devices command.

Menu, toolbar, shortcut keys and reference information 6

6.1 Main menu

The SIMATIC Automation Tool provides the following menus for device operations.

- File (Page 93)
- Edit (Page 94)
- Operations (Page 95)
- Options (Page 96)
- Tools (Page 107)
- Help (Page 107)

If you press the Alt key, the underlined letter indicates the Alt key you can use to activate a menu or sub-menu command.

Additionally, some you can activate some of the menu commands with shortcut key combinations (Page 109).

6.1.1 File menu

Tool icon	Menu command	Description
📄	**New**	Creates a new SIMATIC Automation Tool project
📂	**Open**	Displays an "Open" dialog where you can browse to a folder, select an .sat project file, and provide a password to open a protected project file. The "Open" dialog displays the projects folder (Page 100), but you can browse to any location for a project.
💾	**Save**	Saves (Page 89) the device table data in a .sat file. If there is no filename, then this operation uses the "Save As" command. The projects folder (Page 100) is the default folder for saving projects.
	Save As	Saves the device table data is saved in a .sat file. You can browse to a folder, assign a .sat project filename, and assign a password to protect the project file.
	Import	Imports (Page 90) data from a file in .csv format to the device table
	Export	Exports (Page 90) device table data to a file in .csv format.
	Exit	Closes the application. If the project was modified since the last save operation, then the "Save" operation is performed.

6.1.2 Edit menu

Tool icon	Menu command	Description
✂	Cut	Cut the selected data and copy this data to the clipboard. Clipboard entries are compatible with Excel, so data can be shared between the two applications. Read-only cells are not deleted.
⧉	Copy	Copy the selected data to the clipboard in Excel compatible format.
📋	Paste	Paste the data contained in the clipboard to selected field(s) in the SIMATIC Automation Tool. Read-only cells are not modified.
	Select	
	• Select Row(s)	Select the device table rows that have focus.
	• Deselect Row(s)	Deselect the device table rows that have focus.
	Expand	
	• Device	Expand the current device
	• All Selected Devices	Expand all selected devices
	• All Devices	Expand all rows for devices and modules.
	Collapse	
	• Device	Collapse the current device
	• All Selected Devices	Collapse all selected devices
	• All Devices	Collapse all rows for devices and modules.
	Insert	
	• Device	Insert a new device row at the selected row and push the following device rows downward. You can use this command to quickly add a device to the device table. If you use this command to insert a device that is behind a router, the device name is colored blue. The blue color means that the MAC address based operations (identify device, set IP address, and set PROFINET name) are not possible and the corresponding Device table cells are disabled.
	Delete	
	• Cell Text	Delete contents of current cell
	• Device	Delete contents of current device row
	• All Selected Devices	Delete contents of all selected devices
	• All Devices	Delete contents for all device rows
	Refresh	
	• Device	Refresh the current device
	• All Selected Devices	Refresh all selected devices
	• All Devices	Refresh all devices

6.1.3 Operations menu

Tool icon	Menu command	Description
?	Scan Network	
	- Scan Entire Network	Scan device network (Page 46)
	- Refresh Status of Selected Devices	Refresh selected devices in the device table (Page 46)
▶	RUN	Put selected CPUs in RUN mode. (Page 54)
■	STOP	Put selected CPUs in STOP mode. (Page 54)
↓	Update	
	- Set IP Address	Update the CPU with the IP Address information for the selected device(s) (Page 51)
	- Set PROFINET Name	Update the CPU with the PROFINET Name for the selected device(s) (Page 53)
	- Program Update	Update the CPU program or HMI operating system and runtime software, for the selected device(s) (Page 56)
	- Firmware Update	Update the CPU firmware with the program update file(s) for the selected device(s) (Page 69)
💡	Identify	Flash the LEDs on CPU devices or HMI screens. (Page 55)Use this feature to identify the physical location of a device.
⇄	Reset	
	- Memory reset	Perform a memory reset on selected devices. (Page 78)
	- Reset to Factory Defaults	Reset selected devices to factory defaults. (Page 79)
	- Format Memory Card	Format memory card in selected devices. (Page 81)
ℹ	Diagnostics	
	- Show CPU Diagnostics	Show diagnostic buffer for a selected CPU. (Page 86)
	- Retrieve Service Data	Retrieve Service Data for selected devices (Page 83)
🕓	Set time	Set time in selected CPUs to your PG/PC time. (Page 85)
→	Backup/Restore	

Tool icon	Menu command	Description
	• **Backup Device to File**	Perform a backup of all selected devices. (Page 74) The SIMATIC Automation Tool saves a backup file for each selected device.
	• **Restore Device from File**	Restore data from backup file(s) to the corresponding device(s). (Page 74)
📄▾	**File Operations**	**Note:** File operations apply only to CPU files.
	• **Upload Data Logs**	Upload selected Data Log files to your PG/PC. (Page 67)
	• **Delete Data Logs**	Delete selected Data Log files. (Page 67)
	• **Upload Recipes**	Upload recipe files from the selected CPUs to your PG/PC. (Page 64)
	• **Add/Replace Recipe**	Add/Replace recipe files from the PG/PC to the selected CPUs. (Page 64) This operation adds recipes if they don't exist and replaces recipes if they do exist.
	• **Delete Recipe**	Delete selected recipes from the corresponding CPUs. (Page 64)

6.1.4 Options menu

6.1.4.1 Options menu

The Options menu contains the following menu commands:

Tool icon	Menu command	Description
	Settings	Opens the Settings dialog where you can set default settings for the following categories: • General (Page 98) • Communications (Page 99) • Projects (Page 100) • Firmware Update (Page 101) • Program Update (Page 102) • Service Data (Page 102) • Backup / Restore (Page 103) • Recipes (Page 104) • Data Logs (Page 104) • Event Log (Page 105) • Import/Export (Page 106)
	Start Automation License Manager	Starts the Automation License Manager with which you can license the SIMATIC Automation Tool

6.1.4.2 SIMATIC Automation Tool pathnames

The pathname examples for the "Options > Settings" dialog show pathnames of folders in C:\Users\MyAccount\SIMATIC Automation Tool\, where "MyAccount" represents your user ID.

If you use the browse function from the device table to locate a file such as a firmware update file or program update file, you do not see your user ID as a folder under "Users". Instead you see the folder "My Documents". When using the SIMATIC Automation Tool browse function, browsing to the "My Documents" folder is equivalent to browsing to the folder with your user ID ("MyAccount") name.

From Windows Explorer, the Documents folder under the Libraries folder is also equivalent to the "My Documents" folder and the "MyAccount" folder.

The following sections of the "Options > Settings" menu command dialog provide a default path that you can change:

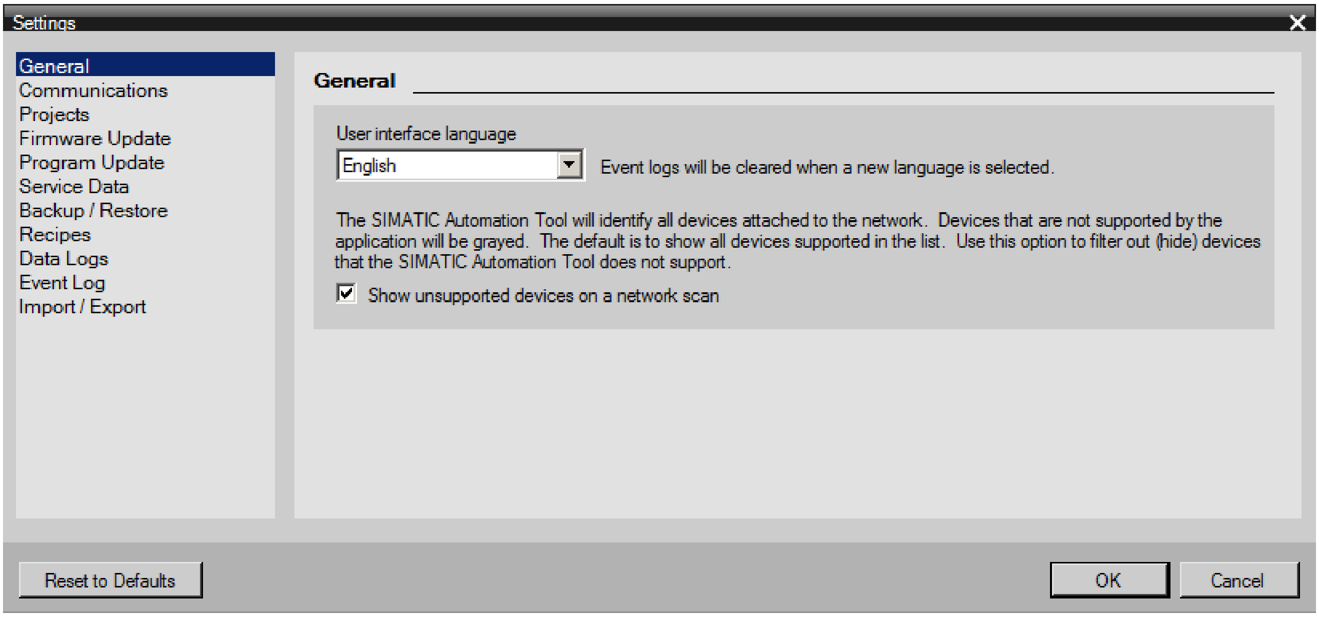
- Projects (Page 100)
- Firmware update (Page 101)
- Program update (Page 102)
- Service Data (Page 102)
- Backup/Restore (Page 103)
- Recipes (Page 104)
- Data Logs (Page 104)
- Event Log (Page 105)

6.1.4.3 General settings

You can select the user interface language: English, German, French, Spanish, or Italian

Select the check box to show unsupported devices on a network scan (Page 46). The SIMATIC Automation Tool displays unsupported devices as disabled by using gray text in the device table.

If you deselect the check box, the SIMATIC Automation Tool filters out unsupported devices from the device table.



User interface language: English

Event logs will be cleared when a new language is selected.

The SIMATIC Automation Tool will identify all devices attached to the network. Devices that are not supported by the application will be grayed. The default is to show all devices supported in the list. Use this option to filter out (hide) devices that the SIMATIC Automation Tool does not support.

☑ Show unsupported devices on a network scan

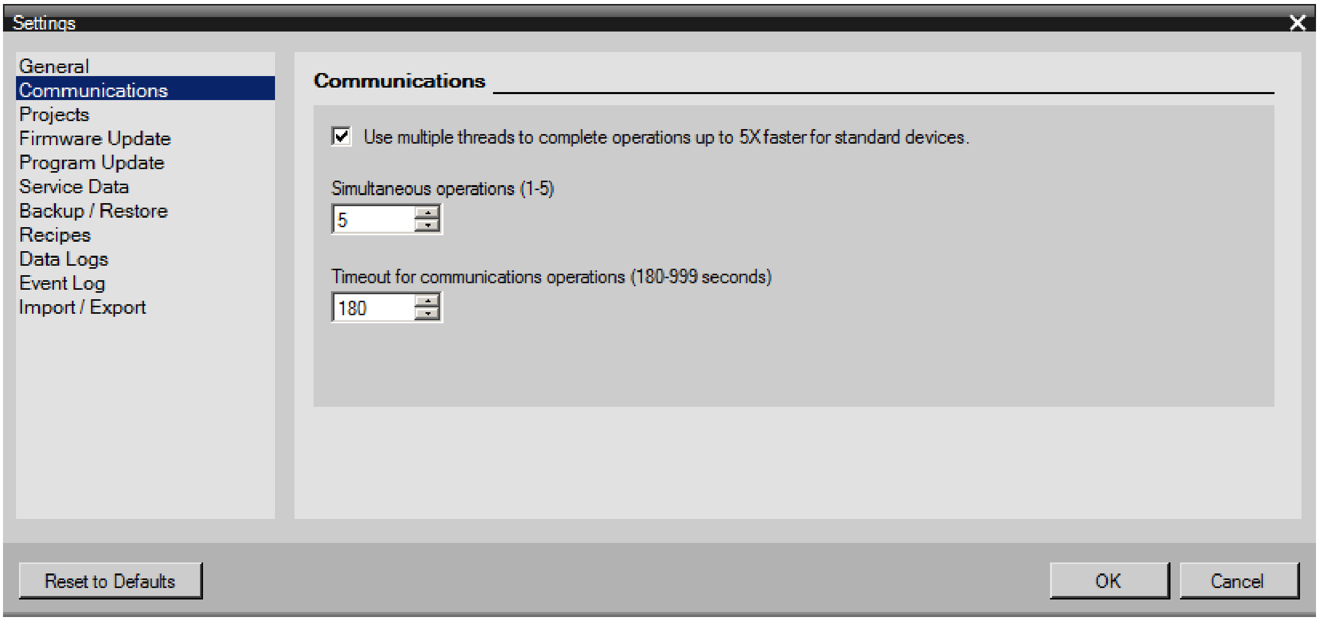
Note

Changing the user interface language change clears the Event Log

When you change the user interface language, the SIMATIC Automation Tool clears the Event Log is cleared.

6.1.4.4 Communications settings

You use the Communications options to set options related to multi-threading.



Communications

☑ Use multiple threads to complete operations up to 5X faster for standard devices.

Simultaneous operations (1-5): 5

Timeout for communications operations (180-999 seconds): 180

Using multiple threads for operations

If your network has a star topology where each CPU has a direct connection to the PG/PC through an Ethernet switch, then you can safely use the multiple threads option.

If your network has a chain topology, disable this option to prevent one CPU from disrupting the communication to other devices. A chain topology, for example, would be chain connections from the PG/PC to CPU 1 to CPU 2 to CPU 3 to others.

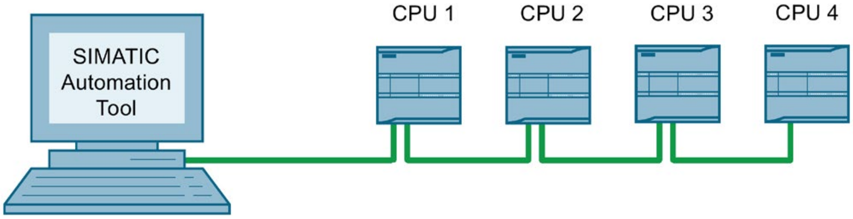


Figure 6-1 Example: Chain topology

With multi-threading, a thread that causes CPU 1 to restart disrupts an operation that is in progress for CPU 2, or any other CPUs in the chain. Note that a chain topology might also be implemented with CM or CP modules.

Simultaneous operations

SIMATIC Automation Tool performance might be increased by allowing operations on multiple devices to occur simultaneously on multiple threads.

Note

Communication problems with the SIMATIC Automation Tool

For example, you send an operation command to multiple devices, but a device does not complete the operation and a communication error displayed for that CPU. However, other devices are communicating and executing the operation as expected. If you have this problem, then reduce the number of simultaneous (threads/connections). Close and restart the SIMATIC Automation Tool, then try the group operation again.

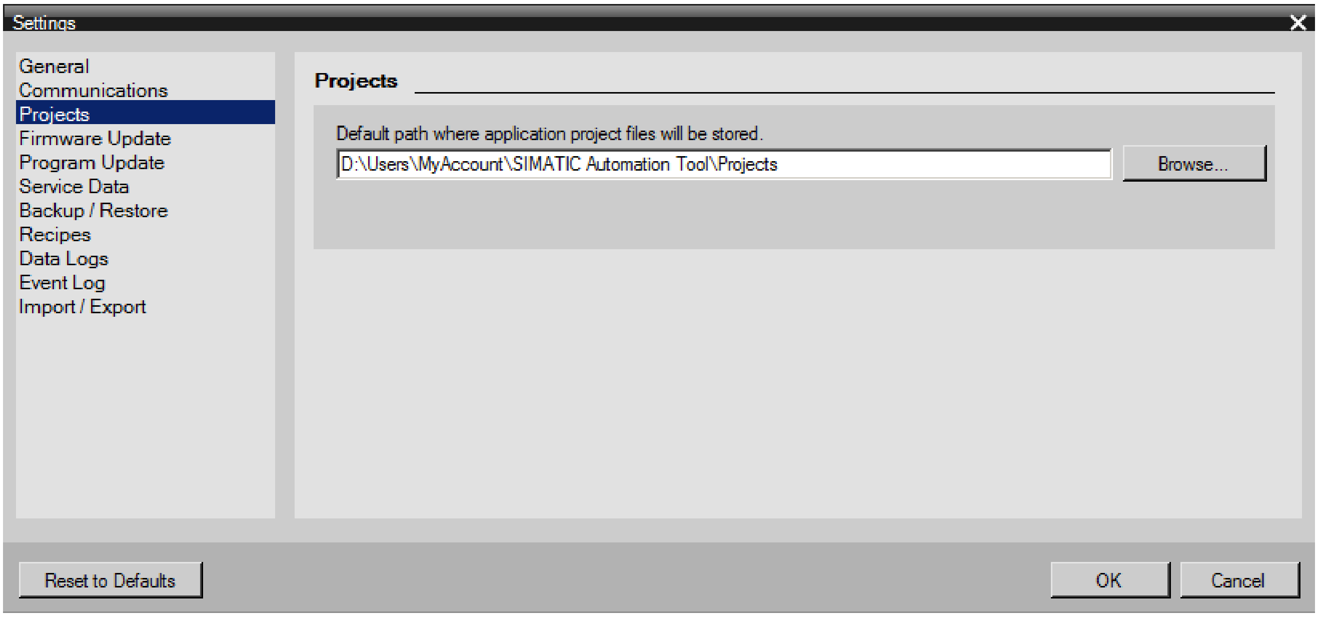
Timeout for communications operations

If you send an operation command to a device and the connection has a very slow data transfer rate, then you might get a communication timeout error. If you have this problem, then increase the timeout for communications operations.

6.1.4.5 Projects settings

You can accept the default path to save SIMATIC Automation Tool project data (Page 89) or assign a new path.

Your path might have a different drive letter and "MyAccount" represents the login name of the current user (Page 97).



Default path where application project files will be stored.

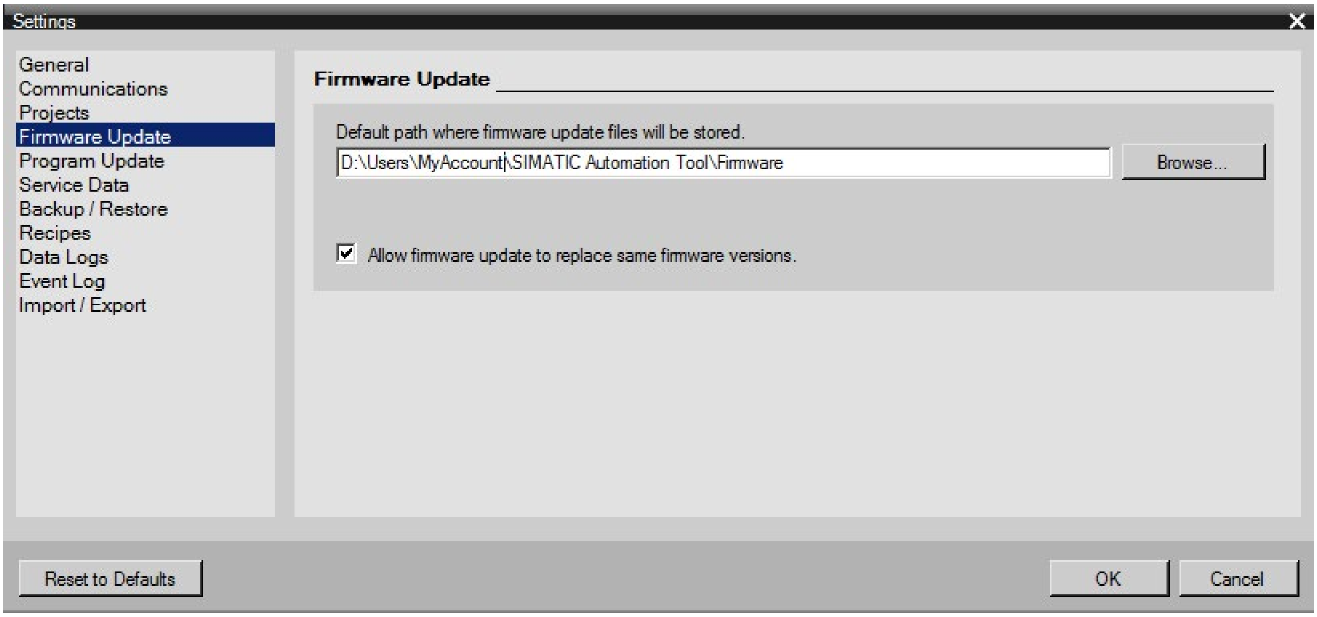
D:\Users\MyAccount\SIMATIC Automation Tool\Projects

6.1.4.6 Firmware update settings

You can accept the default path to firmware update files (Page 69) or assign a different path.

Your path might have a different drive letter and "MyAccount" represents the login name of the current user (Page 97).

Click the check box to allow or disallow a firmware update with the same firmware version. Disallowing the replacement of an identical firmware version saves processing time by preventing unnecessary operations.



Default path where firmware update files will be stored.

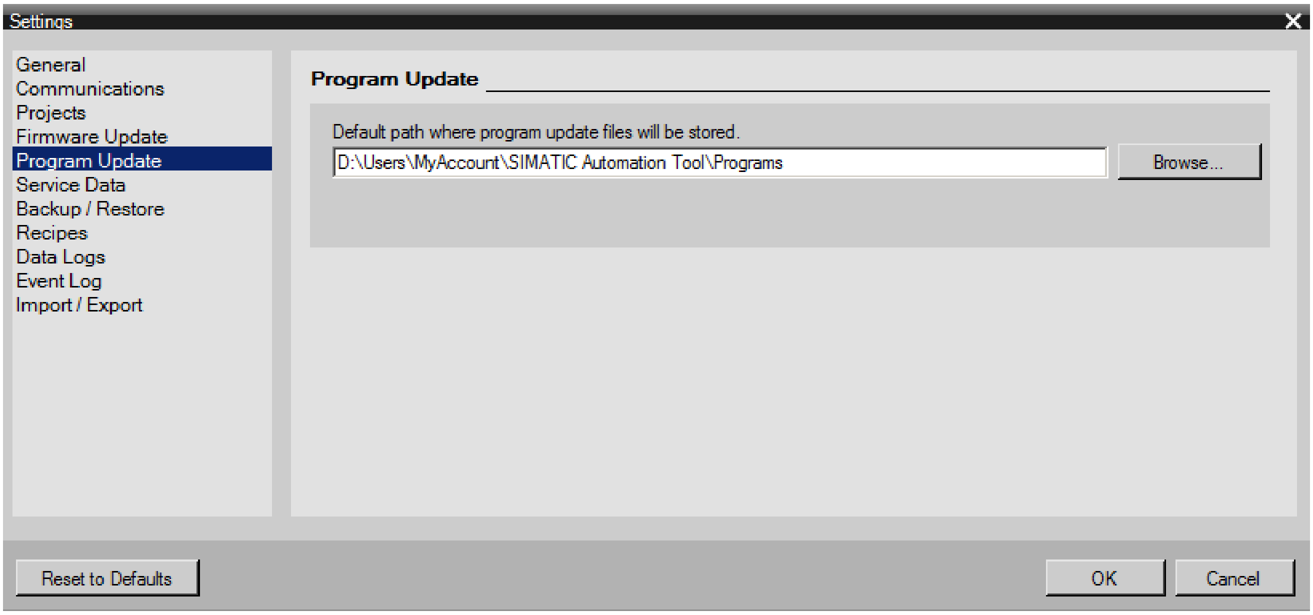
D:\Users\MyAccount\SIMATIC Automation Tool\Firmware

☑ Allow firmware update to replace same firmware versions.

6.1.4.7 Program update settings

You can accept the default path to program files (Page 56) or assign a different path.

Your path might have a different drive letter and "MyAccount" represents the login name of the current user (Page 97).



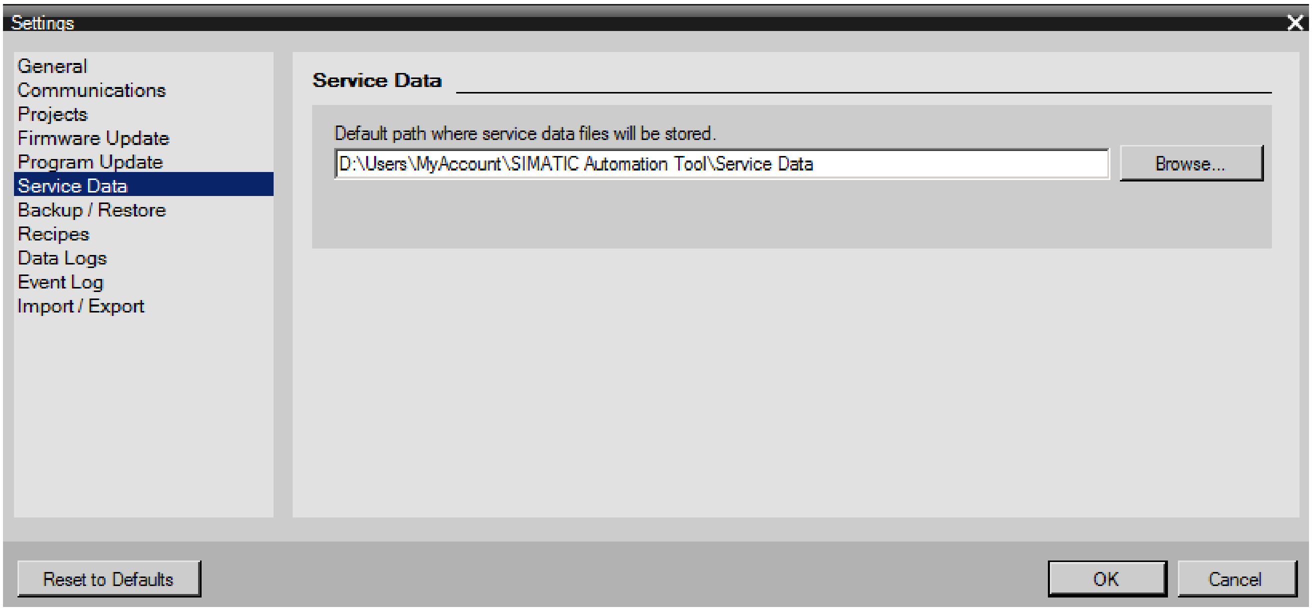
Default path where program update files will be stored.

D:\Users\MyAccount\SIMATIC Automation Tool\Programs

6.1.4.8 Service Data settings

You can accept the default path to Service Data files (Page 83) or assign a different path.

Your path may have a different drive letter and "MyAccount" represents the login name of the current user (Page 97).



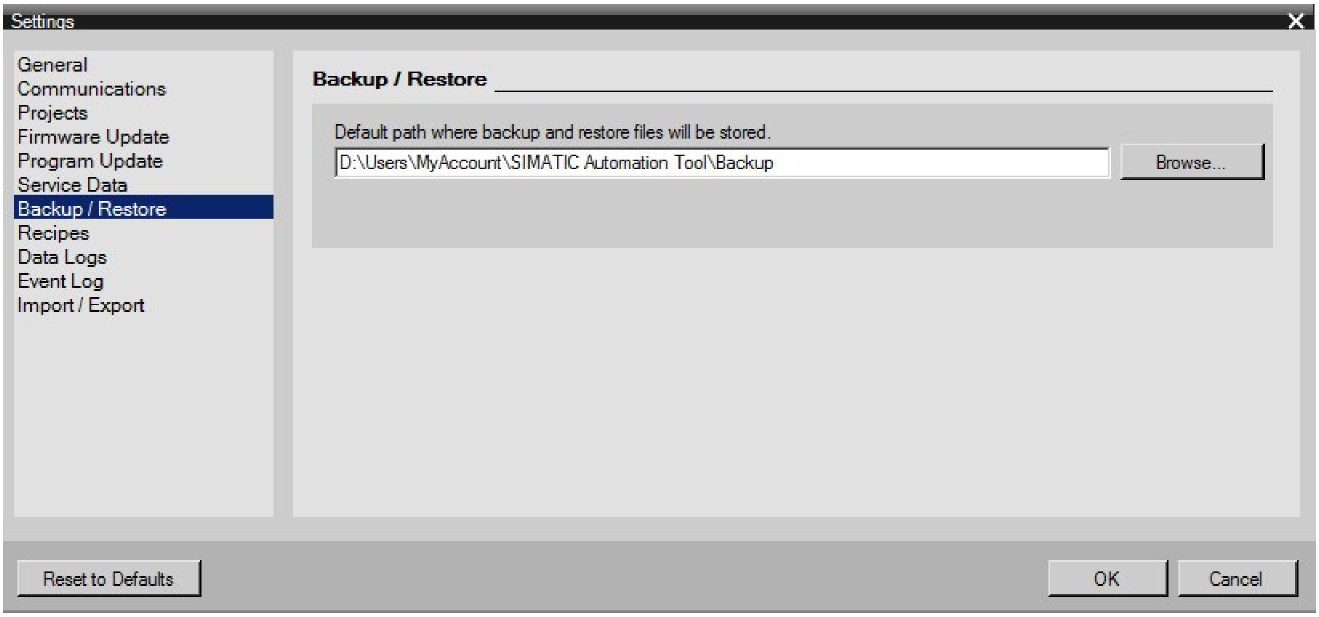
Default path where service data files will be stored.

D:\Users\MyAccount\SIMATIC Automation Tool\Service Data

6.1.4.9 Backup/Restore settings

You can accept the default path to Backup and Restore files (Page 74) or assign a different path.

Your path may have a different drive letter and "MyAccount" represents the login name of the current user (Page 97).



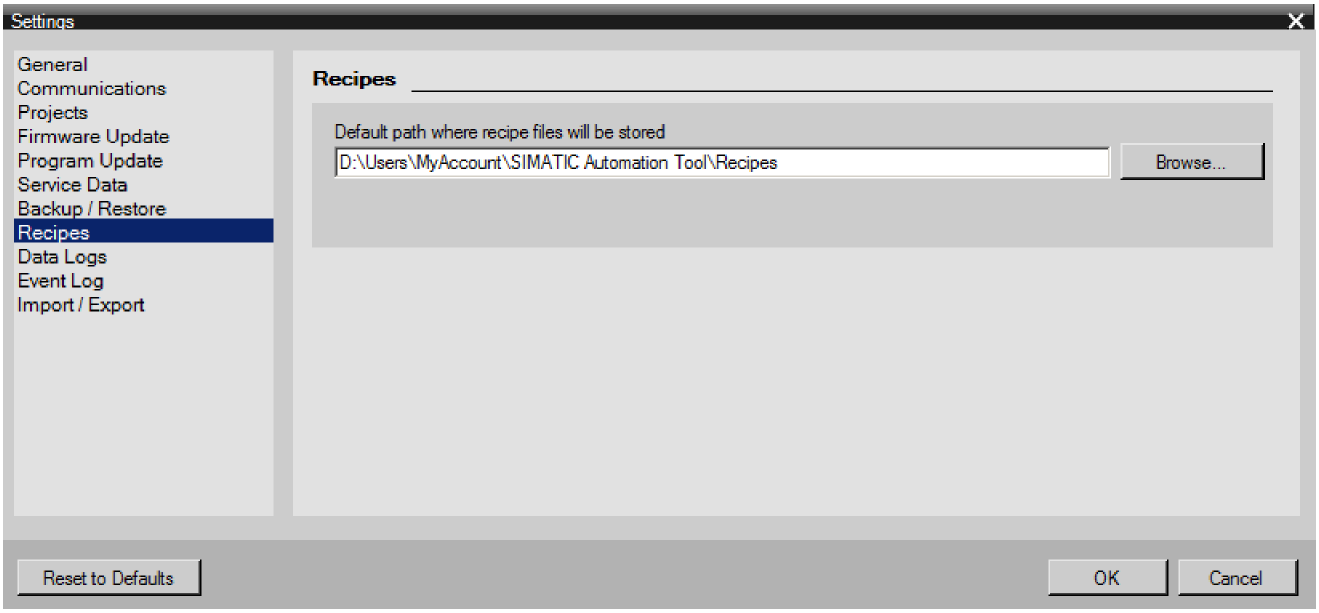
Default path where backup and restore files will be stored.

D:\Users\MyAccount\SIMATIC Automation Tool\Backup

6.1.4.10 Recipes settings

You can accept the default path to recipe files (Page 64) or assign a different path.

Your path may have a different drive letter and "MyAccount" represents the login name of the current user (Page 97).



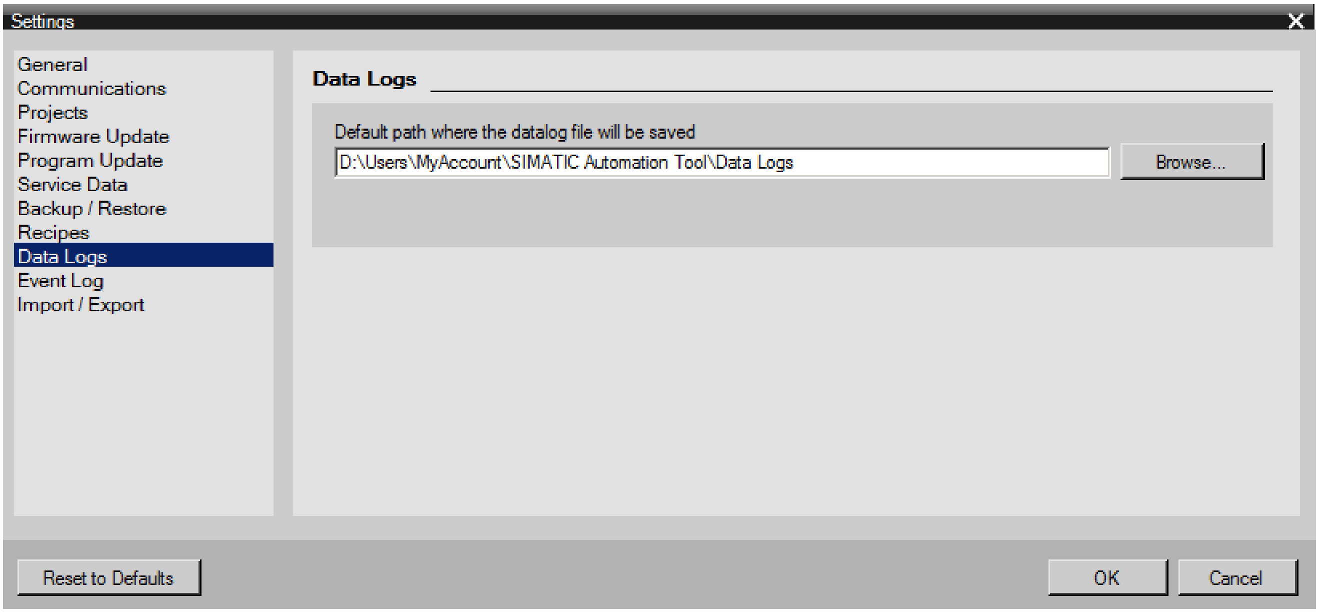
Default path where recipe files will be stored

D:\Users\MyAccount\SIMATIC Automation Tool\Recipes

6.1.4.11 Data Logs settings

You can accept the default path to Data Log files (Page 67) or assign a different path.

Your path may have a different drive letter and "MyAccount" represents the login name of the current user (Page 97).



Default path where the datalog file will be saved

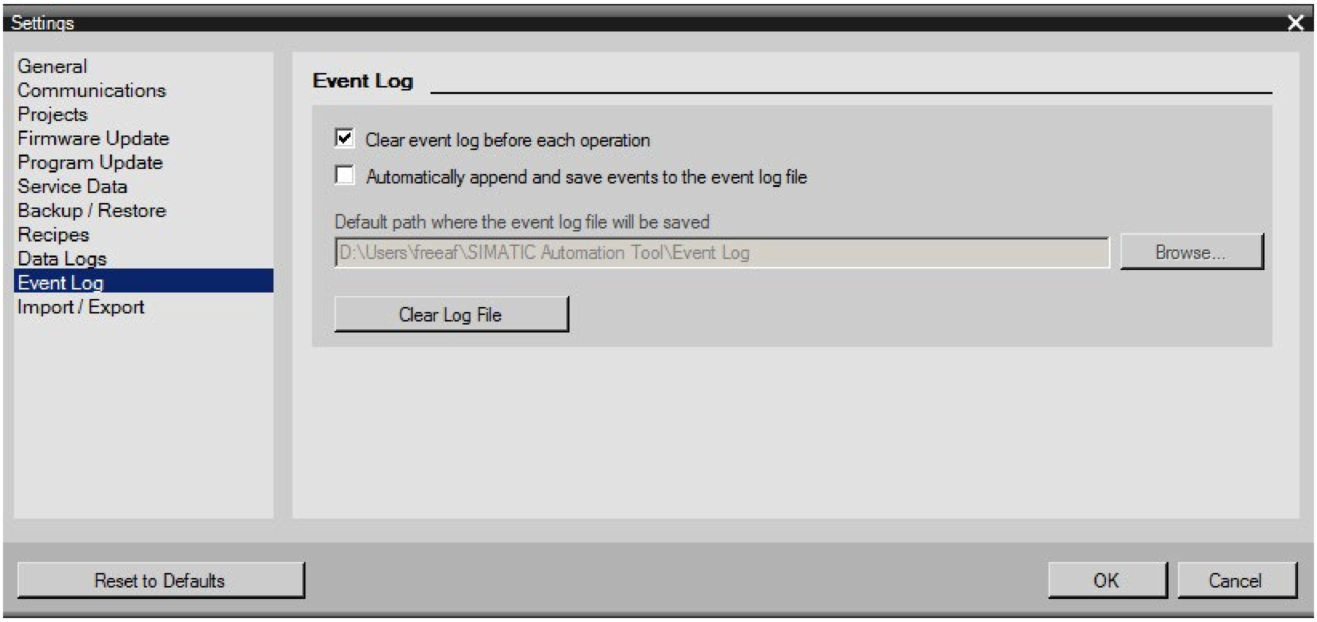
D:\Users\MyAccount\SIMATIC Automation Tool\Data Logs

6.1.4.12 Event Log settings

By default, the SIMATIC Automation Tool clears the Event Log (Page 40) at the start of each device operation. You can deselect "Clear log before each operation" if you want to disable the default.

When you select the "Automatically save log file", then you can accept the default path or assign a different path.
Your path may have a different drive letter and "MyAccount" represents the login name of the current user (Page 97). The SIMATIC Automation Tool then saves each message in the Event Log window to the file "EventLogFile.csv". When you close and re-open the SIMATIC Automation Tool, logging automatically resumes in the Event Log file.

You can clear the content of the Event Log file by clicking the "Clear Log" button. This clears the contents of the file, but does not delete it.



Event Log

☑ Clear event log before each operation

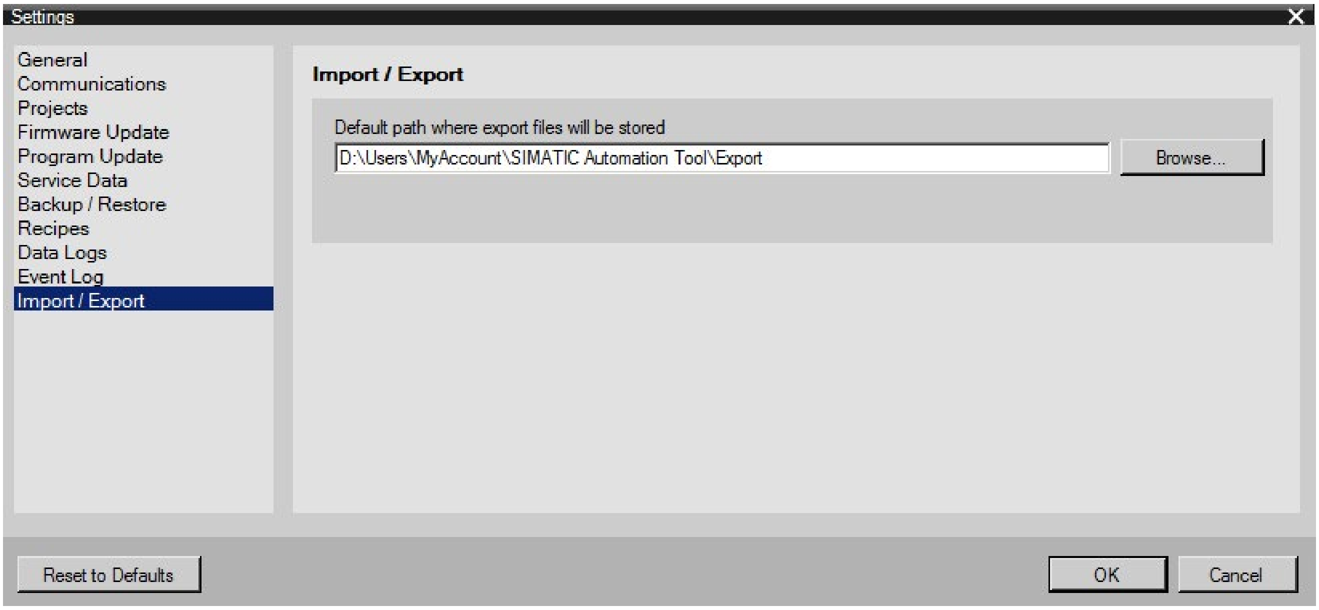
□ Automatically append and save events to the event log file

Default path where the event log file will be saved: D:\Users\freeaf\SIMATIC Automation Tool\Event Log

6.1.4.13 Import / Export settings

You can accept the default path for storage of import / export files (Page 90) or assign a different path.

Your path might have a different drive letter and "MyAccount" represents the login name of the current user (Page 97).



Default path where export files will be stored

D:\Users\MyAccount\SIMATIC Automation Tool\Export

6.1.5 Tools menu

Tools menu

The Tools menu is also accessible from the ⚙ toolbar icon.

Tool icon	Menu command	Description
	Check for Firmware Updates	Check for firmware updates for a single selected device. The SIMATIC Automation Tool displays the Web page with firmware update files for the selected device.
	Preload Firmware Update Files	Look for firmware update files for the selected devices The SIMATIC Automation Tool then preloads firmware update files in the "New Firmware Version" drop-down lists for the selected devices.

6.1.6 Help menu

Tool icon	Menu command	Description
	View user guide	Open the SIMATIC Automation Tool user guide.
	About SIMATIC Automation Tool	Displays the About dialog that contains: • Product name: The text "(No license)" next to the product name indicates that you do not have a license, which limits the functionality. • Version • Check for updates button, which allows you to find SIMATIC Automation Tool updates.

6.2 Toolbar icons

Tool icon	Description
	New: Create a new SIMATIC Automation Tool project file with the ".sat" file name extension.
	Open: Display an "Open" dialog that can browse to a folder, select a project file, and provide a password to open the encrypted project file.
💾	Save the opened project data to a file. If no filename and password are assigned, then the "Save As" dialog is displayed.
✂	Cut the selected data and copy the data to the clipboard. Clipboard data are compatible with Excel so data can be shared between the two applications.
	Copy the selected data to the clipboard.
	Paste the data contained in the clipboard to the selected field(s).
?	Scan the selected network, with the following options: • Scan the selected network interface for accessible CPUs and modules. • Refresh the status of all devices in the device table
▶	RUN: Put selected CPUs in RUN mode.
■	STOP: Put selected CPUs in STOP mode.
	Update device(s) with data from the SIMATIC Automation Tool from one of the following choices: • Set IP address • Set PROFINET name • Program update • Firmware update You must select the corresponding device table tabs to enter the data to update.
💡	Identify devices by flashing device LEDs or HMI screens on selected devices. Use this feature to identify the physical location of a device.
	Reset functions for selected devices: • Memory reset • Reset to factory defaults • Format memory card
ℹ	Access diagnostic information: • Read a CPU diagnostic buffer • Retrieve Service Data from selected devices
	Set time: Set the system time in selected CPUs to current PG/PC time.
	Backup and restore functions: • Create Backup file(s) for selected CPUs and HMI devices. • Restore selected device(s) from backup file(s)

Tool icon	Description
	File operations: • Upload or delete Data Logs • Upload, add, replace, or delete recipes Delete operations require that the CPU be in STOP mode.
	Check for firmware updates or preload firmware update files
D-Link DUB-E100 USB 2.0 Fast Ethernet Adapter.TCPIP.1	Network interface drop-down list: Select the Ethernet network interface that is connected to the industrial control network.

6.3 Shortcut keys

The SIMATIC Automation Tool supports the following shortcut keys for navigating in the device table and for some of the menu commands (Page 93).

CTRL+PgUp	Switches between tabs, from left to right
CTRL+PgDn	Switches between tabs, from right to left
CTRL+A	Selects the entire table
CTRL+C	Copies the selection to the clipboard
CTRL+O	Displays the project open dialog to open a new project file
CTRL+S	Displays the Save As dialog
CTRL+V	Pastes the contents of the clipboard at the insertion point and replaces any selection
CTRL+X	Cuts the selected cells
CTRL+Z	Undo the last edit or delete action
ARROW KEYS	Move one cell up, down, left or right
SHIFT+ARROW KEYS	Extends the selection
DELETE	Removes the contents of the active cell
ENTER	Completes cell editing and validates data
ESC	Cancels cell editing restoring the cell to original value
HOME	Moves to the beginning of a row
CTRL+HOME	Moves to the beginning of the table
END	Moves to the end of a row
CTRL+END	Moves to the end of the table
PAGE DOWN	Moves one screen down in the table
PAGE UP	Moves one screen up in the table
SPACEBAR	Selects or clears the rows check box, or multiple rows, if selected
TAB	Moves one cell to the right
Alt+F4	Exit

6.4 Safety program definition

A safety program is a program you create in STEP 7 that includes safety blocks. If you have installed STEP 7 Safety Advanced, then whenever you add a Fail-Safe CPU to your STEP 7 project, the Program Blocks folder automatically includes the safety blocks. When you download this program to a Fail-Safe CPU, it is a safety program.

SIMATIC Automation Tool API for .NET framework 7

7.1 Introduction to the API

The SIMATIC Automation Tool API allows you to create custom applications based on the functionality available in the SIMATIC Automation Tool application. You can optimize a custom application to combine operations and support user workflows specific to your industrial automation network.

7.2 API software license and version compatibility

Software license required for V3.0 and later versions

You must have a valid SIMATIC Automation Tool license to use the API (Application Programming Interface).

When you install the SIMATIC Automation Tool, provide a license, and agree to all the license terms, you have access to the API for your own programming. You do not have permission to copy or redistribute any part of the SIMATIC Automation Tool including any API .exe or .dll files.

To distribute your custom user interface application to a third party, the third party must also have a valid license for the SIMATIC Automation Tool V3.1

Compatibility with previous versions

The V3.1 API is not compatible with previous versions. You must rewrite programs written with versions of the API prior to V3.1.

See also

Network example (Page 237)

7.3 Designing a user interface application for Fail-Safe devices and safety-relevant operations

7.3.1 API support for safety-relevant operations and Fail-Safe devices

The SIMATIC Automation Tool V3.1 API supports safety-relevant operations that were not possible with prior versions of the SIMATIC Automation Tool:

- Program update
- Restore device from backup file
- Reset to factory defaults
- Format memory card

Note

The "*SIMATIC Safety - Configuring and Programming*" manual contains a warning identified as "S078". This warning states the following, "The use of tools for the automation/operation of user interfaces (e.g. TIA Portal, Web server) which allow access protection for the F-CPU to be bypassed (e.g. saving or automatic entry of an F-CPU password or Web server password), is not permitted."

This warning does not apply to the SIMATIC Automation Tool. The SIMATIC Automation Tool and the SIMATIC Automation Tool API work with F-CPUs and can store CPU passwords for F-CPUs.

Safety features that the API provides

TÜV SÜD has certified the SIMATIC Automation Tool V3.1 and the associated API.

The SIMATIC Automation Tool API uses diverse and redundant techniques. The API thus helps protect user application program code from performing potentially perform unsafe operations. The API provides the following features:

- Independent connection and legitimization process for each safety-relevant operation
- Identity checks for safety-relevant operations to Fail-Safe devices
- Identification of safety programs
- Required use of safety F-CPU password for any safety-relevant operation to a password-protected F-CPU
- Use of 32-bit CRC checksums to compare Fail-Safe device online and offline representations
- Use of hamming codes (Page 122) to indicate TRUE and FALSE states
- Comparison of F-signatures after the Program Update and Restore from Backup operations to verify that the operation completed successfully

7.3.2 User interface programming guidelines for safety-relevant operations

⚠ WARNING

Protect safety-relevant operations as much as possible

Fail-Safe CPUs together with Fail-Safe I/O and safety programs provide the capability for a high degree of operational safety.

When you use the SIMATIC Automation Tool API, ensure that safety-relevant operations are as safe as possible. Siemens assumes no liability for user interface applications developed with the SIMATIC Automation Tool API. The software developer assumes all liability.

Failure to follow adequate programming practices can result in death or personal injury when the user operates your user interface application.

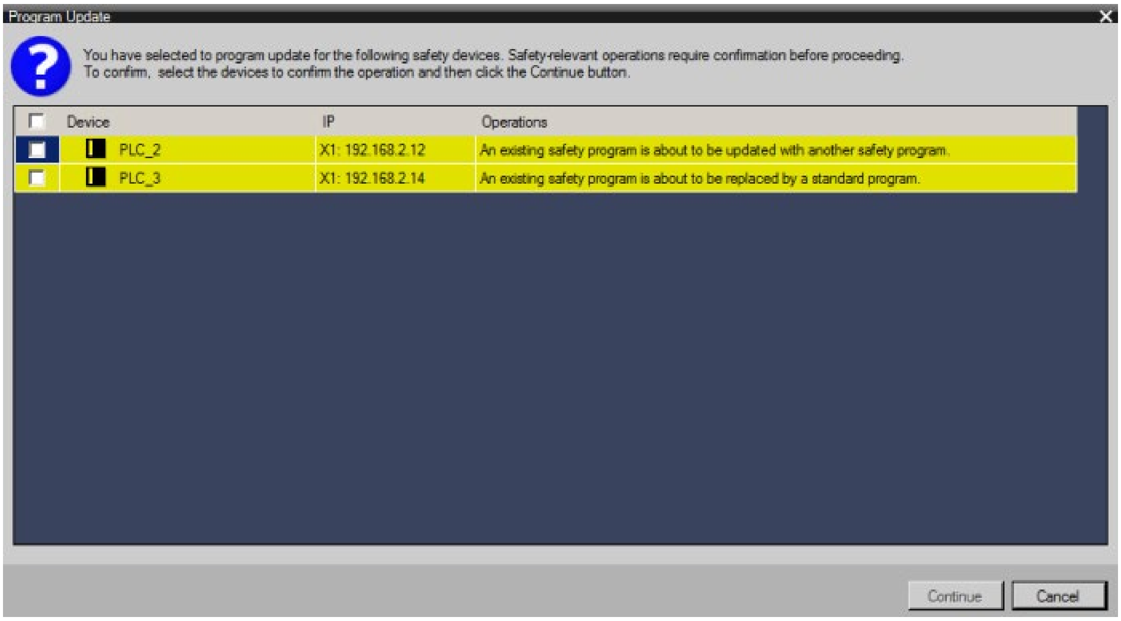
Identifying and protecting safety-relevant operations

The SIMATIC Automation Tool V3.1 presents a confirmation dialog for safety-relevant operations. The user must manually select each Fail-Safe device for the operation and then click "Continue", before processing can begin.

As you develop your user interface application using the API, identify whether an operation is one of the following safety-relevant operations:

- Program update
- Restore device from backup file
- Reset to factory defaults
- Format memory card

For operations that are safety-relevant, provide a confirmation dialog for your users. Use the DetermineConfirmationMessage API method (Page 209) to determine the type of confirmation dialog to display. Providing an additional confirmation dialog protects users from accidentally performing an unintended safety-relevant operation. The following dialog is an example of a confirmation dialog from the SIMATIC Automation Tool for a Program Update operation:



You have selected to program update for the following safety devices. Safety-relevant operations require confirmation before proceeding.
To confirm, select the devices to confirm the operation and then click the Continue button.

□	Device	IP	Operations
□	PLC_2	X1: 192.168.2.12	An existing safety program is about to be updated with another safety program.
□	PLC_3	X1: 192.168.2.14	An existing safety program is about to be replaced by a standard program.

Recommended programming practices

Use the following programming practices to ensure that you protect safety-relevant operations and minimize the chance of unsafe user action:

- Perform all safety-relevant operations on a single thread.
- Require entry of the safety F-CPU password for safety-relevant operations. Verify the entered password against the CPU password. Use asterisks to hide passwords from the display when the user enters passwords.
- Check the return codes of all methods. Ensure that your program logic only proceeds upon successful method returns.
- Include appropriate exception handling in your implementation. The API throws exceptions for critical internal faults that it detects. Be sure that your software handles any exceptions in an appropriate manner.
- For all safety-relevant operations, evaluate whether the operation succeeded. Display a message to the user upon a successful program update. Display an error message upon an unsuccessful program update. Follow the Event Log conventions of the SIMATIC Automation Tool for informing the user about safety-relevant operation outcomes.
- Use hamming codes (Page 122) in your application to implement Boolean states.
- Use coloring (Page 116) in the application to indicate Fail-Safe devices, safety programs, safety F-CPU passwords and other user-entered data.
- Prompt for confirmation for all operating mode changes (RUN/STOP).
- Refresh the user interface after each operation so that the application displays the correct device data.

Program update requirements

For program updates on selected F-CPUs, provide an additional dialog for the user to reselect the Fail-Safe devices and confirm the following operations:

- Updating an existing safety program with another safety program
- Updating an existing safety program with a standard program
- Updating an existing standard program with a safety program
- Updating a CPU with no program with a safety program
- Deleting an existing safety program

Following a safety program update, verify that the F-signature of the program updated in the CPU matches the F-signature of the program update file.

Restore from backup requirements

Before restoring a backup file, evaluate whether the file is a safety program and prompt for user confirmation according to the same requirements for program updates.

Certification

Note

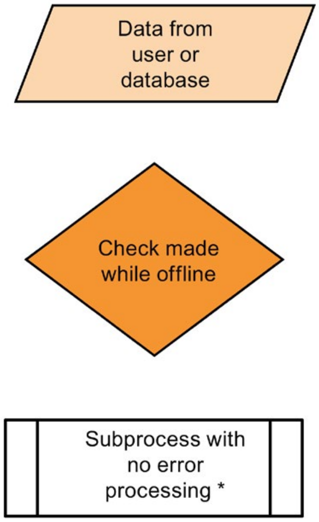
Acquire certification for your user interface application

Siemens strongly suggests using a notified body such as TÜV SÜD to certify the safety of your design and implementation.

7.3.3 Color coding safety fields in your user interface

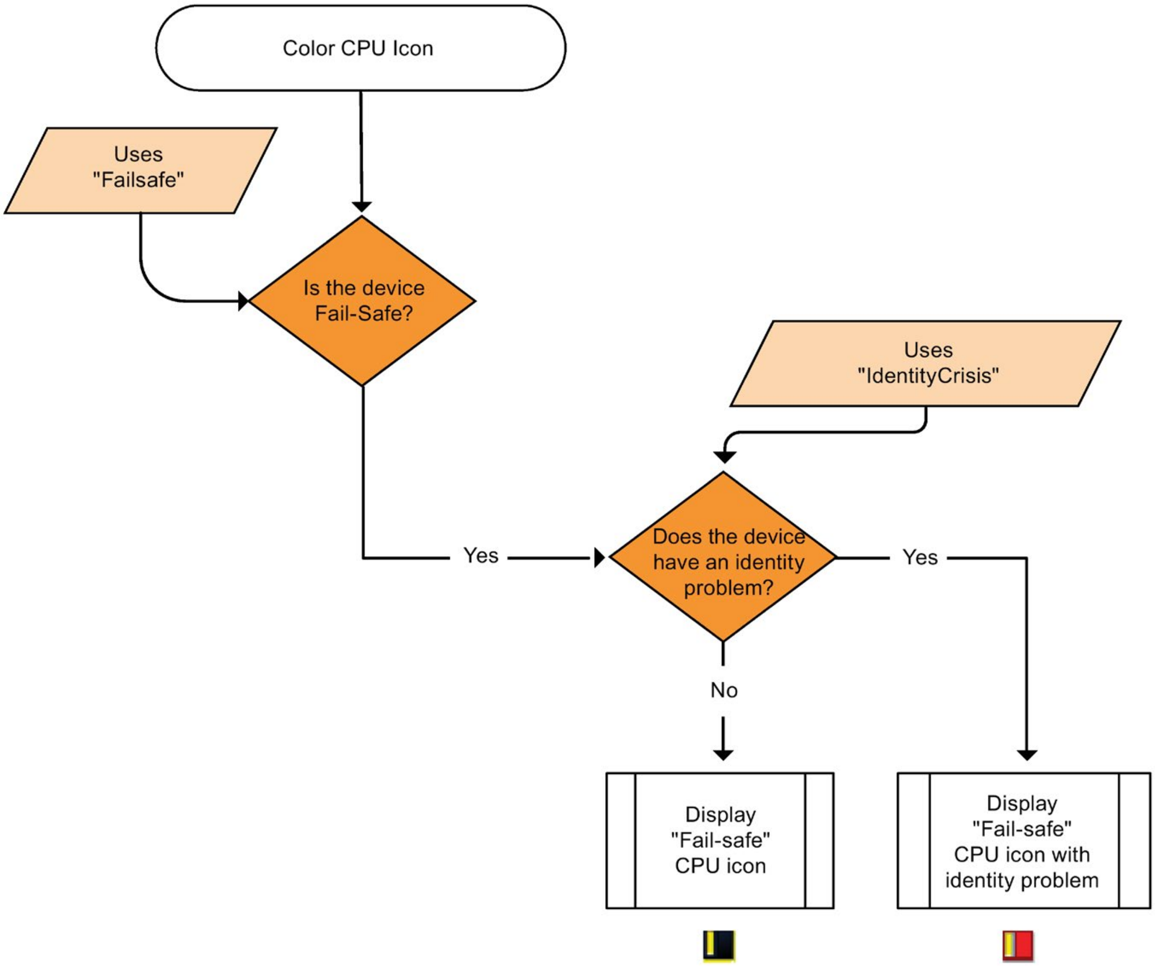
Siemens strongly recommends that you use color coding to give the user a visual indication of anything related to Fail-Safe CPUs and safety programs. The decision trees indicate the logic Siemens uses in color coding various safety-relevant fields in the SIMATIC Automation Tool user interface. Consider adopting an identical or similar approach as you design your application.

Decision tree conventions

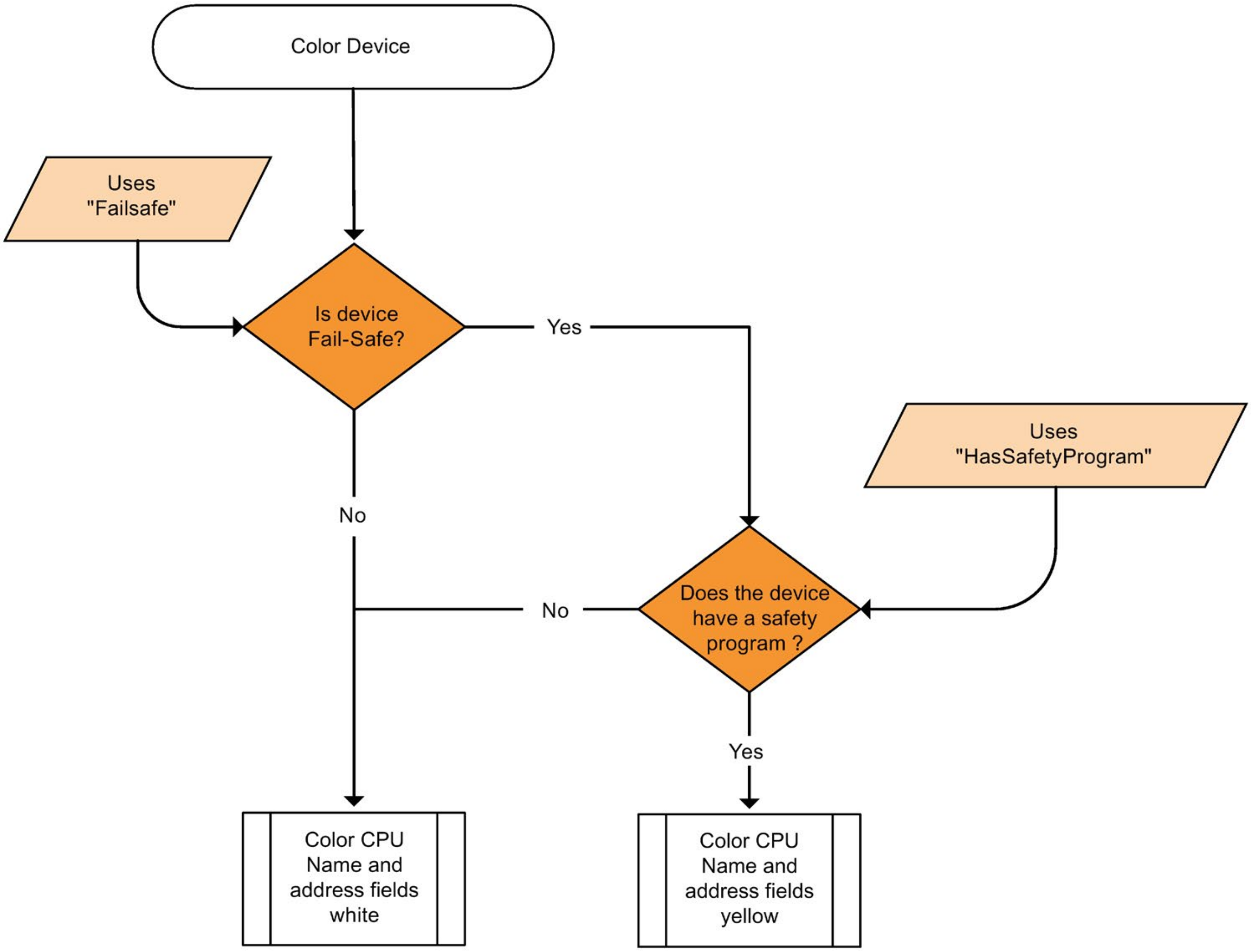

Data from user or database
Check made while offline
Subprocess with no error processing *

* Subprocesses that cannot fail, have no error processing, or represent a situation where the application intentionally ignores errors

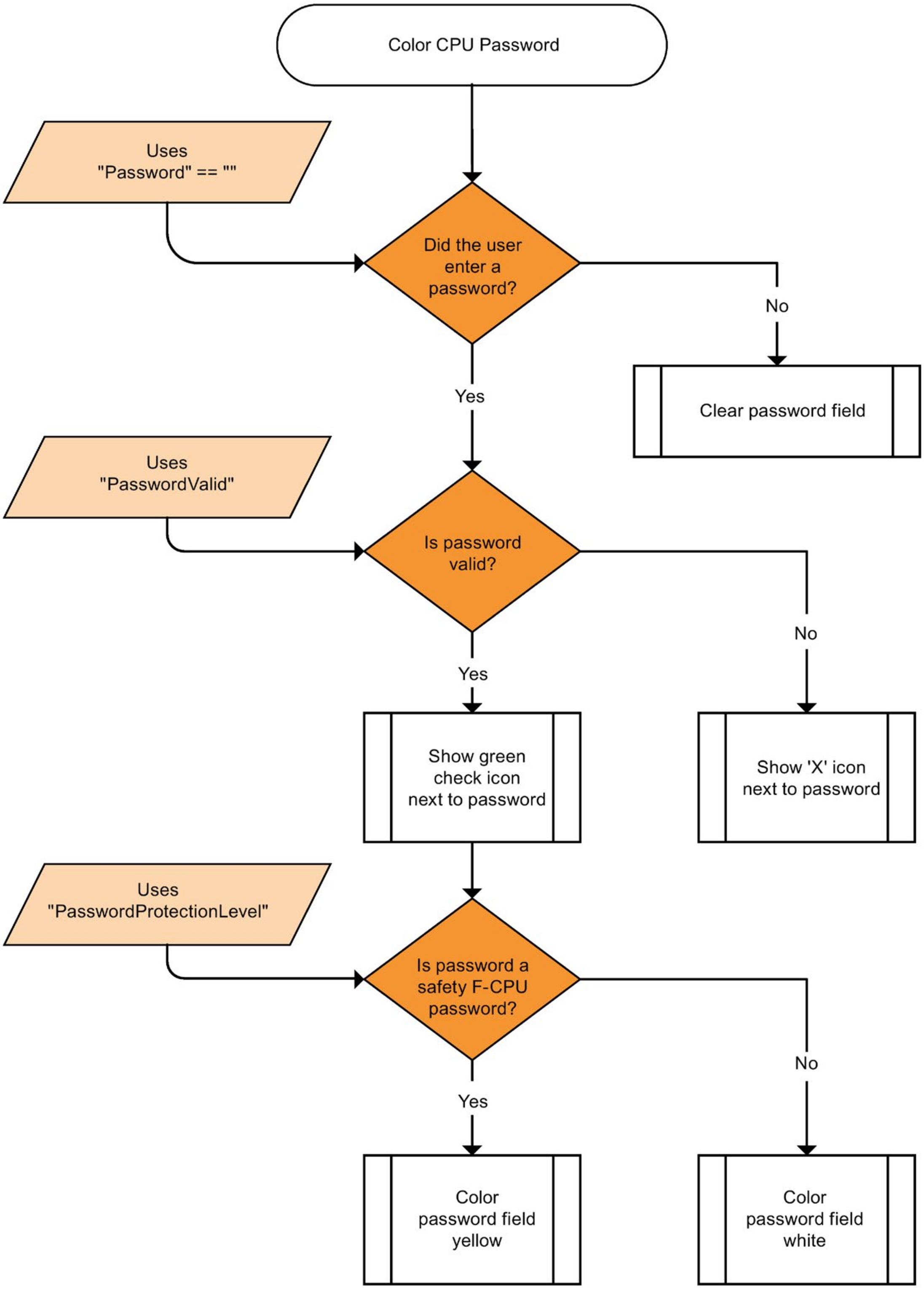
7.3.3.1 Coloring a CPU device icon


Color CPU Icon
Uses "Failsafe"
Is the device Fail-Safe?
Uses "IdentityCrisis"
Yes
Does the device have an identity problem?
Yes
No
Display "Fail-safe" CPU icon
Display "Fail-safe" CPU icon with identity problem

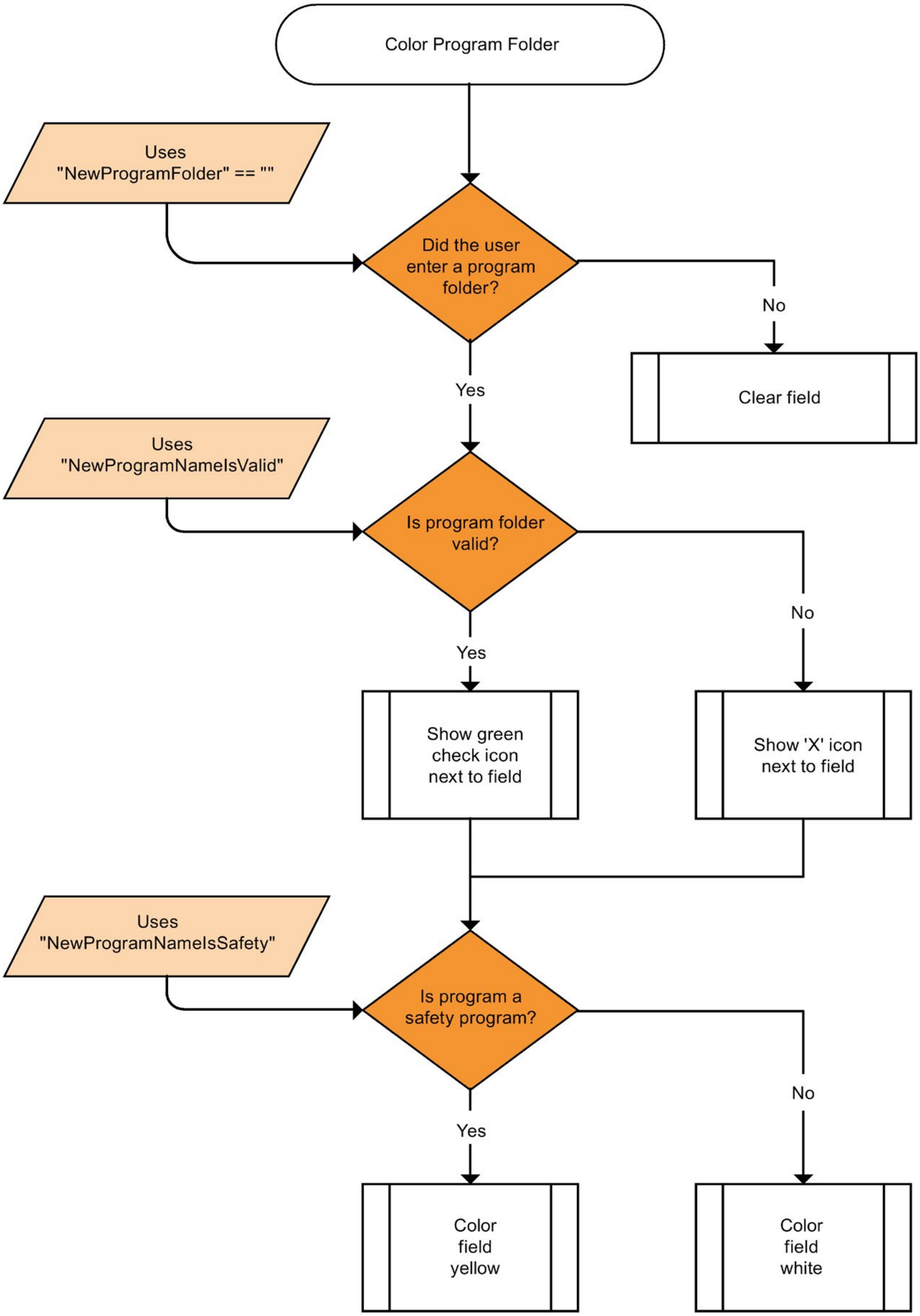
7.3.3.2 Coloring device data


Color Device
Uses "Failsafe"
Is device Fail-Safe?
Yes
No
Uses "HasSafetyProgram"
Does the device have a safety program ?
No
Yes
Color CPU Name and address fields white
Color CPU Name and address fields yellow

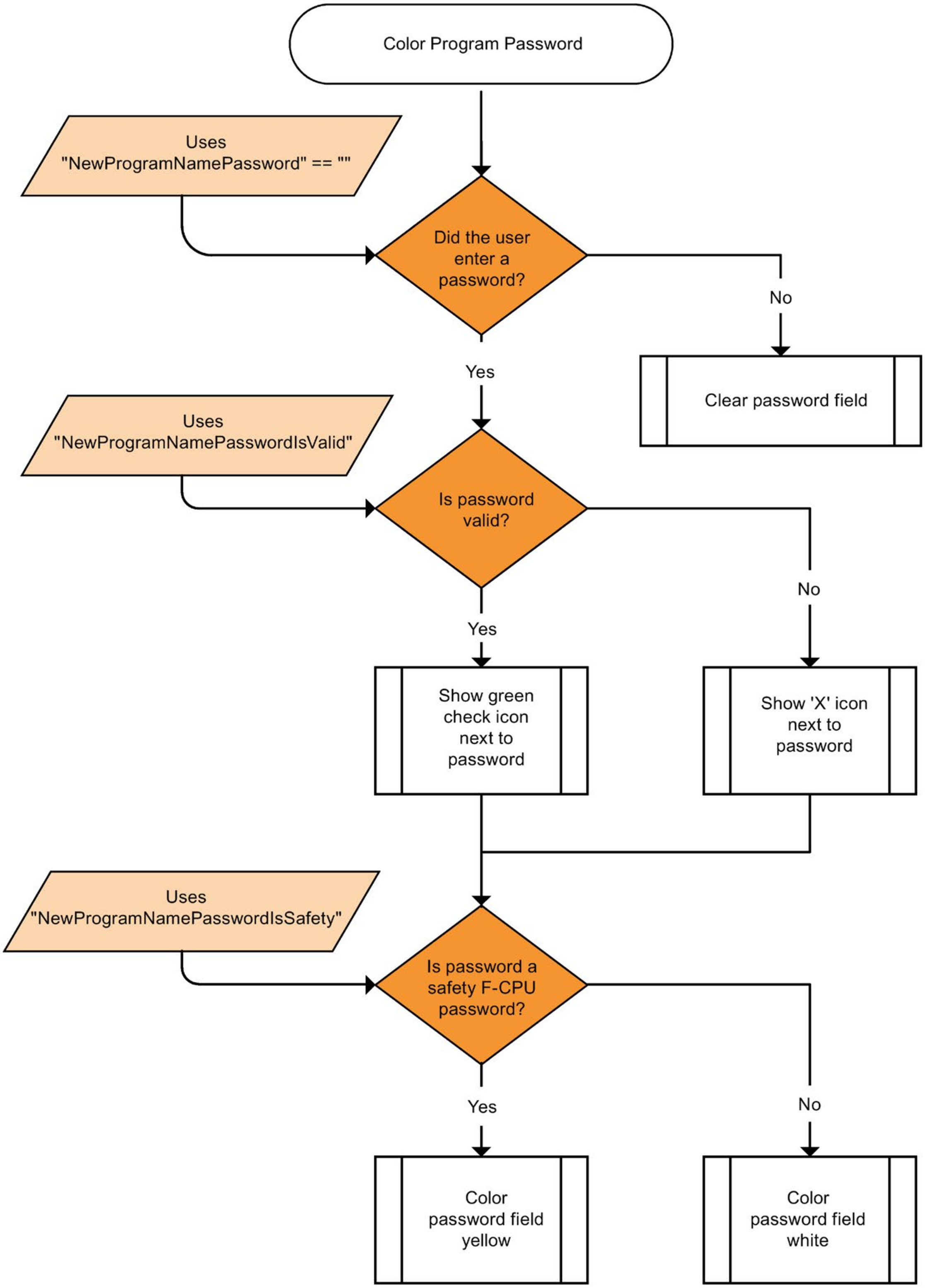
7.3.3.3 Coloring a CPU password


Color CPU Password
Uses
"Password" == ""
Did the user enter a password?
No
Clear password field
Yes
Uses
"PasswordValid"
Is password valid?
No
Yes
Show green check icon next to password
Show 'X' icon next to password
Uses
"PasswordProtectionLevel"
Is password a safety F-CPU password?
No
Yes
Color password field yellow
Color password field white

7.3.3.4 Coloring a program folder


Color Program Folder
Uses "NewProgramFolder" == ""
Did the user enter a program folder?
No
Clear field
Yes
Uses "NewProgramNameIsValid"
Is program folder valid?
No
Yes
Show green check icon next to field
Show 'X' icon next to field
Uses "NewProgramNameIsSafety"
Is program a safety program?
No
Yes
Color field yellow
Color field white

7.3.3.5 Coloring a program password


Color Program Password
Uses "NewProgramNamePassword" == ""
Did the user enter a password?
No
Clear password field
Yes
Uses "NewProgramNamePasswordIsValid"
Is password valid?
No
Yes
Show green check icon next to password
Show 'X' icon next to password
Uses "NewProgramNamePasswordIsSafety"
Is password a safety F-CPU password?
Yes
No
Color password field yellow
Color password field white

7.3.4 Hamming codes

Hamming codes are binary codes. They can detect incidental bit errors. The SIMATIC Automation Tool uses 32-bit hamming codes with a hamming distance of eight. The API uses hamming codes to represent all Boolean values related to safety-relevant operations. You can program your user interface application to use the provided Boolean value states for safety-relevant operations. Because the API implements these states with hamming codes, you can have high confidence in the data integrity of the safety-relevant Boolean states.

7.4 Architectural overview

Networks

The `.NET` class `Network` represents the industrial network as a whole. This class performs functions using a network interface card (NIC) installed on a PG/PC. The `Network` class is used to scan for available interface cards and to select the network interface card.

- Network constructor (Page 134)
- QueryNetworkInterfaceCards method (Page 135)
- SetCurrentNetworkInterface method (Page 136)
- CurrentNetworkInterface property (Page 136)
- ScanNetworkDevices method (Page 137)
- SetCommunicationsTimeout method (Page 138)
- GetCommunicationsTimeout method (Page 139)
- CheckValidLicense method (Page 139)
- GetEmptyCollection method (Page 140)

Devices

The individual devices on the network are represented by interfaces. Each interface class provides properties and methods appropriate for the represented network device. Each hardware device on the network is best represented by one of the following interfaces:

`IProfinetDevice` – Any device directly accessible on the industrial network can be represented by this interface.

`ICPU` – This represents S7 CPUs that are directly connected to the network. Specific functionality is supported for CPUs.

`IHMI` – This represents SIMATIC HMIs that are directly connected to the network. Specific functionality is supported for HMIs.

`IBaseDevice` – This interface is used to represent devices not directly connected to the Ethernet network, but accessible through another device. For example, a PROFIBUS slave station that is connected to a CPU on the network is represented as an `IBaseDevice`.

`IModule` – This interface is used to represent individual I/O modules that are plugged into a CPU, PROFINET device, or PROFIBUS station.

`IHardware` – This is the base class for all other interfaces. This interface provides access to properties that are common for all hardware items recognized on the network.

The interfaces are grouped into collections that represent groups of devices. Collections are provided to support iteration, filtering, and searching.

`IProfinetDeviceCollection` – A collection of all devices directly accessible on the network.

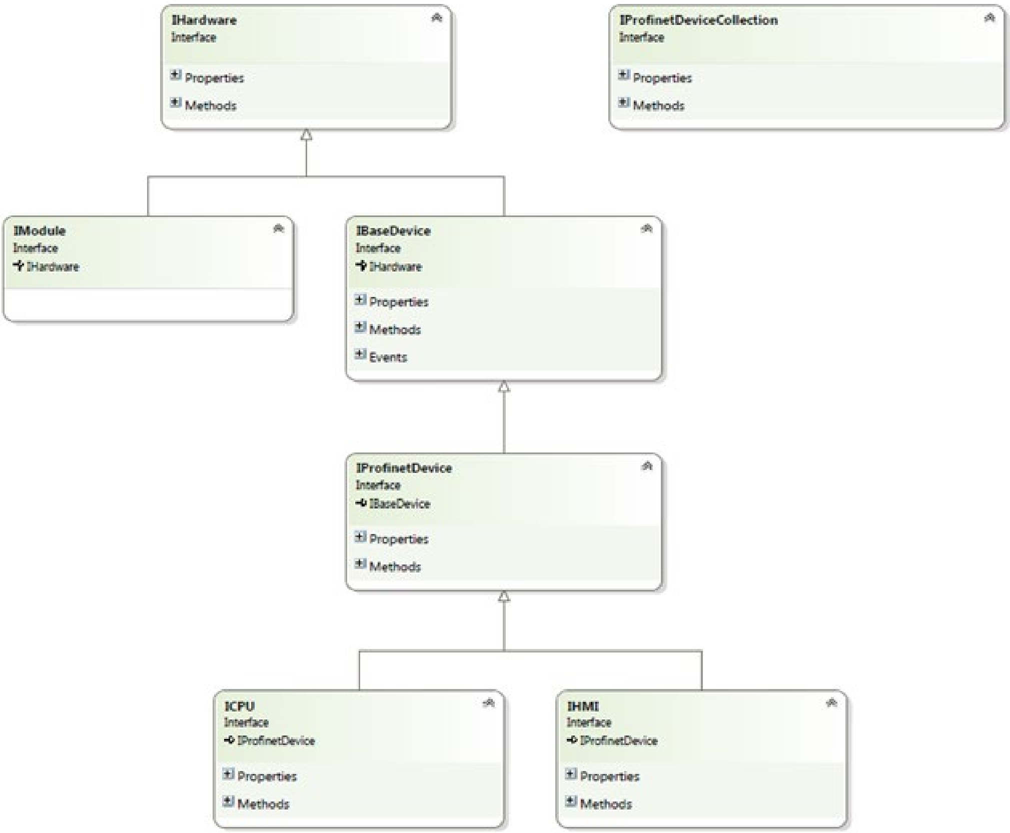
`IModuleCollection` – A collection that may represent modules plugged into a CPU or IM.

`IHardwareCollection` – This collection represents a CPU and all its modules.

Device classes, interfaces, and methods:

- IProfinetDeviceCollection class (Page 141)
- IProfinetDevice interface (Page 153)
- ICPU interface (Page 172)
- IHMI interface (Page 215)

The following class diagram shows the inheritance relationship between these interface classes:



Note

See the example (Page 237) industrial network and the SIMATIC Automation Tool API classes that are used to represent each network component.

See also

FirmwareUpdate method (Page 158)

Backup method (ICPU interface) (Page 188)

SetOperatingState method (Page 200)

Modules property and IModuleCollection class (Page 170)

Restore method (ICPU interface) (Page 187)

SetIP method (Page 162)

Identify method (Page 160)

RefreshStatus method (Page 157)

Reset method (Page 161)

SetProfinetName method (Page 164)

GetCurrentDateTime method (Page 194)

GetDiagnosticsBuffer method (Page 195)

MemoryReset method (Page 196)

ProgramUpdate method (Page 180)

ResetToFactoryDefaults method (Page 197)

SetCurrentDateTime method (Page 201)

UploadServiceData method (Page 206)

IModule interface (Page 171)

DeleteDataLog method (Page 190)

DeleteRecipe method (Page 192)

UploadDataLog method (Page 202)

UploadRecipe method (Page 204)

Backup method (IHMI interface) (Page 218)

ProgramUpdate method (IHMI interface) (Page 219)

Restore method (IHMI interface) (Page 221)

DownloadRecipe method (Page 189)

RemoteInterfaces properties (Page 211)

IProfinetDevice events (Page 167)

7.5 Referencing the API in a user interface application

Siemens delivers the API with several DLLs, executables and source files:

- `AutomationToolAPI.dll`
- `DeviceManagerClient.dll` (HMI)
- `hmitr.dm.client.proxy.dll` (HMI)
- `hmitr.dm.client.stub.exe` (HMI)
- `hmitr.ipc.dll` (HMI)
- AsModels folder and subfolders (Offline object models)

The API was created with Microsoft Visual Studio 2015 SP2 Update 3 using the .NET framework 4.6.2. You can use this API with applications that you create with this version of Visual Studio or later versions. All code examples and screen captures in this document were made with Visual Studio 2015 SP2 Update 3, in the C# programming language.

To include the API in your application, you must add `AutomationTool.dll` as a "reference" in the Visual Studio solution.

In any source file where the API classes are referenced, you must add the following statement referencing the API namespace:

```
using Siemens.Automation.AutomationTool.API;
```

To compile any of the code samples in this document, the correct `using` statement must be present in the same source file (*.cs) as the example code. For simplicity, the individual code examples in this document do not include the using statement.

To use the API at runtime, the correct version of S7 communications must be installed on the PG/PC. The easiest way to ensure you have the correct files is to install the SIMATIC Automation tool on that machine. Once installed, you can place the API dll (AutomationToolAPI.dll) together with the HMI related files listed above in any folder on the PG/PC and use them successfully.

7.6 Common support classes

7.6.1 EncryptedString class

Before describing the operations available through the API, it is important to have an understanding of some common classes that are used in most of the code examples.

The `EncryptedString` class

Many API operations require a legitimized connection to a protected S7 CPU. For these operations, a password is required as one of the parameters to the method. The S7 CPU accepts the password in an encrypted format. To accomplish this, the API provides the `EncryptedString` class.

Constructor	Description
`EncryptedString()`	An empty encrypted string
`EncryptedString(string strText)`	An encrypted string

Property name	Return type	Description
`IsEmpty`	`bool`	True, if there is no password
`IsEncrypted`	`byte`	True if there is an encrypted password

Method name	Return type	Description
`ToString()`	`string`	Hexadecimal string representation of the encrypted password
`Clear()`	`void`	Clears the encrypted password
`GetHash()`	`byte[]`	Password encrypted hash array representation of the password
`WriteToStream(Stream stream)`	`void`	Serialize password from a stream
`ReadFromStream(Stream stream)`	`void`	Deserializes a password from a stream

This class provides a way to encrypt a plain-text password so that you can legitimize a CPU connection. Many of the code examples show a typical usage of this class.

If you wish to encrypt a password to use multiple times in your code, you can instantiate the `EncryptedString`, then pass it as a parameter to multiple calls, as follows:

```
EncryptedString pwd = new EncryptedString("password");

devAsCpu.Selected = true;
devAsCpu.SetPassword(pwd);
```

Note

If a CPU is not password protected, simply pass an empty string to the `EncryptedString` constructor. For example, the following code is successful for a CPU with no protection configured:

```
devAsCpu.SetPassword(new EncryptedString("Password"));
devAsCpu.Selected = true;
Result retVal = devAsCpu.RefreshStatus();
```

The `EncryptedString` object does not store the user-specified plain-text password. However, if your application codes passwords as literal strings you create a security risk.

For example, `new EncryptedString("myPassword")`

The plain-text "`myPassword`" is compiled into the user application, and may be visible to others using .NET reflection.

7.6.2 Result class

The `Result` class encapsulates the logic that determines if a given API action succeeded. Most API actions involve some level of network communications. Many also involve opening a connection to a network device. Such actions are never guaranteed to be successful. The `Result` object returned by an API action should always be inspected for success or failure.

Constructor	Description
`Result()`	Creates successful result with no warnings
`Result(ErrorCode nCode)`	Creates specific error with no warnings

Property name	Return type	Description
Warnings	`ErrorCode[]{get;}`	Returns all warnings in an array of error codes
Error	`ErrorCode {get;}`	Returns error code
HasWarnings	`Bool {get;}`	True when warnings exists
Failed	`Bool {get;}`	True when the result failed
Succeeded	`Bool {get;}`	True when the result is successful Succeeded = not Failed

In many instances, it may be sufficient to know whether a given action was successful. In this case, a check of the `Succeeded` property is all that is required:

```
dev.Selected = true;
Result retVal = dev.RefreshStatus();
if (retVal.Succeeded)
{
    //-------------------------------------
    // Continue operations....
    //-------------------------------------
}
```

In other cases it may be helpful to have more information about the failure. To inspect the specific error, use the Code property, as follows:

```
dev.Selected = true;
Result retVal = dev.RefreshStatus();
if (retVal.Succeeded)
{
    //-------------------------------------
    // Continue operations....
    //-------------------------------------
}
else
{
    //-------------------------------------
    // What happened?
    //-------------------------------------
    switch (retVal.Error)
    {
        case ErrorCode.AccessDenied:
            break;
        case ErrorCode.TooManySessions:
            break;
    }
}
```

See the ErrorCode (Page 229) topic for a list of values.

The `Result` class also provides a language-specific error description. The `GetDescription` method uses a `Language` value as a parameter.
For example, the following code returns the error description in German:

```
String strError = result.GetErrorDescription(Language.German);
```

See the Language (Page 232) enumeration topic for a list of values.

SIMATIC Automation Tool V3.1 has a new warnings feature for cases where you need to be aware of issues that have occurred. For instance, the Refresh that is performed on the device at the end of a Program Update may create warnings that are not directly related to the main calling function. You can access these warning through the `Result` class, as follows.

```
if (retVal.HasWarnings)
{
    foreach (ErrorCode warning in retval.Warnings)
    {
        //-----------------------------------
        // Continue operations....
        //-----------------------------------
    }
}
```

7.6.3 Diagnostic class

A diagnostic item contains diagnostic information for a single event. The diagnostic buffer can be read from a CPU. See the ICPU interface (Page 172) chapter for details.

Constructor	Description
`DiagnosticsItem()`	Creates default diagnostic item

Property Name	Return type	Description
`TimeStamp`	`DateTime {get;}`	Time stamp of the diagnostic event
`State`	`Byte {get;}`	0=Outgoing event; 1=Incoming event
`Description1`	`String {get;}`	Basic description
`Description2`	`String {get;}`	Detailed description

7.6.4 DataChangedEventArgs class

A data changed event contains information about data that has changed within the API. See the IProfinet interface (Page 153) chapter for details.

Constructor	Description
`DataChangedEventArgs(DataChangedType type)`	Creates event of specific type
`DataChangedType type`	Type of data that has changed

Property Name	Return type	Description
`Type`	`Data-ChangedType`	Type of event

Used with the following event handler:

```
public delegate void DataChangedEventHandler(object sender, DataChangedEventArgs e);
```

7.6.5 ProgressChangedEventArgs class

A progress changed event contains information about data that has changed within the API. See the IProfinet interface (Page 153) chapter for details.

Constructor	Description
`ProgressChangedEventArgs(ProgressAction action, int index, int count, uint hardwareID)`	Used to create and default a progress changed event args class
`ProgressAction action`	Type of progress that has occurred.
`int index`	Index of current item being processed
`int count`	Total items to process
`uint hardwareID`	ID of the item being processed

Property Name	Return type	Description
`ID`	`uint {get;}`	ID of item
`Cancel`	`bool {get;}`	Set to true to terminate current operation
`Count`	`int {get;}`	Maximum value
`Index`	`int {get;}`	Current value
`Action`	`ProgressAction {get;}`	Action type of this event

Used with the following event handler:

```
public delegate void ProgressChangedEventHandler(object sender,
ProgressChangedEventArgs e);
```

7.7 Common support interfaces

7.7.1 IRemoteFile interface

IRemoteFile is an interface used to represent files used in datalogs and recipes.

Property name	Return type	Description
Selected	`bool {get; set;}`	The selected state
FileSize	`ulong {get;}`	The size on the file on the CPU
Name	`string {get;set;}`	The file name and extension on the device.

7.7.2 IRemoteFolder interface

`IRemoteFolder` is used to represent folders used in datalogs and recipes.

Method name	Return type	Description
`SetRemoteFile(string strFile)`	`Result`	
`string strFile`		Full file name and path of the remote file

Property name	Return type	Description
`NewFileNameErrorCode`	`Result {get;}`	The error code saved after calling `SetRemoteFile` Method.
`NewFileNameIsValid`	`bool {get;}`	True if the file name is valid
`FileUpdateAllowed`	`bool {get;}`	True if the folder can add or replace a file in the list
`SelectedCount`	`int {get;}`	The number of files selected
`Files`	`List<IRemoteFile> {get;}`	Array of files in this folder
`FolderType`	`RemoteFolderType {get;}`	Type of folder (datalog or recipe)
`Exists`	`bool {get;}`	True if this folder exists on the device
`Selected`	`bool {get;set;}`	True if the folder is selected
`NewFile`	`string {get;}`	Full file path of the file to add or replace
`NewFileName`	`string {get;}`	Name of the file to add or replace
`Name`	`string {get;}`	Name of the folder

7.7.3 IRemoteInterface interface

`IRemoteInterface` is an interface used to represent distributed I/O on a network.

Property name	Return type	Description
`Devices`	`List<IBaseDevice>{get;}`	Array of remote interfaces used to represent decentralized I/O
`InterfaceType`	`RemoteInterfaceType{get;}`	Type of remote interface such as PROFINET or PROFIBUS
`Name`	`string {get;}`	The file name and extension on the device.

7.7.4 IHardware interface

`IHardware` is an interface used to represent the basic common hardware interface for devices and modules. `IModule` extends the `IHardware` interface.

Method name	Return type	Description
`SetFirmwareFile(string strFile)`	`Result`	Sets the firmware file to update on this device or module.

Property Name	Return Type	Description
`Comment`	`string {get;set;}`	Comment for each device and module
`Selected`	`bool {get;set;}`	Used to for external storage of the selected state
`NewFirmwareNameErrorCode`	`Result {get;}`	Last error from SetFirmwareFile method
`FirmwareUpdateAllowed`	`bool {get;}`	True when this device or module supports firmware update
`NewFirmwareNameIsValid`	`bool {get;}`	True when the firmware file is valid for this device or module
`Failsafe`	`bool {get;}`	True if device or module is failsafe
`Supported`	`bool {get;}`	True if this device or module is supported
`NewFirmwareFile`	`string {get;}`	Full file path of the firmware file
`NewFirmwareVersion`	`string {get;}`	Version of the firmware file which is displayed in the dropdown
`Configured`	`bool {get;}`	True if this device or module is configured.
`HardwareNumber`	`short {get;}`	Hardware revision number of device or module.
`SlotName`	`string {get;}`	Name of the slot of device or module.
`SubSlot`	`uint {get;}`	Sub slot number of device or module.
`Slot`	`uint {get;}`	Slot number of device or module.
`StationNumber`	`uint {get;}`	Station number of device or module.
`SerialNumber`	`string {get;}`	Serial number of device or module.
`FirmwareVersion`	`string {get;}`	Firmware version of device or module.

Property Name	Return Type	Description
ArticleNumber	string {get;}	Article number of device or module.
Description	string {get;}	Description of article number of device or module.
Name	string {get;}	Name of device or module.
ID	uint {get;}	ID of device or module.

7.7.5 IModule interface

IModule is an interface used to represent a module. IModule extends the IHardware interface.

Method name	Return type	Description
None		

Property name	Return type	Description
None		

7.7.6 IBaseDevice interface

IBaseDevice is an interface used to extend the IHardware interface which represents the most basic device type.

Method name	Return type	Description
GetHardwareFromID(uint hardwareID)	IHardware	Finds a device or module using an ID

Property name	Return type	Description
HardwareInDisplayOrder	IHardwareCollection	Array of hardware items in the order to be displayed
HardwareInFirmwareOrder	IHardwareCollection	Array of hardware items in the order of firmware update order
Modules	IModuleCollection	Array of modules
ThreadNumber	int	Current thread number of operation
Family	DeviceFamily	Family type enum

Events	Return type	Description
ProgressChanged	ProgressChangedEventHandler	Called to monitor progress
DataChanged	DataChangedEventHandler	Called when data changes in the API

7.7.7 IHardwareCollection interface

`IHardwareCollection` is an interface used to represent array of `IHardware` interfaces. This interface extends a .NET List class.

Property name	Type	Description
None		

7.7.8 IModuleCollection interface

`IModuleCollection` is an interface used to represent array `IModule` interface. This interface extends a .NET List class. This interface is used to represent local and remote modules in the hardware rack.

Property name	Type	Description
None		

7.8 Network class

7.8.1 Network constructor

The `.NET` class `Network` performs functions using a network interface card (NIC) installed on the PG/PC. The `Network` class is used to scan for available interface cards and to select the interface card that communicates with the industrial network.

To interact with the industrial network, your program declares a variable of type `Network`, as follows:

```
Network myNetwork = new Network();
```

You can use this object to find available network interfaces, and to select a network interface.

7.8.2 QueryNetworkInterfaceCards method

Return type	Method name
Result	QueryNetworkInterfaceCards

Parameters			
Name	Data type	Parameter type	Description
aInterfaces	List<string>	Out	A collection of all the network interface cards on the PG/PC listed by name.

To identify the available network interface cards, use the QueryNetworkInterfaceCards method, as shown in the following example:

```
Network myNetwork = new Network();

List<String> interfaces = new List<String>();
Result retVal = myNetwork.QueryNetworkInterfaceCards(out interfaces);
if (retVal.Succeeded)
{
  //---------------------------------------------------------
  // The method returns a List of strings.
  // Each string in the list represents an available NIC.
  // The list can be iterated using array notation.
  //---------------------------------------------------------
  for (Int32 index = 0; index < interfaces.Count; index++)
  {
    String strInterfaceName = interfaces[index];
  }
}
```

As the example shows, the method outputs a list of strings. Each item in the list represents an available network interface card, identified by name.

The QueryNetworkInterfaceCards method returns a Result object. This represents the status of the operation. At a high level, this object will indicate whether the operation succeeded (the Succeeded property is true) or failed (the Succeeded property is false). There are many reasons that an operation might fail.

For a complete description of the Result class, see also Result class (Page 127)

7.8.3 SetCurrentNetworkInterface method

Return type	Method name
`Result`	`SetCurrentNetworkInterface`

Parameters			
Name	Data type	Parameter type	Description
`strInterface`	`string`	In	The name of the network interface to use. Normally, this will be one of the names returned from the `QueryNetworkInterfaceCards` method.

To use one of the identified network interface cards to access the industrial network, it is necessary to "set" this interface. The following code shows how to assign one of the identified network interfaces for API operations. In this example, the code selects to use the first network interface card identified in the previous example.

```
Network myNetwork = new Network();

List<String> interfaces = new List<String>();
Result retVal = myNetwork.QueryNetworkInterfaceCards(out interfaces);
if (retVal.Succeeded)
{
    retVal = myNetwork.SetCurrentNetworkInterface(interfaces[0]);
    if (retVal.Succeeded)
    {
            //---------------------------------------------------
            // The action succeeded. Continue with operations.
            //---------------------------------------------------
    }
}
```

7.8.4 CurrentNetworkInterface property

This property is provided to query for the currently-selected network interface. This property is read-only. The following example shows how to use this property.

```
Network myNetwork = new Network();
string currentInterface = myNetwork.CurrentNetworkInterface;
```

Note

This property returns an empty string if no network interface was selected by a previous call to the `SetCurrentNetworkInterface` method.

7.8.5 ScanNetworkDevices method

Return type	Method name
`Result`	`ScanNetworkDevices`

Parameters			
Name	Data type	Parameter type	Description
`strFile`	`IProfinetDeviceCollection`	Out	A collection containing an `IProfinetDevice` element for each accessible device on the industrial network.

Once a network interface is selected, it is possible to query for the devices on the industrial network. The `ScanNetworkDevices` method outputs a collection of items, where each item represents a device connected directly to the industrial Ethernet network. These devices may include CPUs, local modules, decentralized IO stations, HMI, and other devices.

The following example creates a collection of all accessible devices on the selected network interface.

```
Network myNetwork = new Network();

IProfinetDeviceCollection scannedDevices;

Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
if (retVal.Succeeded)
{
    //---------------------------------------------------
    // The action succeeded. Continue with operations.
    //---------------------------------------------------
}
```

This method outputs an `IProfinetDeviceCollection`. This class is discussed in the next chapter.

Note

SIMATIC Automation Tool software license required for `ScanNetworkDevices` method.

If no SIMATIC Automation Tool software license is found at runtime, then the `ScanNetworkDevices` method returns an empty collection. No device information is reported to the calling application.

7.8.6 SetCommunicationsTimeout method

Return type	Method name
`Result`	`SetCommunicationsTimeout`

Parameters			
Name	Data type	Parameter type	Description
`nTimeout`	`uint`	Out	A collection containing an `IProfinetDevice` element for each accessible device on the industrial network.

You can set a time limit on the called operations. `SetCommunicationsTimeout` allows you to specify a time limit in seconds from 180 to 999 seconds. Any value outside of this range results in a failure of the operation.

The following example shows how to use the method to set a time limit on `ScanNetworkDevices` operations.

```
Network myNetwork = new Network();

IProfinetDeviceCollection scannedDevices;

Result retVal = Network.SetCommunicationsTimeout(180); //Timeout in 3 minutes
retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
if (retVal.Succeeded)
{
    //-----------------------------------------------------
    // The action succeeded. Continue with operations.
    //-----------------------------------------------------
}
```

7.8.7 GetCommunicationsTimeout method

Return type	Method name
uint	GetCommunicationsTimeout

After the value of the communications timeout has been set, you can retrieve the timeout value with the `GetCommunicationsTimeout` call. This method returns the current timeout value.

The following example shows how to retrieve the current timeout value and set a new timeout value of 180 seconds, if the current timeout value is greater than 180 seconds.

```
uint timeout = Network.GetCommunicationsTimeout();
if (timeout > 180)
{
    Result retVal = Network.SetCommunicationsTimeout(180);
}
```

7.8.8 CheckValidLicense method

Return type	Method name
`Result`	`CheckValidLicense`

A valid license is required to use the API. The `CheckValidLicense` method can be called to determine if a valid license exists.

The following example shows the use of the `CheckValidLicense` method.

```
Result result = Network.CheckValidLicense();
if (result.Succeeded)
{
    //---------------------------------------------------
    // A valid license
    //---------------------------------------------------
}
Else
{
    //---------------------------------------------------
    // No valid license
    //---------------------------------------------------
}
```

7.8.9 GetEmptyCollection method

Return type	Method name
`IProfinetDeviceCollection`	`GetEmptyCollection`

IProfinetDeviceCollection is an interface and cannot be instantiated. You can open a project or perform an import without doing a scan. Call this method to return an empty collection.

The following example shows how to use the method `GetEmptyCollection`.

```
IProfinetDeviceCollection collection = Network.GetEmptyCollection();
MemoryStream stream = new MemoryStream();

Result result = collection.WriteToStream(stream);
if (result.Succeeded)
{
    //----------------------------------------------------
    // Collection was serialized successfully
    //----------------------------------------------------
{
```

7.9 IProfinetDeviceCollection class

7.9.1 Iterating items in the collection

7.9.1.1 Iterating items in the collection

The `ScanNetworkDevices` method outputs an object of type `IProfinetDeviceCollection`. This class provides the ability to iterate the items in the collection in multiple ways. It also provides methods to filter the items in the collection based on certain criteria. The following sections describe the functionality available for the collection.

```
Network myNetwork = new Network();

IProfinetDeviceCollection scannedDevices;


Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
if (retVal.Succeeded)
{

    for (int deviceIdx = 0; deviceIdx < scannedDevices.Count; deviceIdx++)
    {
        //-------------------------------------------------------------
        // Each item in the collection is an IProfinetDevice.
        // This interface is described in detail in the next section
        //-------------------------------------------------------------
IProfinetDevice dev = scannedDevices[deviceIdx];
    }
}
```

The collection also supports iteration using the foreach syntax. The following example shows the same collection iterated using this syntax:

```
Network myNetwork = new Network();
IProfinetDeviceCollection scannedDevices;

Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
if (retVal.Succeeded)
{
    foreach (IProfinetDevice dev in scannedDevices)
    {
        //-----------------------------------------------
        // The variable "dev" now represents the next item in the collection
        //-----------------------------------------------
    }
}
```

7.9.1.2 GetEnumerator method

Return type	Method name
IEnumerator<IProfinetDevice>	GetEnumerator

Parameters			
Name	Data type	Parameter type	Description
None			

This method is used to enumerate all `IProfinetDevices` in the `IProfinetDeviceCollection`.

7.9.1.3 Count property

Return type	Property name
int	Count

This property returns the count of the number of `IProfinetDevices` in the `IProfinetDeviceCollection.`

7.9.1.4 [ ] property

Return type	Property name
IProfinetDevice	this[int index]

This property returns the `IProfinetDevice` at a specific index. See example below .

```
IProfinetDeviceCollection collection = Network.GetEmptyCollection();
MemoryStream stream = new MemoryStream();

Result result = collection.WriteToStream(stream);
if (retVal.Succeeded)
{
    //---------------------------------------------------
    // Collection was serialized successfully
    //---------------------------------------------------

    IProfinetDevice device = collection[0];
}
```

7.9.2 Filtering items in the collection

7.9.2.1 Collection items

The collection will contain an item for each device on the industrial Ethernet network. The collection can contain devices from different multiple product families (for example, S7-1200, S7-1500, ET200S).

The collection can also contain different "categories" of devices (for example, CPUs or IO stations). For different categories of devices, specific operations are available. So it may be useful at times to filter the collection to include only certain devices.

7.9.2.2 FilterByDeviceFamily method

Return type	Method name
`List<IProfinetDevice>`	`FilterByDeviceFamily`

Parameters			
Name	Data type	Parameter type	Description
`Familiestoinclude`	`List<DeviceFamily>`	Int	Families to include

This method returns a collection that includes only devices of the specified product families. The filter is first constructed as a list of one or more device families. For example, this declaration creates a filter for only S7-1200 and S7-1500 devices.

Pass this filter to the `FilterByDeviceFamily` method. The result is an `IProfinetDeviceCollection` that contains only the devices of the specified product families.

```
Network myNetwork = new Network();
List<DeviceFamily> fams = new List<DeviceFamily> { DeviceFamily.CPU1200,
DeviceFamily.CPU1500 };

IProfinetDeviceCollection scannedDevices;
Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);

List<IProfinetDevice> onlyPlus = scannedDevices.FilterByDeviceFamily(fams);
```

The resulting collection can then be iterated to perform actions only on the included devices.

Note

Passing an empty `List<DeviceFamily> w`ill result in the return of an empty collection.

7.9.2.3 FilterOnlyCPUs method

Return type	Method name
`List<ICPU>`	`FilterOnlyCPUs`

The SIMATIC Automation Tool API supports many operations that are only allowed for CPUs. For this reason, it is useful to filter the collection to include only the CPUs discovered on the network.

```
Network myNetwork = new Network();
IProfinetDeviceCollection scannedDevices;
Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);

List<ICPU> cpus = scannedDevices.FilterOnlyCpus();
foreach (ICPU cpu in cpus)
{
    //---------------------------------------------------------
    // Iterate through the list that only includes CPU devices
    //---------------------------------------------------------
}
```

This method returns a list of `ICPU`. Additional API operations are supported for CPU devices. The `ICPU` interface provides these operations. The `ICPU` interface is described in detail in the ICPU interface (Page 172) chapter.

7.9.3 Finding a specific device in the collection

7.9.3.1 FindDeviceByIP method

You can search for a specific device in the collection.

Return type	Method name
IProfinetDevice	FindDeviceByIP

Parameters			
Name	Data type	Parameter type	Description
ip	uint	In	The IP address to search for

The following example shows searching for a device at a specified IP address. If the device is not found in the collection, a NULL reference is returned.

```
Network myNetwork = new Network();
uint targetIPAddress = 0xC0A80001; // 192.168.0.1
IProfinetDeviceCollection scannedDevices;

Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
if (!retVal.Succeeded)
    return;

IProfinetDevice dev = scannedDevices.FindDeviceByIP(targetIPAddress);
if (dev != null)
{
    // Found it
}
```

7.9.3.2 FindDeviceByMAC method

The `FindDeviceByMAC` method can search for a device with a specific MAC address.

Return type	Method name
`IProfinetDevice`	`FindDeviceByMAC`

Parameters			
Name	Data type	Parameter type	Description
`mac`	`ulong`	In	The MAC address to search for

The following example searches for a device at a specified MAC address. If the device is not found in the collection, a NULL reference is returned.

```
ulong targetMAC = 0x112233445566; // equivalent to string 11:22:33:44:55:66
Network myNetwork = new Network();
IProfinetDeviceCollection scannedDevices;

Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
if (!retVal.Succeeded)
    return;

IProfinetDevice dev = scannedDevices.FindDeviceByMAC(targetMAC);
if (dev != null)
{
    // Found it
}
```

7.9.4 Serialization

7.9.4.1 Transferring a collection to/from an external data file

The following methods are provided to enable serializing and transfer of a collection's contents to/from an external data file. These methods are used by the SIMATIC Automation Tool application to support user project files.

7.9.4.2 WriteToStream method

Return type	Method name
`Result`	`WriteToStream`

Parameters			
Name	Data type	Parameter type	Description
`stream`	`Stream`	In	The destination for serialized output of the collection.

This method is used to externally store the contents of the collection. The following example shows the usage of this method:

```
Network myNetwork = new Network();
IProfinetDeviceCollection scannedDevices;

Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
if (!retVal.Succeeded)
    return;

FileStream f = File.Create("myDataFile.SAT");

retVal = scannedDevices.WriteToStream(f);

f.Close();
```

This method internally serializes version information, to support forward compatibility of saved data.

7.9.4.3 ReadFromStream method

The `ReadFromStream` method is used to create the collection from a previously created serialization file. The following example shows how to use this method:

Return type	**Method name**
`Result`	`ReadFromStream`

Parameters			
Name	Data type	Parameter type	Description
`stream`	`Stream`	In	The source for de-serializing the collection

This method is used to create the collection from a previously created serialization file. The following example shows the usage of this method:

```
IProfinetDeviceCollection devices = Network.GetEmptyCollection();

FileStream f = File.OpenRead("myDataFile.SAT");

Result retVal = devices.ReadFromStream(f);

f.Close();
```

7.9.5 Manually adding items to the collection

Depending on the physical topology of the industrial network, devices may exist on the network that cannot respond to a DCP command (such as those used by the `ScanNetworkDevices` method), but that can be accessed by IP address. For this scenario, methods are provided to allow you to manually add a device to the collection based on its address.

7.9.5.1 InsertDeviceByIP method

Return type	Method name
Result	InsertDeviceByIp

Parameters			
Name	Data type	Parameter type	Description
index	int	In	Location in the collection to insert the value
ip	uint	In	The IP address of the device to add to the collection.

The following code scans the network, and then manually adds a device, at a specific IP address, at the specified index.

```
Network myNetwork = new Network();
IProfinetDeviceCollection scannedDevices;

Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
if (!retVal.Succeeded)
    return;

UInt32 missingDeviceIPAddress = 0xC0A80001; // 192.168.0.1
Int32 index = 0;

retVal = scannedDevices.InsertDeviceByIP(index, missingDeviceIPAddress);
```

7.9.5.2 InsertDeviceByMAC method

Return type	**Method name**
`Result`	`InsertDeviceByMAC`

Parameters			
Name	Data type	Parameter type	Description
`index`	`int`	In	Location in the collection to insert the value
`mac`	`ulong`	In	The MAC Address of the device to add to the collection.

The following code scans the network, and then manually adds a device at a specified MAC address at the specified index.

```
Network myNetwork = new Network();
IProfinetDeviceCollection scannedDevices;

Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
if (!retVal.Succeeded)
    return;

UInt64 targetMAC = 0x112233445566; // equivalent to string 11:22:33:44:55:66
Int32 index = 0;

retVal = scannedDevices.InsertDeviceByMAC(index, targetMAC);
```

7.9.6 Copying data from a collection

7.9.6.1 CopyUserData method

You may find a situation where maintaining the options set on the current `IProfinetDeviceCollection` may become necessary. Instead of having the user re-enter this information, the following method is provided by the API.

Return type	Method name
`Result`	`CopyUserData`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`oldNetwork`	`IProfinetDeviceCollection`	In	Previous list used in the application

The following code copies user entered data from one network scan to another scan.

```
Network myNetwork = new Network();
IProfinetDeviceCollection scannedDevices;

Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
if (!retVal.Succeeded)
    return;

IProfinetDeviceCollection rescannedDevices;
retVal = myNetwork.ScanNetworkDevices(out rescannedDevices);
if (!retVal.Succeeded)
    return;

retVal = rescannedDevices.CopyUserData(scannedDevices);
if (!retVal.Succeeded)
    return;
```

7.9.7 Removing devices from the collection

7.9.7.1 Clear method

Return type	Method name
`void`	`Clear`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
None			

This method is used to clear the contents of the scanned devices.

7.9.7.2 Remove method

Return type	Method name
`void`	`Remove`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`device`	`IProfinetDevice`	In	Device to remove from the list

This method is used to delete a specific item from the collection.

7.10 IProfinetDevice interface

7.10.1 IProfinetDevice properties

Each item in the `IProfinetDeviceCollection` collection is represented by the `IProfinetDevice` interface. This interface provides access to the data and operations that are common to all devices directly connected to the industrial network.

The `IProfinetDevice` interface supports the following properties which provide information about the network device. These properties are all read-only. To ensure they will return the current information, your code should first call the `RefreshStatus` method on the device.

Property name	Return type	Description
`ArticleNumber`	`string {get;}`	The order number for the module. This is also known as MLFB or "article number".
`Comment`	`string {get;set;}`	This allows the user to specify a comment for the device and is used in the SIMATIC Automation Tool user interface. The comment is not relevant for API operations.
`Configured`	`bool {get;}`	True when the device has a valid configuration
`DefaultGateway`	`uint {get;}`	The default gateway address of the device, represented as an unsigned integer. The encoded gateway address uses one byte to represent each decimal value in the address. For example, the encoded value 0xC0A80001 is equivalent to the more common string representation of 192.168.0.1
`DefaultGatewayString`	`string {get;}`	The default gateway address of the device, represented as a string in the form "xx.xx.xx.xx" (i.e. "192.168.0.1")
`Description`	`string {get;}`	A description of the hardware item, based on the article number. This is the same description that the user would see in TIA Portal. (i.e. "CPU-1215 DC/DC/DC")
`DeviceFound`	`bool {get;}`	Was the device found on the network scan?
`DuplicateIP`	`bool {get;}`	Does the device have an IP address that is a duplicate?

Property name	Return type	Description
`DuplicateProfinetName`	`bool {get;}`	Does the device have a PROFINET Name that is a duplicate?
`Failsafe`	`bool {get;}`	Based on its ArticleNumber, Is this a failsafe device?
`Family`	`DeviceFamily {get;}`	What is the family of the device? For more information refer to the description of the `DeviceFamily` enum.
`FirmwareUpdateAllowed`	`bool {get;}`	Does this device support firmware update?
`FirmwareVersion`	`string {get;}`	The current firmware version of the device
`ID`	`uint {get;}`	The unique identifier for every device and module in the station. This is used as the unique identifier when executing a FirmwareUpdate.
`HardwareNumber`	`short {get;}`	The hardware version or "F-Stand" for the device. (Functional State)
`IP`	`uint {get;}`	The IP Address of the device, represented as an unsigned integer. The encoded IP Address uses one byte to represent each decimal value in the IP Address. For example, the encoded value 0xC0A80001 is equivalent to the more common string representation of "192.168.0.1" NOTE: SIMATIC Automation Tool V3.1 supports only IPv4 addresses. Ipv6 addressing is not supported.
`IPString`	`string {get;}`	The IP Address of the device, represented as a string in the form "xx.xx.xx.xx" (i.e. 192.168.0.1)
`MAC`	`ulong {get;}`	The unique MAC assigned to the device. The encoded MAC address uses one byte to encode each of the 6 octets defined for the address. For example, the encoded MAC address 0x112233445566 is equivalent to the more common string representation of 11:22:33:44:55:66

Property name	Return type	Description
`MACString`	`string {get;}`	The unique MAC assigned to the device, represented as a string in the form 11:22:33:44:55:66.
`Modules`	`IModuleCollection {get;}`	A collection of the modules plugged on the station. This property is described in detail here.
`Name`	`string {get;}`	The name of the device.
`NewFirmwareFile`	`string {get;}`	Location of the firmware file to be used in firmware update
`NewFirmwareNameErrorCode`	`Result {get;}`	ErrorCode attached to new Firmware name
`NewFirmwareNameIsValid`	`bool {get;}`	Is the set Firmware file valid?
`NewFirmwareVersion`	`string {get;}`	This property is used in the SIMATIC Automation Tool user interface. It is not relevant for API operations.
`NewDefaultGateway`	`String {get;}`	This property is used in the SIMATIC Automation Tool user interface. It is not relevant for API operations.
`NewIP`	`String {get;set;}`	This property is used in the SIMATIC Automation Tool user interface. It is not relevant for API operations.
`NewProfinetName`	`String{get;set;}`	This property is used in the SIMATIC Automation Tool user interface. It is not relevant for API operations.
`NewProgramName`	`String {get;set;}`	This property is used in the SIMATIC Automation Tool user interface. It is not relevant for API operations.
`NewRestoreName`	`String{get;set;}`	This property is used in the SIMATIC Automation Tool user interface. It is not relevant for API operations.
`ProfinetName`	`String {get;}`	The Profinet name for the device.
`ResetToFactoryAllowed`	`bool {get;}`	Is ResetToFactory allowed on the device?
`Selected`	`bool {get;set;}`	Marks the device as selected to enable operations to be performed
`SerialNumber`	`string {get;}`	The unique serial number for the device
`Slot`	`uint {get;}`	The slot number for the hardware item

Property name	Return type	Description
`SlotName`	`string {get;}`	This property is used in the SIMATIC Automation Tool user interface. It is not relevant for API operations.
`StationNumber`	`uint {get;}`	The station number of the device.
`SubSlot`	`uint {get;}`	The subslot of the device. This is relevant for pluggable submodules such as S7-1200 SB modules.
`Supported`	`bool {get;}`	True when the MLFB number exits in the database and the device is supported by current SIMATIC Automation Tool API.
`SubnetMask`	`uint {get;}`	The subnet mask of the device, represented as an unsigned integer. The encoded subnet mask uses one byte to represent each decimal value in the address. For example, the encoded value 0xFFFFFF00 is equivalent to the more common string representation of 255.255.255.0
`SubnetMaskString`	`string {get;}`	The subnet mask of the device, represented as a string in the form "xx.xx.xx.xx" (i.e. 192.168.0.1)

See also

DeviceFamily (Page 228)

Modules property and IModuleCollection class (Page 170)

7.10.2 IProfinetDevice methods

7.10.2.1 RefreshStatus method

Return type	Method name
Result	RefreshStatus

When the `IProfinetDeviceCollection` collection is created by calling the `ScanNetworkDevices` method, only a minimal amount of information is learned about each device. To get all the available information for the device, it is necessary to call the `RefreshStatus` method. This method makes a connection to the device, queries for various information, and then disconnects from the device.

The following code will call `RefreshStatus` for each device on the network.

```
Network myNetwork = new Network();

IProfinetDeviceCollection scannedDevices;

Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
if (retVal.Succeeded)
{
    foreach (IProfinetDevice dev in scannedDevices)
    {

        retVal = dev.RefreshStatus();
        if (retVal.Succeeded)
        {
                //-------------------------------------------------
                // Operation successful and the data can be trusted
                //-------------------------------------------------
        }
    }
}
```

The `RefreshStatus` method connects to the device to read information. The device may be password-protected against such access. Therefore, this method (and all methods that internally connect with the device) requires a password parameter.
The example shows the `IProfinetDevice` class. `ICPU` needs a password set on the device before calling the `RefreshStatus` method through the use of `SetPassword(EncryptedString)`, to legitimatize the connection.

See also

EncryptedString class (Page 126)

7.10.2.2 FirmwareUpdate method

Return type	Method name
`Result`	`FirmwareUpdate`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`hardwareID`	`uint`	In	The hardware identifier of the module
`bUpdateSameVersion`	`Bool`	In	If true, the method will proceed with the update. The update proceeds if the update file is the same version as the current firmware version of the module.

This method will update the firmware version for the specified hardware item (`hardwareID`) on the device. The `hardwareID` may specify either the device itself, or a module on the same rack.

Some devices do not support the firmware update feature. Check the property `FirmwareUpdateAllowed` to ensure that the current device supports this feature.

The following example searches for a device at a specific IP address and updates the firmware in that device.

```
Network myNetwork = new Network();
IProfinetDeviceCollection scannedDevices;

Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
uint targetIPAddress = 0xC0A80001; // 192.168.0.1
string updateFile = @"c:\myUpdates\6ES7 221 - 1BF32 - 0XB0 V02.00.00.upd";

if (!retVal.Succeeded)
    return;

IProfinetDevice dev = scannedDevices.FindDeviceByIP(targetIPAddress);
if (dev != null)
{
    dev.Selected = true;
    dev.SetFirmwareFile(updateFile);
    retVal = dev.FirmwareUpdate(dev.ID, true);
}
```

Using the `FirmwareUpdate` method, it is also possible to update the firmware for a module on a central station. The following code shows how to search for a CPU at a specific address and then searches the modules on that CPU for a specific article number. The firmware is then updated in modules that match the search criteria.

```
Network myNetwork = new Network();
IProfinetDeviceCollection scannedDevices;
Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
uint targetIPAddress = 0xC0A80001; // 192.168.0.1
string targetModule = @"6ES7 221 - 1BF32 - 0XB0";
string updateFile = @"c:\myUpdates\6ES7 221 - 1BF32 - 0XB0 V02.00.00.upd";

if (!retVal.Succeeded)
    return;

IProfinetDevice dev = scannedDevices.FindDeviceByIP(targetIPAddress);
if (dev != null)
{

    retVal = dev.RefreshStatus();
    if (!retVal.Succeeded)
        return;

    //----------------------------------
    // Search the modules on the CPU
    //----------------------------------
    IModuleCollection mods = dev.Modules;
    foreach (IModule mod in mods)
    {
        if (mod.ArticleNumber == targetModule)
        {
            mod.Selected = true;
            mod.SetFirmwareFile(updateFile);

            //----------------------------------------
            // Update firmware for matching module(s)
            //----------------------------------------
            dev.FirmwareUpdate(mod.ID, true);
        }
    }
}
```

Notice that the `FirmwareUpdate` method is called on the CPU. The `hardwareID` passed to the method indicates which module to update.

Note

Classic and Plus firmware update files

There are two different types of firmware update files.

- Classic firmware update folders contain several files that make up the firmware update. The header.upd or cpu_hd.upd in this folder is the file that is passed to the `FirmwareUpdate` method.
- The Plus firmware update file is a single update file. This is the file that is passed to the `FirmwareUpdate` method.

7.10.2.3 Identify method

Return type	Method name
`Result`	`Identify`

This method flashes a device LED or HMI screen for a specific network device. The flashing light helps identify the physical location of the device.

The following example flashes the LED or screen for the device that uses the IP address 192.168.0.1.

```
Network myNetwork = new Network();
uint targetIPAddress = 0xC0A80001; // 192.168.0.1
IProfinetDeviceCollection scannedDevices;

Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
if (retVal.Succeeded)
{
    //-----------------------------------------------------------
    // Search for the device at that IP, and flash LED/HMI screen
    //-----------------------------------------------------------
    IProfinetDevice dev = scannedDevices.FindDeviceByIP(targetIPAddress);
    if (dev != null)
    {
        retVal = dev.Identify();
    }
}
```

7.10.2.4 Reset method

Return type	Method name
`Result`	`Reset`

This method is used to reset a PROFINET device to its factory settings.

The following example calls the `Reset` method for a device at a specific IP address.

```
Network myNetwork = new Network();
uint targetIPAddress = 0xC0A80001; // 192.168.0.1
IProfinetDeviceCollection scannedDevices;

Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
if (retVal.Succeeded)
{
    //---------------------------------------------------------------
    // Search for device at that IP, and reset to factory default values
    //---------------------------------------------------------------
    IProfinetDevice dev = scannedDevices.FindDeviceByIP(targetIPAddress);
    if (dev != null)
    {
        retVal = dev.Reset();
    }
}
```

Note

This method cannot be used to reset a CPU. The `ICPU` interface supports a `ResetToFactory` method that is specific for CPUs.

7.10.2.5 SetIP method

Return type	Method name
`Result`	`SetIP`

Parameters			
Name	Data type	Parameter type	Description
`nIP`	`uint`	In	New encoded IP address
`nSubnet`	`uint`	In	New encoded subnet address
`nGateway`	`uint`	In	New encoded gateway address

This method is used to set or modify the IP address of a device.

For this operation to be successful, the device port configuration must be set to the "IP address is set directly on the device" option.

The following example searches for a device at a specified MAC address, and sets its IP address.

```
Network myNetwork = new Network();
ulong targetMACAddress = 0x112233445566; // 11:22:33:44:55:66
IProfinetDeviceCollection scannedDevices;

Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
if (retVal.Succeeded)
{
    //-----------------------------------------------
    // Search for the device at that MAC, and Set IP
    //-----------------------------------------------
    IProfinetDevice dev = scannedDevices.FindDeviceByMAC(targetMACAddress);
    if (dev != null)
    {
        retVal = dev.SetIP(0xC0A80001, 0xFFFFFF00, 0x0);
    }
}
```

Note

The `SetIP` method expects the addresses to be in encoded format (as shown above). The addresses can be converted from string format to encoded uint using the following C# code:

```
string userEnteredAddress = @"192.168.0.1"; // For example

//------------------------------
// Convert string address to uint
//------------------------------

System.Net.IPAddress ip = IPAddress.Parse(userEnteredAddress);
byte[] bytes = ip.GetAddressBytes();
Array.Reverse(bytes);

uint encodedIp = BitConverter.ToUInt32(bytes, 0);
// encoded IP address available for use
```

7.10.2.6 SetProfinetName method

Return type	Method name
`Result`	`SetProfinetName`

Parameters			
Name	Data type	Parameter type	Description
strName	String	In	New name for the PROFINET station

This method is used to set (or modify) the PROFINET station name for the device. For this operation to be successful, the device port must be configured with the "PROFINET name is set directly on the device" option".

```
Network myNetwork = new Network();
ulong targetMACAddress = 0x112233445566; // 11:22:33:44:55:66
IProfinetDeviceCollection scannedDevices;

Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
if (retVal.Succeeded)
{
    //------------------------------------------------
    // Search for the device at that MAC, and Set PROFINET Name
    //------------------------------------------------
    IProfinetDevice dev = scannedDevices.FindDeviceByMAC(targetMACAddress);
    if (dev != null)
    {
        retVal = dev.SetProfinetName("new name");
    }
}
```

7.10.2.7 ValidateIPAddressSubnet method

Return type	Method name
`Result`	`ValidateIPAddressSubnet`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`nIP`	`uint`	In	IP Address
`nSubnetMask`	`uint`	In	Subnet Mask

This method is used to validate a combination of IP and Subnet mask to verify that the pair is compatible.

The following example searches for a device at given MAC address and validates that the IP address and subnet mask of the device are compatible.

```
Network myNetwork = new Network();
ulong targetMACAddress = 0x112233445566; // 11:22:33:44:55:66
IProfinetDeviceCollection scannedDevices;

Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
if (retVal.Succeeded)
{
    //-----------------------------------------------
    // Search for the device at that MAC, and Set IP
    //-----------------------------------------------
    IProfinetDevice dev = scannedDevices.FindDeviceByMAC(targetMACAddress);

    if (dev != null)
    {
        retVal = dev.ValidateIPAddressSubnet(dev.IP, dev.SubnetMask);
    }
}
```

7.10.2.8 ValidatePROFINETName method

Return type	Method name
`Result`	`ValidatePROFINETName`

Parameters			
Name	Data type	Parameter type	Description
`strName`	`string`	In	PROFINET Name to validate

This method uses a PROFINET name that you provide. The PROFINET name is tested for validity as a device PROFINET name.

The following example searches for a device at given MAC address and validates that the given PROFINET Name is valid, before assigning it to the device.

```
Network myNetwork = new Network();
ulong targetMACAddress = 0x112233445566; // 11:22:33:44:55:66
IProfinetDeviceCollection scannedDevices;

Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
if (retVal.Succeeded)
{
    //------------------------------------------------
    // Search for the device at that MAC, and Set IP
    //------------------------------------------------
    IProfinetDevice dev = scannedDevices.FindDeviceByMAC(targetMACAddress);

    if (dev != null)
    {
        string name = "ValidName";
        retVal = dev.ValidatePROFINETName(name);
        if (retVal.Succeeded)
        {
            retVal = dev.SetProfinetName(name);
        }
    }
}
```

7.10.3 IProfinetDevice events

7.10.3.1 DataChanged event

The `DataChanged` event is supported on the `IProfinetDevice` interface.

This event allows the program to monitor whether changes have occurred to a given device on the network, due to other operations through the API. For example, if the program keeps a reference to a specific `IProfinetDevice`, it is possible to "listen" for certain changes to the device.

In the following example, the code attaches to the `DataChanged` event for every device on the network.

```
private void AttachEvents(IProfinetDeviceCollection devices)
{
  foreach (IProfinetDevice dev in devices)
  {
    dev.DataChanged += new DataChangedEventHandler(Dev_DataChanged);
  }
}

private void Dev_DataChanged(object sender, DataChangedEventArgs e)
{
  if (e.Type == DataChangedType.OperatingState)
  {
    //-------------------------------
     The mode changed for this device
    //-------------------------------
  }
}
```

Now, when any actions by the API cause a device to change operating mode, the method `Dev_DataChanged` is called.

Note

The `DataChanged` event does not actively monitor the live network, but monitors the properties of the `IProfinetDevice`. The state of this object must change in order to trigger the event.

The DataChangedEventArgs class

The `DataChanged` event handler will be passed a `DataChangedEventArgs` object. As shown in the above example, this class has a single property (Type) of type `DataChangedType`.

See also DataChangedType enumeration (Page 228)

7.10.3.2 ProgressChanged event

The `ProgressChanged` event is supported on the `IProfinetDevice` interface.

This event allows the program to monitor the progress of methods that take a long time. `FirmwareUpdate` is one example of such a method.

To utilize the event, an event handler is attached to the event. The event handler is called when there is a change in the progress of the operation.

The following example shows how you can monitor execution progress. This example shows a method that updates the firmware for a device on the network. This operation may take noticeable time. To monitor the progress of the action, an event handler is defined and attached to the `ProgressChanged` event. Once the firmware update is complete, the event handler is detached from the event.

```
private void UpdateCpuAtAddress(IProfinetDeviceCollection devices,uint
targetIPAddress, string updateFile)
{
    IProfinetDevice dev = devices.FindDeviceByIP(targetIPAddress);
    if (dev != null)
    {
        dev.ProgressChanged += new

        ProgressChangedEventHandler(Dev_ProgressChanged);
        dev.SetFirmwareFile(updateFile);
        dev.FirmwareUpdate(dev.ID, true);

        dev.ProgressChanged -= new

        ProgressChangedEventHandler(Dev_ProgressChanged);
    }
}

private void Dev_ProgressChanged(object sender, ProgressChangedEventArgs e)
{
    IProfinetDevice device = sender as IProfinetDevice;
    double percent = 0;
    if (device != null)
    {
        if (e.Count != 0)
        {
            string sPercent = e.Index.ToString() + " %";
        }
    }
}
```

The ProgressChangedEventArgs class

The `ProgressChanged` event handler is passed a `ProgressChangedEventArgs` object. This object has the following properties:

Property Name	Return Type	Description
`Action`	`ProgressAction`	A description of the current action. See also ProgressAction enumeration (Page 234)
`Cancel`	`bool`	Was the action canceled?
`Count`	`int`	The total amount of data to transfer
`ID`	`uint`	The hardware ID
`Index`	`int`	The current amount of data transferred

7.11 IModuleCollection class and module properties

7.11.1 Modules property and IModuleCollection class

The `IProfinetDevice` interface provides information about any modules (for example, signal modules, signal boards, CMs, CPs) plugged on the station. The Modules property returns a collection of these modules.

The following code shows accessing this information, given an IProfinetDevice (created in our earlier example).

```
Network myNetwork = new Network();
IProfinetDeviceCollection scannedDevices;

Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
if (retVal.Succeeded)
{
    //---------------------------------------------------
    // To ensure the information is current and complete,
    // first call RefreshStatus()
    //---------------------------------------------------
    retVal = scannedDevices[0].RefreshStatus();
    if (retVal.Succeeded)
    {
        //-----------------------------------------------------
        // The Modules property returns a collection of IModule
        //-----------------------------------------------------
        IModuleCollection modules = scannedDevices[0].Modules;
        foreach (IModule mod in modules)
        {
            //---------------------------------------------------
            // Get article number for every module on the station
            //---------------------------------------------------
            string displayArticleNum = mod.ArticleNumber;
        }
    }
}
```

7.11.2 IModule interface

Each module on the station is represented as an `IModule` interface. This interface provides a subset of the properties available for a device.

The `IModule` interface provides no methods. All operations on a module must be initiated at the device.
The IModule interface supports the following properties.

Property Name	Return Type	Description
`ArticleNumber`	`string`	The order number for the module. This is also known as MLFB or "article number".
`Comment`	`string`	This allows the user to specify a comment for the device and is used in the SIMATIC Automation Tool user interface. The comment is not relevant for API operations.
`Configured`	`bool`	True when the device has a valid configuration
`Description`	`string`	A description of the hardware item, based on the article number. This is the same description that the user would see in TIA Portal. (i.e. "CPU-1215 DC/DC/DC")
`Failsafe`	`bool`	Based on its ArticleNumber, Is this a failsafe device?
`FirmwareUpdateAllowed`	`bool`	Does this device support firmware update?
`FirmwareVersion`	`string`	The current firmware version of the device
`ID`	`uint`	The unique identifier for every device and module in the station. This is used as the unique identifier when executing a FirmwareUpdate.
`Name`	`string`	The name of the device.
`NewFirmwareNameErrorCode`	`Result`	`ErrorCode` attached to new Firmware name
`NewFirmwareNameIsValid`	`bool`	True when the firmware file is valid for this device or module.
`FirmwareVersion`	`string`	This property is used in the SIMATIC Automation Tool user interface. It is not relevant for API operations.
`Selected`	`bool`	Is the device currently selected? This is the checkbox state in the GUI.
`SerialNumber`	`string`	The unique serial number for the device.
`Slot`	`uint`	The slot number for the hardware item.
`SlotName`	`string`	This property is used in the SIMATIC Automation Tool user interface. It is not relevant for API operations.
`StationNumber`	`uint`	The station number of the device.
`SubSlot`	`uint`	The subslot of the device. This is relevant for pluggable submodules such as S7-1200 SB modules.
`Supported`	`FeatureSupport`	Is the detected network device supported by current SIMATIC Automation Tool operations?

7.12 ICPU interface

7.12.1 Identifying CPU devices in an IProfinetDeviceCollection

As discussed earlier, the `ScanNetworkDevices` method is called to generate an `IProfinetDeviceCollection`. This collection contains an item for every accessible device on the industrial network. These devices may include CPUs and decentralized I/O stations.

The `IProfinetDevice` interface provides properties and methods that are applicable to all categories of devices. However, there are properties and methods that are specific to a CPU device. These properties and methods are accessible using the `ICPU` interface.

To determine if a given `IProfinetDevice` interface actually represents a CPU device, simply cast it to an `ICPU`. If this cast is successful, then the network device is a CPU, and the properties/methods on the `ICPU` interface can be used. The following example illustrates this.

```
Network myNetwork = new Network();
IProfinetDeviceCollection scannedDevices;
Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
if (retVal.Succeeded)
{
    foreach (IProfinetDevice dev in scannedDevices)
    {
        ICPU devAsCpu = dev as ICPU;
        if (devAsCpu != null)
        {
            //----------------------------------------------------
            // The device is a CPU.
            // The ICPU interface can be used to interact with it.
            //----------------------------------------------------
        }
    }
}
```

Note

The `ICPU` interface inherits from `IProfinetDevice`. Therefore all the properties and methods supported on `IProfinetDevice` are also supported on `ICPU`.

Note

New, for the API V3.1 `ICPU` interface, is the requirement for setting the `Selected` and `SelectedConfirmed` flags, before you can perform operations on the devices through the API. `Selected` is required for all functions belonging to the `ICPU` interface while you must set the `SelectedConfirmed` , if the `ICPU` represents a safety operation. The safety-relevant operations are Program Update, Format Memory Card, Reset to Factory and Restore.

7.12.2 ICPU properties

The `ICPU` interface extends `IProfinetDevice` by adding the following properties. These properties are read-only. To ensure they will return the current information, your code should first call the `RefreshStatus` method.

Property name	Return type	Description
`RemoteInterfaces`	`List<IRemoteInterface>`	A list of any remote I/O interfaces configured for the CPU. The usage of this property is described in a later section of this document.
`DataLogFolder`	`IRemoteFolder`	Information about any Data Logs found on the SIMATIC Memory Card of the CPU
`RecipeFolder`	`IRemoteFolder`	Information about any Recipes found on the SIMATIC Memory Card of the CPU
`OperatingMode`	`OperatingState`	Designates the current mode of the CPU. This value is read-only
`IdentityCrisis`	`bool`	True when the identity of the device cannot be determined.
`LastRefreshSuccessful`	`bool`	True when the last call to `RefreshStatus` completed successfully.
`SelectedConfirmed`	`bool`	Methods that perform safety-relevant operations must set the `SelectedConfirmed` Flag to TRUE, when the user reselects one or more devices from a confirmation dialog for the operation and confirms the operation. `SelectedConfirmed` means that the operation is selected and confirmed.
`Initialized`	`bool`	True when the device or module has a valid configuration.
`InterfaceNumber`	`int`	The interface that the device is connected through
`Password`	`EncryptedString`	CPU Password used in functions performed on the device
`PasswordProtectionLevel`	`ProtectionLevel`	The protection level of a legitimized CPU password
`Protected`	`bool`	Is the CPU currently password protected. This means a password is required to access some or all features depending on access level.
`PasswordValid`	`bool`	Is the call to `SetPassword()` valid?

See also RemoteInterfaces property (Page 211)

7.12.3 ICPU flags

7.12.3.1 Program Update flags

To successfully perform safety-relevant functions on a device, more information is needed from the device. The following flags have been added to ensure that the Program Update function can be performed on a safety device correctly and securely.

Property name	Return type	Description
`NewProgramNamePassword`	`EncryptedString`	CPU Password used to attempt a connection after Program Update has finished. Value is set through the use of `SetProgramPassword(EncryptedString)`
`HasSafetyProgram`	`bool`	Boolean value set if the device has a Safety program present on the device. This is determined when connecting to a CPU.
`NewProgramNameIsValid`	`bool`	True when the method `SetProgramFolder` is called with a valid program.
`NewProgramNameIsSafety`	`bool`	True when the method `SetProgramFolder` is called with a valid a safety program
`NewProgramNameHasSafetyPassword`	`bool`	True when the method `SetProgramFolder` is called with a valid a safety program. False if a standard program is opened.
`NewProgramNamePasswordIsValid`	`bool`	True when the method `SetProgramPassword` is called with a valid password.
`NewProgramNamePasswordIsSafety`	`bool`	True when the method `SetProgramPassword` is called with a valid password and the password for the new program have a safety F-CPU password.
`ProgramUpdateSucceeded`	`bool`	True when the method `ProgramUpdate` method succeeds. The program update may still return an error.
`NewProgramNamePasswordPresent`	`bool`	True when the method `SetProgramFolder` is called and the program is password protected.
`NewProgramNamePasswordLevel`	`ProtectionLevel`	What is the protection level of the CPU password for the new program?
`NewProgramName`	`string`	What is the name of the new program?
`NewProgramFolder`	`string`	What is the folder location for the new program? Value is set through the `SetProgramFolder` method
`NewProgramNameFSignature`	`uint`	What is the FSignature of the new project? Used in the comparison process to determine if `ProgramUpdat`e finished successfully
`NewProgramNameIP`	`uint`	IP address that is stored in the new program
`NewProgramNameSubnetMask`	`uint`	Subnet mask of the device in the new program
`NewProgramNameGateway`	`uint`	Gateway of the device in the new program

Property name	Return type	Description
NewProgramNameErrorCode	Result	Accessible way to find issues that may be present in validating the new program, such as if the program is invalid for the device or if the IP found in the program already exists on the network
NewProgramNamePasswordError-Code	Result	Stores the error code of the last call to SetProgramPassword.

7.12.3.2 Restore flags

To successfully perform safety-relevant functions on a device, more information is needed from the device. The following flags are added to the version 3.1 API, so that the Restore from Backup function can be performed on a safety device correctly and securely. Due to the differences between the formatting of the program file and the restore file, not all information present for Program Update can be retrieved from backup file data.

Property name	Return type	Description
NewRestoreNamePassword	EncryptedString	CPU Password used to attempt connection after Restore has finished. Value is set through the use of SetBackup-FilePassword(EncryptedString)
NewRestoreNameIsValid	bool	True when the method SetBackupFile is called and the restore file is valid
NewRestoreNameIsSafety	bool	True when the method SetBackupFile is called and the restore file is a safety restore file
NewRestoreNamePassword-IsValid	bool	True when the call SetBackupFilePassword contains a valid password.
NewRestoreNamePasswordIsSa-fety	bool	True when the call SetBackupFilePassword contains a valid safety password.
RestoreSucceeded	bool	Did the Restore operation succeed?
NewRestoreName	string	What is the name of the new program?
NewRestoreFile	string	What is the file location for the new program? Value is set through the SetbackupFile method
NewRestoreNameFSignature	uint	What is the FSignature of the new project? Used in the comparison process to determine if Restore finished successfully
NewRestoreNameErrorCode	Result	Accessible way to find issues that may be present in validating the new program, such as if the program is invalid or incompatible with the device

7.12.3.3 Feature flags

In the SIMATIC Automation Tool V3.0 API release, flags identifying if certain features were permitted on a device are found at the `IProfinetDevice` level.
The V3.1 API moved these flags up to the `ICPU` and `IHMI` levels. The return type of these flags has changed from FeatureTypes to bool values.

Property name	Return type	Description
`ChangeModeAllowed`	`bool`	TRUE if the CPU supports a mode change operation? (RUN and STOP)
`BackupAllowed`	`bool`	TRUE if the CPU supports Backup operation.
`MemoryResetAllowed`	`bool`	TRUE if the CPU supports reset to factory operation.
`ProgramUpdateAllowed`	`bool`	TRUE if the CPU supports ProgramUpdate operation
`RestoreAllowed`	`bool`	TRUE if the CPU supports restore operation.
`FormatMCAllowed`	`bool`	TRUE if the CPU supports format memory card operation.
`PasswordAllowed`	`bool`	TRUE if the CPU supports a password.
`RemoteRecipesAllowed`	`bool`	TRUE if the CPU supports recipe operations.
`RemoteDataLogsAllowed`	`bool`	TRUE if the CPU supports data log operations.
`ServiceDataAllowed`	`bool`	TRUE if the CPU supports service data upload operation.
`SetTimeAllowed`	`bool`	TRUE if the CPU supports set and read time operation.
`DiagBufferAllowed`	`bool`	TRUE if the CPU supports diagnostic buffer operation.

7.12.4 ICPU methods

7.12.4.1 Protected CPUs and passwords

The following methods are provided on the `ICPU` interface. Most actions on the `ICPU` interface require a legitimized connection to the CPU. This may require a password. For this reason, most of the methods on the `ICPU` interface require a password parameter.

7.12.4.2 SetPassword method

Return type	Method name
`Result`	`SetPassword`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`password`	`EncryptedString`	In	Sets the CPU password for the object that is used to perform operations

Previously the CPU password parameter was handled by being passed along with the method call on the object. SIMATIC Automation Tool V3.1 changes this by allowing you to set the CPU password for the object instead of sending the CPU password each time with the parameters of a method.

The following example shows how to set the CPU password on a device.

```
Network myNetwork = new Network();
ulong targetMACAddress = 0x112233445566; // 11:22:33:44:55:66
IProfinetDeviceCollection scannedDevices;

Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
if (retVal.Succeeded)
{
    //-----------------------------------------------
    // Search for the device at that MAC, and SetPassword
    //-----------------------------------------------
    ICPU dev = scannedDevices.FindDeviceByMAC(targetMACAddress) as ICPU;
    if (dev != null)
    {
        if (dev.Protected)
        {
            retVal = dev.SetPassword(new EncryptedString("Password"));
        }
    }
}
```

7.12.4.3 SetProgramFolder method

Return type	Method name
`Result`	`SetProgramFolder`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`strFolder`	`string`	In	Sets the folder location for the downloaded program

Previously, the folder path parameter for Program Update was handled by being passed along with the method call on the object.

SIMATIC Automation Tool V3.1 changes this by allowing for the user to set the location of the folder on the device.

The method sets the following flags on the `ICPU` object.

- NewProgramFolder
- NewProgramName
- NewProgramNameIP
- NewProgramNameSubnetMask
- NewProgramNameGateway
- NewProgramNameIsValid

When the operation is performed on a safety object, you have the option for setting the following:

- NewProgramNameIsSafety
- NewProgramNameHasSafetyPassword

The following example shows how to set the program folder on a device.

```
Network myNetwork = new Network();
ulong targetMACAddress = 0x112233445566; // 11:22:33:44:55:66
IProfinetDeviceCollection scannedDevices;

Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
if (retVal.Succeeded)
{
    //-----------------------------------------------
    // Search for the device at that MAC, and SetProgramFolder
    //-----------------------------------------------
    ICPU dev = scannedDevices.FindDeviceByMAC(targetMACAddress) as ICPU;
    if (dev != null)
    {
        retVal = dev.SetProgramFolder(@"C:\MyFolder");
        if (retVal.Error == ErrorCode.ProgramPasswordNeeded)
        {
            retVal = dev.SetPassword(new EncryptedString("Password"));
        }
    }
}
```

Note

If the selected program has a password, then `SetProgramFolder` returns a failure. You must call the `SetProgramPassword` method and return successfully before you can call the `ProgramUpdate` method.

7.12.4.4 SetProgramPassword method

Return type	Method name
`Result`	`SetProgramPassword`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`password`	`EncryptedString`	In	Sets the CPU password for the project that is to be passed to CPU during ProgramUpdate.

After you perform a `ProgramUpdate`, the application attempts to reconnect to the device. If the program placed on the CPU is password protected, than setting the updated CPU password correctly allows you to regain access to the device, without the need to set another CPU password.

The method sets the following flags on the ICPU object:

- NewProgramNamePasswordIsValid
- NewProgramNamePasswordIsSafety
- NewProgramNamePasswordLevel

The following example shows how to set an updated CPU password on a device.

```
Network myNetwork = new Network();
ulong targetMACAddress = 0x112233445566; // 11:22:33:44:55:66
IProfinetDeviceCollection scannedDevices;

Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
if (retVal.Succeeded)
{
    //-----------------------------------------------
    // Search for the device at that MAC, and SetProgramFolder
    //-----------------------------------------------
    ICPU dev = scannedDevices.FindDeviceByMAC(targetMACAddress) as ICPU;
    if (dev != null)
    {
        retVal = dev.SetProgramFolder(@"C:\MyFolder");
        if (retVal.Error == ErrorCode.ProgramPasswordNeeded)
        {
            retVal = dev.SetPassword(new EncryptedString("Password"));
        }
    }
}
```

7.12.4.5 ProgramUpdate method

Return type	Method name
`Result`	`ProgramUpdate`

This method performs a program update on the CPU.

The following preconditions must be met in order to perform the operation:

- The device must be selected
  - If the device is a Fail-Safe device, `SelectedConfirmed` must be true
- The `NewProgramFolder` for the object must be set
  - If the new program contains a CPU password, then the `NewProgramPasswordPresent` must be true and have the value set through `SetProgramPassword`
- Program Update must be supported by the device (`ProgramUdpateAllowed`)

The following example searches the `IProfinetDeviceCollection` for a CPU at a specific IP address, and updates the program for that CPU.

```
Network myNetwork = new Network();
ulong targetMACAddress = 0x112233445566; // 11:22:33:44:55:66
IProfinetDeviceCollection scannedDevices;

Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
if (retVal.Succeeded)
{
    ICPU cpu = scannedDevices.FindDeviceByMAC(targetMACAddress) as ICPU;
    if (cpu != null && cpu.ProgramUpdateAllowed == true)
    {
        // Select cpu to update
        cpu.Selected = true;

        retVal = cpu.SetProgramFolder(@"C:\MyFolder");
        if (retVal.Error == ErrorCode.ProgramPasswordNeeded)
        {
            retVal = cpu.SetProgramPassword(new EncryptedString("Password"));
            if (retVal.Failed == true)
                return;
        }
        if (retVal.Failed == true)
            return;

        // Unique IP Address?
        if (cpu.DuplicateIP == true)
            return;

        // Is the device supported?
        if (cpu.Supported == false)
            return;

        // Is the device initialized?
        if (cpu.Initialized == false)
            return;

        // Is the device failsafe?
        if (cpu.Failsafe == true)
        {
            ConfirmationType type =
cpu.DetermineConfirmationMessage(FailsafeOperation.ProgramUpdateOperation);

            // Verify type and confirm
            cpu.SelectedConfirmed = true;

            // Check to make sure we can update
            if (cpu.HasSafetyProgram == true || cpu.Protected == true
||cpu.NewProgramNameIsSafety == true)
            {
                // Is the device password protected?
                if (cpu.Protected == true)
                {
                // Was a valid password supplied?
                if (cpu.PasswordValid == false)
                return;

                // Are we legitimized to the safety level?
                bool bSufficientAccess = cpu.PasswordProtectionLevel ==
ProtectionLevel.Failsafe;
                if (bSufficientAccess == false)
                return;

                }
            }
        }
        else
        {
            // Is a safety program about to be loaded?
            if (cpu.NewProgramNameHasSafetyPassword == true)
                return;
        }

        // Is the device password protected?
        if (cpu.Protected == true)
        {
            // Was a valid password supplied?
            if (cpu.PasswordValid == false)
                return;


            // Do we have a sufficient legitimization level for the operation to
succeed?
            bool bSufficientAccess = cpu.PasswordProtectionLevel ==
ProtectionLevel.Failsafe || cpu.PasswordProtectionLevel == ProtectionLevel.Full;
            if (bSufficientAccess == false)
                return;
        }
        // Perform a program update
        retVal = cpu.ProgramUpdate();

        // Reset
        cpu.SelectedConfirmed = false;
    }
}
```

Note

The folder name passed to the `ProgramUpdate` method should contain a folder called SIMATIC.S7S. The SIMATIC.S7S folder contains the program to download.

7.12.4.6 SetBackupFile method

Return type	Method name
Result	SetBackupFile

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`strFile`	`string`	In	Sets the location for a backup file

Previously the file path parameter for Restore was handled by being passed along with the method call on the object. SIMATIC Automation Tool V3.1 changes this by allowing you to set the location of the folder on the PG/PC data storage device.

The method sets the following flags on the ICPU object:

- NewRestoreName
- NewRestoreFile
- NewRestoreNameIsValid
- NewRestoreNameIsSafety
- NewRestorenameFSignature

The following example shows how to set the backup file path.

```
Network myNetwork = new Network();
ulong targetMACAddress = 0x112233445566; // 11:22:33:44:55:66
IProfinetDeviceCollection scannedDevices;

Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
if (retVal.Succeeded)
{
    ICPU cpu = scannedDevices.FindDeviceByMAC(targetMACAddress) as ICPU;
    if (cpu != null && cpu.RestoreAllowed == true)
    {
        // Select cpu to update
        cpu.Selected = true;

        retVal = cpu.SetBackupFile(@"C:\MyFolder\Backup.s7pbkp");
        if (retVal.Error == ErrorCode.ProgramPasswordNeeded)
        {
            retVal = cpu.SetBackupFilePassword(new EncryptedString("Password"));
            if (retVal.Failed == true)
                return;
        }

        if (retVal.Failed == true)
            return;

        // Unique IP Address?
        if (cpu.DuplicateIP == true)
            return;

        // Is the device supported?
        if (cpu.Supported == false)
            return;

        // Is the device initialized?
        if (cpu.Initialized == false)
            return;

        // Is the device failsafe?
        if (cpu.Failsafe == true)
        {
            ConfirmationType type =
            cpu.DetermineConfirmationMessage(FailsafeOperation.RestoreOperation);

            // Verify type and confirm
            cpu.SelectedConfirmed = true;

            // Check to make sure we can update
            if (cpu.HasSafetyProgram == true || cpu.Protected == true ||
cpu.NewRestoreNameIsSafety == true)
            {
                // Is the device password protected?
                if (cpu.Protected == true)
                {
                    // Was a valid password supplied?
                    if (cpu.PasswordValid == false)
                        return;


                    // Are we legitimized to the safety level?
                    bool bSufficientAccess = cpu.PasswordProtectionLevel ==
ProtectionLevel.Failsafe;
                    if (bSufficientAccess == false)
                        return;
                }
            }
        }
        else
        {
            // Is a safety program about to be loaded?
            if (cpu.NewRestoreNameIsSafety == true)
                return;
        }

        // Is the device password protected?
        if (cpu.Protected == true)
        {
            // Was a valid password supplied?
            if (cpu.PasswordValid == false)
                return;

            // Do we have a sufficient legitimization level for the operation to
succeed?
            bool bSufficientAccess = cpu.PasswordProtectionLevel ==
ProtectionLevel.Failsafe || cpu.PasswordProtectionLevel == ProtectionLevel.Full;
            if (bSufficientAccess == false)
                return;
        }
        // Perform a restore
        retVal = cpu.Restore();

        // Reset
        cpu.SelectedConfirmed = false;
    }
}
```

Note

If the selected backup file has a CPU password, `SetBackupFile` returns with a failure. The `SetBackupFilePassword` method must be called and return successfully, before you can call the `Restore` method.

7.12.4.7 SetBackupFilePassword method

Quintessence

Return type	Method name
`Result`	`SetBackupFilePassword`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`password`	`EncryptedString`	In	Sets the password for the project that is passed to CPU during restore.

After you perform a Restore, the application will attempt to reconnect to the device. If the program loaded on the CPU is password protected, setting the updated CPU correctly lets the user regain access to the device, without the need to set another CPU password.

The method sets the following flag on the `ICPU` object.

- NewRestoreNamePassword

The following example shows how to set an updated CPU password on a device.

```
Network myNetwork = new Network();
ulong targetMACAddress = 0x112233445566; // 11:22:33:44:55:66
IProfinetDeviceCollection scannedDevices;

Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
if (retVal.Succeeded)
{
    //-----------------------------------------------
    // Search for the device at that MAC, and SetBackupFilePassword
    //-----------------------------------------------
ICPU dev = scannedDevices.FindDeviceByMAC(targetMACAddress) as ICPU;
if (dev != null)
    {
        if (dev.Protected)
        {
            retVal = dev.SetBackupFile(@"C:\MyFolder");
            retVal = dev.SetBackupFilePassword(new EncryptedString("Password"));
        }
    }
}
```

7.12.4.8 Restore method (ICPU interface)

Return type	Method name
`Result`	`Restore`

This method is used to restore the information from a previous backup of the CPU. Some CPUs do not support the backup/restore feature.

The following preconditions must be met in order to perform the operation.

- The device must be selected.
  - If the device is a failsafe device, `SelectedConfirmed` must be true.
- The `NewProgramFolder` for the object must be set.
  - If the new program contains a CPU password, the `NewProgramPasswordPresent` must be true and have the value set through `SetProgramPassword`
- `Restore` must be supported to perform the action (`RestoreAllowed`)

The following example searches the `IProfinetDeviceCollection` for a CPU at a specific IP address. When found, it checks that the password protected CPU supports the restore feature, then calls the Restore method on a Backup file that does not have a CPU password.

```
Network myNetwork = new Network();
uint targetIPAddress = 0xC0A80001; // 192.168.0.1
string bkFile = @"C:\MyCPUBackupFile.s7pbkp";
IProfinetDeviceCollection devices;

Result retVal = myNetwork.ScanNetworkDevices(out devices);
if (retVal.Succeeded)
{
    IProfinetDevice dev = devices.FindDeviceByIP(targetIPAddress);
    if (dev != null)
    {
        ICPU devAsCpu = dev as ICPU;
        if (devAsCpu != null && devAsCpu.RestoreAllowed)
        {
            retVal = devAsCpu.SetPassword(new EncryptedString("Password"));
            retVal = devAsCpu.SetBackupFile(bkFile);
            devAsCpu.Selected = true;
            if (devAsCpu.Failsafe)
                devAsCpu.SelectedConfirmed = true;

            retVal = devAsCpu.Restore();
        }
    }
}
```

7.12.4.9 Backup method (ICPU interface)

Return type	Method name
`Result`	`Backup`

Parameters			
Name	Data type	Parameter type	Description
`strFile`	`string`	In	A fully-qualified path and filename where the backup should be stored.

This method is used to back up the data in a CPU. Some CPUs do not support the backup/restore feature. You can check the property `BackupAllowed` to ensure that the current CPU supports this feature.

The following example searches the `IProfinetDeviceCollection` for a CPU at a specific IP address. When found it checks that the CPU supports the backup feature, and calls the `Backup` method.

```
Network myNetwork = new Network();
uint targetIPAddress = 0xC0A80001; // 192.168.0.1
string bkFile = @"C:\MyCPUBackupFile.s7pbkp";
IProfinetDeviceCollection devices;

Result retVal = myNetwork.ScanNetworkDevices(out devices);
if (retVal.Succeeded)
{
    foreach (IProfinetDevice dev in devices)
    {
        ICPU devAsCpu = dev as ICPU;

        if (devAsCpu != null && devAsCpu.IP == targetIPAddress &&
devAsCpu.BackupAllowed)
        {
            devAsCpu.Selected = true;
            retVal = devAsCpu.Backup(bkFile);
        }
    }
}
```

7.12.4.10 DownloadRecipe method

Return type	Method name
`Result`	`DownloadRecipe`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`strFile`	`string`	In	The complete path and filename of the recipe file to download (transfer from PG/PC to CPU memory card).

This method is used to add or replace a recipe .CSV file on the CPU memory card. Some CPUs do not support remote recipe access. You can check the property `RemoteRecipesAllowed` to ensure that the current CPU supports this feature. The following code example shows writing a recipe to the CPU memory card.

```
Network myNetwork = new Network();
uint targetIPAddress = 0xC0A80001; // 192.168.0.1
string rcpFile = @"C:\NewRecipe.csv";
IProfinetDeviceCollection devices;

Result retVal = myNetwork.ScanNetworkDevices(out devices);
if (retVal.Succeeded)
{
    foreach (IProfinetDevice dev in devices)
    {
        ICPU devAsCpu = dev as ICPU;
        if ((devAsCpu != null) &&
            (devAsCpu.IP == targetIPAddress) &&
            (devAsCpu.RemoteRecipesAllowed))
        {
            retVal = devAsCpu.SetPassword(new EncryptedString("Password"));
            IRemoteFolder recipes = devAsCpu.RecipeFolder;
            recipes.Selected = true;
            recipes.SetRemoteFile(rcpFile);
            retVal = devAsCpu.DownloadRecipe(rcpFile);

        }
    }
}
```

Note

If a recipe with the same name already exists on the CPU memory card, then it is replaced.

7.12.4.11 DeleteDataLog method

Return type	Method name
`Result`	`DeleteDataLog`

Parameters			
Name	Data type	Parameter type	Description
`strFileName`	`string`	In	Filename of Data Log file to delete, from a CPU memory card.

This method is used to delete a Data Log file from a CPU's memory card.

Some CPUs do not support remote Data Log access. Check the property `RemoteDataLogsAllowed` to ensure that the current CPU supports this feature.

The following code example uses the `DataLogFolder` property to iterate all Data Logs on the CPU memory card. Each Data Log is deleted.

```
Network myNetwork = new Network();
IProfinetDeviceCollection devices;

Result retVal = myNetwork.ScanNetworkDevices(out devices);
if (retVal.Succeeded)
{
    foreach (IProfinetDevice dev in devices)
    {
        ICPU devAsCpu = dev as ICPU;
        if (devAsCpu != null)
        {
            if (devAsCpu.RemoteDataLogsAllowed)
            {
                //------------------------------------------
                // First check that data logs are available on the memory card
                //------------------------------------------
                if (devAsCpu.DataLogFolder.Exists)
                {
                    devAsCpu.SetPassword(new EncryptedString("Password"));
                    //------------------------------------------
                    // Search for all data log files
                    //------------------------------------------



                for (Int32i=devAsCpu.DataLogFolder.Files.Count - 1; i>=0; i--)

                {

                    IRemoteFile datalog = devAsCpu.DataLogFolder.Files[i];

                    if (datalog != null)

                    {
                        datalog.Selected = true;
                        //------------------------------------------
                        // Delete the data log.
                        //------------------------------------------
                        devAsCpu.DeleteDataLog(datalog.Name);
                    }
                }
            }
        }
    }
}
```

7.12.4.12 DeleteRecipe method

Return type	Method name
`Result`	`DeleteRecipe`

Parameters			
Name	Data type	Parameter type	Description
`strFileName`	`string`	In	Filename of Recipe file to delete from a CPU memory card

This method is used to delete a recipe file from a CPU's memory card.

Some CPUs do not support remote Recipe access. Check the property `RemoteRecipesAllowed` to ensure that the current CPU supports this feature.

The following code example uses the `RecipeFolder` property to iterate all Recipes on the CPU memory card. Each recipe is deleted.

```
Network myNetwork = new Network();
IProfinetDeviceCollection devices;

Result retVal = myNetwork.ScanNetworkDevices(out devices);
if (retVal.Succeeded)
{
    foreach (IProfinetDevice dev in devices)
    {
        ICPU devAsCpu = dev as ICPU;
        if (devAsCpu != null)
        {
            if (devAsCpu.RemoteDataLogsAllowed)
            {
                //------------------------------------------
                // First check that recipes are available on the memory card
                //------------------------------------------
                if (devAsCpu.RecipeFolder.Exists)
                {
                    devAsCpu.SetPassword(new EncryptedString("Password"));
                    //------------------------------------------
                    // Search for all data log files
                    //------------------------------------------
                for (Int32i=devAsCpu.RecipeFolder.Files.Count - 1; i>=0; i--)
                {
                    IRemoteFile recipe = devAsCpu.RecipeFolder.Files[i];
                    if (recipe != null)
                    {
                        recipe.Selected = true;
                        //------------------------------------------
                        // Delete the recipe.
                        //------------------------------------------
                        devAsCpu.DeleteRecipe(recipe.Name);
                    }
                }
            }
        }
    }
}
```

7.12.4.13 GetCurrentDateTime method

Return type	Method name
Result	GetCurrentDateTime

Parameters			
Name	Data type	Parameter type	Description
DateTime	System.DateTime	Out	Current date and time returned from the CPU

This method gets the current timestamp for the CPU.

The following example searches the IProfinetDeviceCollection for a CPU at a specific IP address, and gets its current date and time.

```
Network myNetwork = new Network();
IProfinetDeviceCollection devices;

uint targetIPAddress = 0xC0A80001; // 192.168.0.1

Result retVal = myNetwork.ScanNetworkDevices(out devices);
if (!retVal.Succeeded)
    return;
foreach (IProfinetDevice dev in devices)
{
    ICPU devAsCpu = dev as ICPU;
    if ((devAsCpu != null) && (devAsCpu.IP == targetIPAddress))
    {
        devAsCpu.SetPassword(new EncryptedString("Password"));
        devAsCpu.Selected = true;

        DateTime curTime = new DateTime();

        retVal = devAsCpu.GetCurrentDateTime(out curTime);
    }
}
```

7.12.4.14 GetDiagnosticsBuffer method

Return type	Method name
`Result`	`GetDiagnosticsBuffer`

Parameters			
Name	Data type	Parameter type	Description
`DiagnosticsItems`	`List<DiagnosticsItem>`	Out	A collection of Diagnostics Items: Each item in the collection represents an entry in the diagnostics buffer.

This method reads the current diagnostics entries from the CPU. Each entry is represented as a `DiagnosticsItem`. This class is described after the code example. The Language enum is described in the API enumerations chapter. The following example searches the `IProfinetDeviceCollection` for a CPU at a specific IP address. When found it reads the diagnostics information from the CPU.

```
Network myNetwork = new Network();
uint targetIPAddress = 0xC0A80001; // 192.168.0.1
List<DiagnosticsItem> aLogs = new List<DiagnosticsItem>();
IProfinetDeviceCollection devices;

Result retVal = myNetwork.ScanNetworkDevices(out devices);
if (retVal.Succeeded)
{
    foreach (IProfinetDevice dev in devices)
    {
        ICPU devAsCpu = dev as ICPU;
        if ((devAsCpu != null) && (devAsCpu.IP == targetIPAddress))
        {
            devAsCpu.SetPassword(new EncryptedString("Password"));
            devAsCpu.Selected = true;
            retVal = devAsCpu.GetDiagnosticsBuffer(out aLogs, Language.English);
            if (retVal.Succeeded)
            {
                for (int idxLog = 0; idxLog < aLogs.Count; idxLog++)
                {
                    string descr = aLogs[idxLog].Description1;
                }
            }
        }
    }
}
```

The DiagnosticsItem class

Note

Change the second parameter in the method `GetDiagnosticsBuffer`, to get the strings in another supported language,.

The `GetDiagnosticsBuffer` method returns a collection of `DiagnosticsItem` objects.

This class defines the following members:

Member name	Data type	Description
`TimeStamp`	`System.DateTime`	Time the diagnostic event was logged.
`State`	`Byte`	Ingoing or outgoing message
`Description1`	`String`	Title
`Description2`	`String`	Detail

7.12.4.15 MemoryReset method

Return type	Method name
`Result`	`MemoryReset`

This method performs a memory reset on the CPU.

The following example searches the `IProfinetDeviceCollection` for a CPU at a specific IP address, and calls `MemoryReset` for that CPU.

```
Network myNetwork = new Network();
uint targetIPAddress = 0xC0A80001; // 192.168.0.1
IProfinetDeviceCollection devices;

Result retVal = myNetwork.ScanNetworkDevices(out devices);
if (retVal.Succeeded)
{
    IProfinetDevice dev = devices.FindDeviceByIP(targetIPAddress);
    if (dev != null)
    {
        ICPU devAsCpu = dev as ICPU;
        if (devAsCpu != null)
        {
            devAsCpu.SetPassword(new EncryptedString("Password"));
            devAsCpu.Selected = true;
            retVal = devAsCpu.MemoryReset();
        }
    }
}
```

7.12.4.16 ResetToFactoryDefaults method

Return type	**Method name**
`Result`	`ResetToFactoryDefaults`

Parameters			
Name	Data type	Parameter type	Description
`password`	`EncryptedString`	In	This method opens a legitimized connection to the device. Therefore, a password may be required

This method resets a CPU to its factory default values.

The following example searches the `IProfinetDeviceCollection` for a CPU at a specific IP address, and calls the `ResetToFactoryDefaults` method. You must set the `SelectedConfirmed` flag to TRUE, for Fail-Safe devices.

```
Network myNetwork = new Network();
ulong targetMACAddress = 0x112233445566; // 11:22:33:44:55:66
IProfinetDeviceCollection scannedDevices;

Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
if (retVal.Succeeded)
{
    ICPU cpu = scannedDevices.FindDeviceByMAC(targetMACAddress) as ICPU;
    if (cpu != null && cpu.ResetToFactoryAllowed == true)
    {
        // Select cpu to update
        cpu.Selected = true;

        // Unique IP Address?
        if (cpu.DuplicateIP == true)
            return;

        // Is the device supported?
        if (cpu.Supported == false)
            return;

        // Is the device initialized?
        if (cpu.Initialized == false)
            return;

        // Is the device failsafe?
        if (cpu.Failsafe == true) { ConfirmationType type =
cpu.DetermineConfirmationMessage(FailsafeOperation.ResetToFactoryOperation);
        // Verify type and confirm
        cpu.SelectedConfirmed = true;

        // Check to make sure we can update
        if (cpu.HasSafetyProgram == true || cpu.Protected == true ||
cpu.NewRestoreNameIsSafety == true)
        {
            // Is the device password protected?
            if (cpu.Protected == true)
            {
                // Was a valid password supplied?
                if (cpu.PasswordValid == false)
                    return;

                // Are we legitimized to the safety level?
                bool bSufficientAccess = cpu.PasswordProtectionLevel ==
ProtectionLevel.Failsafe;
                if (bSufficientAccess == false)
                    return;
            }
        }
    }
    else
    {
        // Is a safety program about to be loaded?
        if (cpu.NewRestoreNameIsSafety == true)
            return;
    }
    // Is the device password protected?
    if (cpu.Protected == true)
    {
        // Was a valid password supplied?
        if (cpu.PasswordValid == false)
            return;

        // Do we have a sufficient legitimization level for the operation to succeed?
        bool bSufficientAccess = cpu.PasswordProtectionLevel ==
ProtectionLevel.Failsafe || cpu.PasswordProtectionLevel == ProtectionLevel.Full;
        if (bSufficientAccess == false)
            return;
     }
        // Perform a reset to factory default values
        retVal = cpu.ResetToFactoryDefaults();

        // Reset
        cpu.SelectedConfirmed = false;
    }
}
```

7.12.4.17 SetOperatingState method

Return type	Method name
`Result`	`SetOperatingState`

Parameters			
Name	Data type	Parameter type	Description
`nRequestState`	`OperatingStateREQ`	In	The new operating state

This method is used to change the operating state of a CPU.

Some CPUs do not support this feature. Check the property C`hangeModeAllowed` to ensure that the current CPU supports this feature.

The following example searches the `IProfinetDeviceCollection` for a CPU at a specific IP address. When found, it checks that the CPU supports the change mode feature, and sets the CPU to RUN.

```
Network myNetwork = new Network();
uint targetIPAddress = 0xC0A80001; // 192.168.0.1
IProfinetDeviceCollection devices;

Result retVal = myNetwork.ScanNetworkDevices(out devices);
if (retVal.Succeeded)
{
    IProfinetDevice dev = devices.FindDeviceByIP(targetIPAddress);
    if (dev != null)
    {
        ICPU devAsCpu = dev as ICPU;
        if ((devAsCpu != null) && (devAsCpu.ChangeModeAllowed))
        {
            devAsCpu.SetPassword(new EncryptedString("Password"));
            devAsCpu.Selected = true;
            retVal = devAsCpu.SetOperatingState(OperatingStateREQ.Run);
        }
    }
}
```

7.12.4.18 SetCurrentDateTime method

Return type	Method name
`Result`	`SetCurrentDateTime`

Parameters			
Name	Data type	Parameter type	Description
`password`	`EncryptedString`	In	This method opens a legitimized connection to the device. Therefore, a password may be required
`time`	`System.DateTime`	In	New value for the CPU current time.

This method sets the current time for the CPU. The configured time transformation rules are not affected by this action. Therefore, the specified `DateTime` value is based on UTC time and not the local time.

The following example traverses the entire industrial network and sets the current time for each CPU device to the current time of the PG/PC.

```
Network myNetwork = new Network();
IProfinetDeviceCollection devices;

Result retVal = myNetwork.ScanNetworkDevices(out devices);
if (retVal.Succeeded)
{
    foreach (IProfinetDevice dev in devices)
    {
        ICPU devAsCpu = dev as ICPU;
        if (devAsCpu != null)
        {
            devAsCpu.SetPassword(new EncryptedString("Password"));
            devAsCpu.Selected = true;
            retVal = devAsCpu.SetCurrentDateTime(DateTime.UtcNow);
        }
    }
}
```

7.12.4.19 UploadDataLog method

Return type	**Method name**
`Result`	`UploadDataLog`

Parameters			
Name	Data type	Parameter type	Description
`strFileName`	`string`	In	The filename of the Data Log to upload from a CPU's removable SIMATIC memory card.
`strDestinationFolder`	`string`	In	Fully qualified path where the uploaded file Data Log file is stored

This method uploads a copy of specified Data Log file from a CPU’s memory card to your PG/PC. Some CPUs do not support remote Data Log access. Check the property `RemoteDataLogsAllowed` to ensure that the current CPU supports this feature.
The following code example uses the `DataLogFolder` property to iterate all Data Logs on the CPU memory card. A copy of each Data Log is uploaded to the folder C:\MyDataLogs.

```
Network myNetwork = new Network();
IProfinetDeviceCollection devices;

Result retVal = myNetwork.ScanNetworkDevices(out devices);
if (retVal.Succeeded)
{
    foreach (IProfinetDevice dev in devices)
    {
        ICPU devAsCpu = dev as ICPU;
        if (devAsCpu != null)
        {
            devAsCpu.SetPassword(new EncryptedString("Password"));

            if (devAsCpu.RemoteDataLogsAllowed)
            {
                //------------------------------------------
                // First check that data logs are available on the memory card
                //------------------------------------------
                if (devAsCpu.DataLogFolder.Exists)
                {
                    //------------------------------------------
                    // Search for all data log files
                    //------------------------------------------
                    foreach (IRemoteFile datalog in devAsCpu.DataLogFolder.Files)
                    {
                        datalog.Selected = true;
                        //------------------------------------------
                        // Upload a copy of each data log.
                        //------------------------------------------
                        devAsCpu.UploadDataLog(datalog.Name, @"C:\MyDataLogs");
                    }
                }
            }
        }
    }
}
```

7.12.4.20 UploadRecipe method

Return type	Method name
`Result`	`UploadRecipe`

Parameters			
Name	Data type	Parameter type	Description
`strFileName`	`string`	In	The filename of the recipe to upload from the CPU memory card.
`strDestinationFolder`	`string`	In	Fully qualified path where the uploaded Recipe file is written

This method upload a copy of a recipe file from a CPU's memory card. Some CPUs do not support remote recipe access. Check the property `RemoteRecipesAllowed` to ensure that the current CPU supports this feature.

The following code example uses the `RecipeFolder` property to iterate all recipes on the CPU memory card. A copy of each Recipe is uploaded to the folder C:\MyRecipes.

```
Network myNetwork = new Network();
IProfinetDeviceCollection devices;

Result retVal = myNetwork.ScanNetworkDevices(out devices);
if (retVal.Succeeded)
{
    foreach (IProfinetDevice dev in devices)
    {
        ICPU devAsCpu = dev as ICPU;if (devAsCpu != null)
        {
            devAsCpu.SetPassword(new EncryptedString("Password"));
            if (devAsCpu.RemoteRecipesAllowed)
            {
                //------------------------------------------
                // First check that recipes are available on the memory card.
                //------------------------------------------
                if (devAsCpu.RecipeFolder.Exists)
                {
                    //------------------------------------------
                    // Search for all recipe files
                    //------------------------------------------
                    foreach (IRemoteFile recipe in devAsCpu.RecipeFolder.Files)
                    {
                        recipe.Selected = true;
                        //------------------------------------------
                        // Upload a copy of each recipe.
                        //------------------------------------------
                        devAsCpu.UploadRecipe(recipe.Name, @"C:\MyRecipes");
                    }
                }
            }
        }
    }
}
```

7.12.4.21 UploadServiceData method

Return type	Method name
`Result`	`UploadServiceData`

Parameters			
Name	Data type	Parameter type	Description
`strPath`	`string`	In	A fully-qualified path to the folder containing the program card contents.

This method can upload the service data from a defective CPU.

The following example searches the `IProfinetDeviceCollection` for a CPU at a specific IP address. It then checks the current `OperatingState` of the CPU. If the CPU is defective, then the service data is uploaded.

```
Network myNetwork = new Network();
IProfinetDeviceCollection devices;

uint targetIPAddress = 0xC0A80001; // 192.168.0.1
string strDiagFolder = @"c:\Diagnostics";

Result retVal = myNetwork.ScanNetworkDevices(out devices);
if (retVal.Succeeded)
{
    IProfinetDevice dev = devices.FindDeviceByIP(targetIPAddress);
    if (dev != null)
    {
        ICPU devAsCpu = dev as ICPU;

        if (devAsCpu != null)
        {
            devAsCpu.SetPassword(new EncryptedString("Password"));
            devAsCpu.Selected = true;
            if (devAsCpu.OperatingMode == OperatingState.Defective)
            {
                retVal = devAsCpu.UploadServiceData(strDiagFolder);
            }
        }
    }
}
```

7.12.4.22 FormatMemoryCard method

Return type	Method name
`Result`	`FormatMemoryCard`

This method is used to format the removable SIMATIC memory card plugged into a CPU.

The following example searches the `IProfinetDeviceCollection` for a CPU at a specific IP address. It then formats the memory card of the device. You must set the `SelectedConfirmed` flag to TRUE, for Fail-Safe devices.

```
Network myNetwork = new Network();
ulong targetMACAddress = 0x112233445566; // 11:22:33:44:55:66
IProfinetDeviceCollection scannedDevices;

Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
if (retVal.Succeeded)
{
    ICPU cpu = scannedDevices.FindDeviceByMAC(targetMACAddress) as ICPU;
    if (cpu != null && cpu.FormatMCAllowed == true)
    {
        // Select cpu to update
        cpu.Selected = true;

        // Unique IP Address?
        if (cpu.DuplicateIP == true)
            return;

        // Is the device supported?
        if (cpu.Supported == false)
            return;

        // Is the device initialized?
        if (cpu.Initialized == false)
            return;

        // Is the device failsafe?
        if (cpu.Failsafe == true)
        {
            ConfirmationType type =
cpu.DetermineConfirmationMessage(FailsafeOperation.ResetToFactoryOperation);

        // Verify type and confirm
        cpu.SelectedConfirmed = true;

        // Check to make sure we can update
        if (cpu.HasSafetyProgram == true || cpu.Protected == true ||
cpu.NewRestoreNameIsSafety == true)
        {
            // Is the device password protected?if (cpu.Protected == true)
            {
                // Was a valid password supplied?
                if (cpu.PasswordValid == false)
                    return;

            // Are we legitimized to the safety level?
            bool bSufficientAccess = cpu.PasswordProtectionLevel ==
ProtectionLevel.Failsafe;
            if (bSufficientAccess == false)
                return;
            }
        }
    }
    else
    {
        // Is a safety program about to be loaded?
        if (cpu.NewRestoreNameIsSafety == true)
            return;
    }

    // Is the device password protected?
    if (cpu.Protected == true)
    {
        // Was a valid password supplied?
        if (cpu.PasswordValid == false)
            return;

        // Do we have a sufficient legitimization level for the operation to succeed?
        bool bSufficientAccess = cpu.PasswordProtectionLevel ==
ProtectionLevel.Failsafe || cpu.PasswordProtectionLevel == ProtectionLevel.Full;
        if (bSufficientAccess == false)
            return;
    }
        // Perform a restore
        retVal = cpu.FormatMemoryCard();

    // Reset
    cpu.SelectedConfirmed = false;
    }
}
```

7.12.4.23 DetermineConfirmationMessage

Return type	Method name
`ConfirmationType`	`DetermineConfirmationMessage`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`operation`	`FailsafeOperation`	In	Operation that is evaluated.

This method determines what the content of the confirmation message contains, when the user attempts to perform a safety-relevant operation.

The `ConfirmationType` class contains a series of hamming codes that are each representative of an error message that should be displayed to the user as follows:

ConfirmationType	Message to be displayed
`SafetyPasswordIsBeingUsed`	An operation to a standard program is about to be initiated using the safety CPU password.
`DeletingExistingSafetyProgram`	An existing safety program is about to be deleted.
`ReplacingExistingSafetyProgram`	An existing safety program is about to be updated with another safety program.
`ReplacingExistingSafetyProgramWithNonSafetyProgram`	An existing safety program is about to be replaced by a standard program.
`LoadingSafetyProgram`	A safety program is about to be loaded for the first time.

The following example searches the `IProfinetDeviceCollection` for a CPU at a specific IP address. This method is used to display a message about the safety-relevant status in a user confirmation dialog box, prior to executing a safety-relevant operation.

```
Network myNetwork = new Network();
uint targetIPAddress = 0xC0A80001; // 192.168.0.1
IProfinetDeviceCollection devices;

Result retVal = myNetwork.ScanNetworkDevices(out devices);
if (retVal.Succeeded)
{
    IProfinetDevice dev = devices.FindDeviceByIP(targetIPAddress);
    if (dev != null)
    {
        ICPU devAsCpu = dev as ICPU;
        if (devAsCpu != null)
        {
            devAsCpu.SetPassword(new EncryptedString("Password"));
            devAsCpu.Selected = true;
            if (devAsCpu.Failsafe)
            {
                devAsCpu.SelectedConfirmed = true;
            }
            ConfirmationType confirm =
devAsCpu.DetermineConfirmationMessage(FailsafeOperation.FormatMCOperation);

            if (confirm == ConfirmationType.DeletingExistingSafetyProgram)
                devAsCpu.FormatMemoryCard();
        }
    }
}
```

7.12.5 RemoteInterfaces properties

7.12.5.1 Decentralized I/O modules

Each CPU may support multiple decentralized I/O interfaces. Information about the devices attached on these remote interfaces is available through the `RemoteInterfaces` property.

The following example shows how to access this information for all the CPUs on a network.

```
Network myNetwork = new Network();
IProfinetDeviceCollection scannedDevices;

Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
if (retVal.Succeeded)
{
    foreach (IProfinetDevice dev in scannedDevices)
    {
        ICPU devAsCpu = dev as ICPU;

        List<IRemoteInterface> decentalNets = devAsCpu.RemoteInterfaces;
        foreach (IRemoteInterface net in decentalNets)
        {
            //-----------------------------------------------------------
            // Inspect the remote interface
            //-----------------------------------------------------------
        }
    }
}
```

7.12.5.2 IRemoteInterface properties

The `IRemoteInterface` interface supports the following properties. These properties are read-only.

Property name	Return type	Description
`Devices`	`List<IBaseDevice>`	A list of any decentralized I/O stations connected to this remote interface
`InterfaceType`	`RemoteInterfaceType`	The communications protocol for this remote interface See also RemoteInterfaceType enumeration (Page 234)
`Name`	`string`	The configured name for the remote interface.

The `Devices` property can be used to traverse a decentralized network. Each device in the decentralized network is represented by an `IBaseDevice` interface. This interface has a subset of the properties available for an `IProfinetDevice` and provides the limited functionality available for these devices in the SIMATIC Automation Tool API.

The following properties are available on the `IBaseDevice` interface.

Property name	Return type	Description
`ArticleNumber`	`string`	The order number for the module. This is also known as MLFB or article number.
`Comment`	`string`	This allows the user to specify a comment for the device. This is used in the SIMATIC Automation Tool user interface. It is not relevant for API operations.
`Configured`	`bool`	Does the device have a valid configuration.
`Description`	`string`	A description of the hardware item, based on the article number. This is the same description that the user would see in TIA Portal. (i.e. "CPU-1215 DC/DC/DC")
`Failsafe`	`FeatureSupport`	Based on its Article number, Is this a failsafe device?
`Family`	`DeviceFamily`	What is the family of the device? For more information refer to the description of the DeviceFamily (Page 228) enum.
`FirmwareUpdateAllowed`	`FeatureSupport`	Does this device support firmware update?
`FirmwareVersion`	`string`	The current firmware version of the device
`HardwareInFirmwareOrder`	`IHardwareCollection`	Hardware collection in firmware order
`HardwareInDisplayOrder`	`IHardwareCollection`	Hardware in displayed order
`HardwareNumber`	`short`	Number identifier
`ID`	`uint`	The unique identifier for every device and module in the station. This is used as the unique identifier when executing a FirmwareUpdate.
`Modules`	`IModuleCollection`	A collection of local modules connected on the station. This property is described in detail here (Page 170).
`Name`	`string`	The name of the device.
`NewFirmwareFile`	`string`	File path to the new firmware file
`NewFirmwareVersion`	`string`	This property is used in the SIMATIC Automation Tool user interface. It is not relevant for API operations.
`NewFirmwareNameIsValid`	`bool`	Is the new firmware file valid?
`Selected`	`bool`	Is the device selected?

Property name	Return type	Description
`SerialNumber`	`string`	The unique serial number for the device.
`Slot`	`uint`	The slot number for the hardware item.
`SlotName`	`string`	This property is used in the SIMATIC Automation Tool user interface. It is not relevant for API operations.
`StationNumber`	`uint`	The station number of the device.
`SubSlot`	`uint`	The subslot of the device. This is relevant for pluggable submodules such as SB-1200.
`Supported`	`FeatureSupport`	Is the detected network device supported by current SIMATIC Automation Tool API operations?

Using the Devices property of the `IRemoteInterface`, it is possible to inspect all the stations on the decentralized network.

To extend the earlier example:

```
Network myNetwork = new Network();
IProfinetDeviceCollection scannedDevices;

Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
if (!retVal.Succeeded)
    return;

foreach (IProfinetDevice dev in scannedDevices)
{
    ICPU devAsCpu = dev as ICPU;
    if (devAsCpu == null)
        continue;

    List<IRemoteInterface> decentalNets = devAsCpu.RemoteInterfaces;
    List<string> orderNumbers = new List<string>();

    foreach (IRemoteInterface net in decentalNets)
    {
        //-----------------------------------------------------------
        // Inspect the remote interface
        //-----------------------------------------------------------
        if (net.InterfaceType == RemoteInterfaceType.Profinet)
        {
            //-------------------------------------------------------
            // Look at each decentral stations
            //-------------------------------------------------------
            List<IBaseDevice> stations = net.Devices;

            foreach (IBaseDevice station in stations)
            {
                orderNumbers.Add(station.ArticleNumber);
            }
        }
    }
}
```

This example traverses all remote PROFINET interfaces and creates a list of the article numbers for all decentralized stations on the industrial network.

Since the `IBaseDevice` also supports the `Modules` property, it is simple to extend the example further to look at not only the decentralized stations, but also all the local modules on each station.

7.13 IHMI interface

7.13.1 IHMI interface

The `ScanNetworkDevices` method is called to generate an `IProfinetDeviceCollection`. This collection contains an item for every accessible device on the industrial network. These devices may include CPUs, HMIs, decentralized I/O stations, and other Siemens devices. The `IProfinetDevice` interface provides properties and methods that apply to all categories of devices.

However, there are methods that are only used for HMI devices. These properties and methods are accessible using the `IHMI` interface.

To determine if a given `IProfinetDevice` interface actually represents a HMI device, cast it to an `IHMI`. If this cast is successful, then the network device is an HMI, and you can use the methods on the `IHMI` interface. The following example illustrates this procedure.

```
Network myNetwork = new Network();
IProfinetDeviceCollection scannedDevices;

Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
if (retVal.Succeeded)
{
    foreach (IProfinetDevice dev in scannedDevices)
    {
        IHMI devAsHmi = dev as IHMI;
        if (devAsHmi != null)
        {
            //-----------------------------------------------------
            // The device is an HMI.
            // The IHMI interface can be used to interact with it.
            //-----------------------------------------------------
        }
    }

}
```

Note

The `IHMI` interface inherits from `IProfinetDevice`. Therefore, all the properties and methods supported on `IProfinetDevice` are also supported on `IHMI`. The following IHMI properties topic describes only the properties/methods that are unique to the `IHMI` interface.

Note

The `IHMI` interface supports the FirmwareUpdate method. However, this method will always return the error `FirmwareUpdateNotSupported`. To update the firmware for an `HMI` device, it is necessary to execute the Program update method (Page 219).

7.13.2 IHMI properties and flags

7.13.2.1 IHMI properties

The V3.1 `IHMI` interface has also received changes from the SIMATIC Automation Tool V3.0 API where many of the parameters have changed into properties on the object.

Property name	ReturntType	Description
`DeviceType`	`string`	Returns the type of HMI that the object represents
`FirmwareDeviceVersion`	`string`	Returns the firmware version that is present on the HMI
`RuntimeDeviceVersion`	`string`	Returns the runtime version that is present on the HMI

7.13.2.2 Program update flags

You can use these flags with the V3.1 IHMI interface.

Property Name	Return Type	Description
`NewProgramNameIsValid`	`bool`	True when the method `SetProgramFolder` is a called with a valid program folder. False if program is not valid.
`ProgramUpdateSucceeded`	`bool`	True when program update is successful even though an error may return from internal refresh status
`NewProgramName`	`string`	Name of the new program
`NewProgramFolder`	`string`	Folder location for the new program: Value is set through the `SetProgramFolder` method
`NewProgramNameErrorCode`	`Result`	Codes to find issues that may be present in validating the new program, such as if the program is invalid for the device or if the IP assigment in the program already exists on the network

7.13.2.3 Restore flags

You can use these flags with the V3.1 IHMI interface.

Property Name	Return Type	Description
`NewRestoreNameIsValid`	`bool`	True when the method SetBackupFolder is a called with a valid backup file. False if backup file is not valid.
`RestoreSucceeded`	`bool`	True when restore is successful even though an error could have been returned from internal refresh status
`NewRestoreName`	`string`	What is the name of the new program?
`NewRestoreFile`	`string`	What is the file location for the new program? Value is set through the SetbackupFile method
`NewRestoreNameErrorCode`	`Result`	Accessible way to find issues that may be present in validating the new program, such as if the program is invalid or incompatible with the device

7.13.2.4 Feature flags

Quintessence

You can use these flags with the V3.1 IHMI interface.

Property Name	Return Type	Description
`BackupAllowed`	`bool`	True if the device allows backups
`ProgramUpdateAllowed`	`bool`	True if the device allows program updates
`RestoreAllowed`	`bool`	True if the device allows restores

7.13.3 Backup method (IHMI interface)

Return type	Method name
Result	Backup

Parameters			
Name	Data type	Parameter type	Description
strFile	string	In	A fully-qualified path and filename where the backup file is stored

This method is used to back up the data for an HMI. Some HMIs do not support the backup/restore feature. Check the property BackupAllowed to ensure that the current HMI supports this feature. The following example searches the IProfinetDeviceCollection for an HMI at a specific IP address. When found, it checks that the HMI supports the backup feature, and calls the Backup method.

```
Network myNetwork = new Network();
uint targetIPAddress = 0xC0A80001; // 192.168.0.1
string bkFile = @"C:\MyCPUBackupFile.s7pbkp";
IProfinetDeviceCollection devices;

Result retVal = myNetwork.ScanNetworkDevices(out devices);
if (retVal.Succeeded)
    {
        foreach (IProfinetDevice dev in devices)
        {
            IHMI devAsHmi = dev as IHMI;

            if ((devAsHmi != null) &&
                (devAsHmi.IP == targetIPAddress) &&
                (devAsHmi.BackupAllowed)
               )
            {
                devAsHmi.Selected = true;
                retVal = devAsHmi.Backup(bkFile);
        }
    }
}
```

7.13.4 ProgramUpdate method (IHMI interface)

Return type	Method name
`Result`	`ProgramUpdate`

This method updates the HMI device's operating system and run-time software. The parameter `strPath` assigns a folder containing the program to load.

To successfully complete the `ProgramUpdate` method on the IHMI interface, you must verify the following:

- Device is selected
- `NewProgramFolder` has been set (Accomplished through `SetProgramFolder`)

The following example searches the `IProfinetDeviceCollection` for an HMI at a specific IP address, and updates the program for that HMI.

```
Network myNetwork = new Network();
uint targetIPAddress = 0xC0A80001; // 192.168.0.1
IProfinetDeviceCollection devices;

Result retVal = myNetwork.ScanNetworkDevices(out devices);
if (retVal.Succeeded)
{
    IProfinetDevice dev = devices.FindDeviceByIP(targetIPAddress);
    if (dev != null)
    {
        IHMI devAsHMI = dev as IHMI;
        if (devAsHMI != null)
        {
            devAsHMI.Selected = true;
            devAsHMI.SetProgramFolder(
@"c:\myFolder\ProgramUpdate\Simatic.HMI\RT_Projects\Project1");
            retVal = devAsHMI.ProgramUpdate();
        }
    }
}
```

The new program folder must contain the following files for successful completion:

```
DownloadTask.xml
ProjectCharacteristics.rdf
```

These files are generally found in a folder that is created (using TIA Portal) in the following format:

```
{DeviceName)\Simatic.HMI\RT_Projects\{ProjectName}.{DeviceName}
```

For example:

```
"C:\Desktop\hmim14000100a\Simatic.HMI\RT_Projects\DasBasicUndMobilePanelen.hmim140001
00a[KTP700 Mobile]"
```

Note

HMI operating system, and runtime software updates

`ProgramUpdate` for an HMI device is different than for a CPU. This method can update firmware, operating system, and runtime software for HMI devices. You do not have the option to select a partial update. SIMATIC Automation Tool updates all data components as necessary, for a consistent download. An HMI Program Update card can have more than one project on the card which requires entering a folder under `\Simatic.HMI\RT_Projects\` to download.

7.13.5 Restore method (IHMI interface)

Return type	Method name
Result	Restore

Use this method to restore HMI device data from a previous backup of the device. Some HMI devices do not support the backup/restore feature. Check the property RestoreAllowed to ensure that the current HMI device supports this feature.

To successfully complete the Restore method on the IHMI interface the user most verify the following:

- Device is selected
- BackupFile has been set (Accomplished through SetBackupFile)

The following example searches the IProfinetDeviceCollection for an HMI at a specific IP address. When found, it checks that the HMI supports the restore feature and then calls the Restore method.

```
Network myNetwork = new Network();
ulong targetMACAddress = 0x112233445566; // 11:22:33:44:55:66
IProfinetDeviceCollection scannedDevices;

Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
if (retVal.Succeeded)
{
    IHMI hmi = scannedDevices.FindDeviceByMAC(targetMACAddress) as IHMI;
    if (hmi != null && hmi.RestoreAllowed == true)
    {
        // Select cpu to update
        hmi.Selected = true;

        retVal = hmi.SetBackupFile(@"C:\MyFolder\Backup.s7pbkp");
        if (retVal.Failed == true)
            return;

        // Unique IP Address?
        if (hmi.DuplicateIP == true)
            return;

        // Is the device supported?
        if (hmi.Supported == false)
            return;

        // Perform a restore
        retVal = hmi.Restore();
    }
}
```

7.13.6 SetProgramFolder method

Return type	Method name
`Result`	`SetProgramFolder`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`strFolder`	`string`	in	Sets the folder location where the program download source is stored.

Previously the folder path parameter for Program Update was passed along with the method call on the object. SIMATIC Automation Tool V3.1 changes this by allowing for the user to set the location of the folder.

The following example shows how to set the program folder on an HMI device.

```
Network myNetwork = new Network();
ulong targetMACAddress = 0x112233445566; // 11:22:33:44:55:66
IProfinetDeviceCollection scannedDevices;

Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
if (retVal.Succeeded)
{
    IHMI hmi = scannedDevices.FindDeviceByMAC(targetMACAddress) as IHMI;
    if (hmi != null && hmi.ProgramUpdateAllowed == true)
    {
        // Select cpu to update
        hmi.Selected = true;

        retVal = hmi.SetProgramFolder(@"C:\MyFolder");
        if (retVal.Failed == true)
            return;

        // Unique IP Address?
        if (hmi.DuplicateIP == true)
            return;

        // Is the device supported?
        if (hmi.Supported == false)
            return;

        // Perform a program update
        retVal = hmi.ProgramUpdate();
    }
}
```

The folder that is specified for "`strPath`" must contain the following files for successful completion:

- DownloadTask.xml
- ProjectCharacteristics.rdf

These files are generally found in a folder that is created (using TIA Portal) in the following format:

{DeviceName)\Simatic.HMI\RT_Projects\{ProjectName}.{DeviceName}

For example:

"C:\Desktop\hmim14000100a\Simatic.HMI\RT_Projects\DasBasicUndMobilePanelen.hmim14000100a[KTP700 Mobile]"

Note

ProgramUpdate for an HMI device is different than for a CPU. For an HMI device, this method may update the operating system and run-time software. You do not have the option o select a partial update. The SIMATIC Automation Tool updates components as necessary, for a consistent download. An HMI Program Update card can have more than one project on the card which requires entering a folder under \Simatic.HMI\RT_Projects\.

7.13.7 SetBackupFile method

Return type	Method name
`Result`	`SetBackupFile`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`strFile`	`string`	in	Sets the folder location where the backup file source is stored.

Previously, the file path parameter for Restore was passed along with the method call on the object. SIMATIC Automation Tool V3.1 changes this behavior and allows you to set the location of the folder.

The method sets the following flags on the ICPU object:

- NewRestoreName
- NewRestoreFile
- NewRestoreNameIsValid

The following example shows how to set the backup file path on an HMI device.

```
Network myNetwork = new Network();
ulong targetMACAddress = 0x112233445566; // 11:22:33:44:55:66
IProfinetDeviceCollection scannedDevices;

Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
if (retVal.Succeeded)
{
    IHMI hmi = scannedDevices.FindDeviceByMAC(targetMACAddress) as IHMI;
    if (hmi != null && hmi.RestoreAllowed == true)
    {
        // Select cpu to update
        hmi.Selected = true;

        retVal = hmi.SetBackupFile(@"C:\MyFolder\Backup.s7pbkp");
        if (retVal.Failed == true)
            return;

        // Unique IP Address?
        if (hmi.DuplicateIP == true)
            return;

        // Is the device supported?
        if (hmi.Supported == false)
            return;

        // Perform a restore
        retVal = hmi.Restore();
    }
}
```

7.14 Support classes

7.14.1 Diagnosticsitem

A diagnostic item contains diagnostic imformation for a single event.

Property Name	Return Type	Description
`TimeStamp`	`DateTime`	Time stamp of the diagnostic event
`State`	`byte`	0=Outgoing event 1=Incoming event
`Description1`	`string`	Basic description
`Description2`	`string`	Detailed description

7.15 Exceptions

7.15.1 CriticalInternalErrorException

Quintessence

The V3.1 API interface has added a custom exception when a critical condition has been detected.

When this exception is triggered, it is recommended that you shut down the application that is using the API. When a critical error occurs in the code, this exception is called.

```
Network myNetwork = new Network();
try
{
    uint targetIPAddress = 0xC0A80001; // 192.168.0.1
    IProfinetDeviceCollection devices;

    Result retVal = myNetwork.ScanNetworkDevices(out devices);
    if (retVal.Succeeded)
    {
        IProfinetDevice dev = devices.FindDeviceByIP(targetIPAddress);
        if (dev != null)
        {
            ICPU devAsCpu = dev as ICPU;
            if (devAsCpu != null)
            {
                devAsCpu.SetPassword(new EncryptedString("Password"));
                devAsCpu.Selected = true;
                if (devAsCpu.Failsafe)devAsCpu.SelectedConfirmed = true;
                retVal = devAsCpu.ResetToFactoryDefaults();
            }
        }
    }
}
catch (CriticalInternalErrorException e)
{
    // A critical internal error has occurred within the API
}
catch (Exception e)
{
    // An exception has occurred within the API

}
```

7.16 API enumerations

7.16.1 DataChangedType

This enumeration defines the possible argument values for the DataChangedEventHandler (Page 167).

```
Invalid
OperatingState
RackInformation
Folders
File
ProfinetName
IPAddress
Password
```

7.16.2 DeviceFamily

This enumeration specifies the product family for a hardware item.

```
CPU1200
CPU1500
ET200AL
ET200ECO
ET200M
ET200MP
ET200PRO
ET200S
ET200SP
HMI
NetworkDevice
None
SITOPUPS
Unsupported
```

7.16.3 ConfirmationType

This enumeration specifies the user confirmation types for safety-relevant operations.

```
Invalid = 0
SafetyPasswordIsBeingUsed = 0x2f161717
DeletingExistingSafetyProgram = 0x40232122
ReplacingExistingSafetyProgram = 0x492a282b
ReplacingExistingSafetyProgramWithNonSafetyProgram = 0x4a2c2b2d
LoadingSafetyProgram = 0x46292728,
```

7.16.4 ErrorCode

This enumeration lists all possible return values for a Result object.

```
OK
AccessDenied
ServiceTimeout
Disconnected
FailedToDisconnect
ServiceNotConnected
TooManySessions
SessionDelegitimated
NotChangableInRun
InvalidFileName
MultiESNotSupported
ServiceAborted
MultiESLimitExceeded
MultiESIncompatibleOtherESVersion
MultiESConflict
WriteProtected
DiskFull
InvalidVersion
Failed
CPUFailedToEnterRunMode
MACAddressIsNotValid
IPAddressIsNotValid
SubnetMaskIsNotValid
GatewayIsNotValid
ProfinetNameIsNotValid
NewIPAddressIsNotValid
NewSubnetMaskIsNotValid
NewGatewayIsNotValid
NewProfinetNameIsNotValid
InvalidPointer
SetIPErrorDueProjectSettings
UnsupportedDevice
SetNameErrorDueProjectSettings
OperationNotSupportedByThisDevice
DeviceNotOnNetwork
FirmwareVersionMatch
FirmwareFileNotCompatibleToNew
FirmwareFileNotCompatibleToOld
FirmwareFileNotCompatibleNotSame
FirmwareFileNotCompatibleSame
FirmwareFileNotCompatibleBuildType
FirmwareFileNotCompatible
FirmwareModuleNotReachable
FirmwareModuleNotAccepted
FirmwareIDNotFound
WriteBlockFailed
InvalidProjectVersion
DeviceIsNotAcceptingChanges
InvalidSignature
```

```
ParmeterOutOfRange
FailedToZipFolderContents
ErrorWritingToFile
ErrorCreatingFile
ErrorCreatingFolder
NoSATLicensePresent
InvalidTimeoutValue
NoDataToBackup
ErrorWritingToStream
ErrorReadingFromStream
InvalidProjectPath
ProjectNotCompatibleWithDevice
FailedToSetProfinetName
FailedToSetIPAddress
DownloadInvalidRecipe
IdentityFailure
DeviceMismatch
InvalidInterface
DeviceNotSelected
FailsafeAccessRequired
InternalApplicationError
InvalidPassword
DuplicateIPAddress
DuplicateProfinetName
SafetyDeviceMustBeConfirmed
NoSDCardPresent
InvalidProgramFolder
FSignaturesDoesNotMatch
FSignaturesMatch
DeviceDoesNotSupportProject
ProjectsUpdateIPNotReachable
RestoreIPNotReachable
ProjectIPNotUnique
SafetyProjectDownloadedToStandardNotAllowed
PasswordDiversityFailed
InvalidBackupFile
IncompatibleBackupFile
InvalidFirmwareFile
OperationWasNotSuccessful
CouldNotValidatePassword
IPAddressAlreadyExistsOnNetwork
MissingProgramFilePassword
InvalidProgramFilePassword
OperationCancelledByUser
InvalidProgramForDevice
InvalidProgramFilePasswordLegitimizationLevel
RestoreAuthenticationWarning
InvalidCPUPassword
MissingPAOM
DeviceNotFound
DeviceAlreadyExists
IPAddressAlreadyOnNetwork
```

```
ProfinetNameAlreadyOnNetwork
FailedToConnect
DeviceNotInitialized
CPUNewerVersionNotSupported
IPSuitNotValid
IPAddressChanged
ScanNoDevicesFound
DeviceCannotBeInserted
InsertDeviceDuplicateIP
InvalidImportLine
IPNotReachable
CouldNotReadFSignature
InvalidNetworkInterface
InsufficientLegitimizationLevel
NoProgramPassword
UnhandledExceptionOccured
ErrorLoadingOfflineDatabase
ProjectVersionV1NotSupported
ProjectOpenCanceled
ProgramPasswordNeeded
InvalidImportFile
FailsafeControlObjectNotFound
RestoreError
IncompatibleProgramFile
UnsupportedProgramFile
ProgramFileFamilyMismatch
DuplicateNewIPAddress
PLCSimAdvancedIsRunning
UnexpectedOperatingSystemError
ServiceActive
RemoteTransferDisabled
HardwareSoftwareNotComplete
LogicalVolumneMissing
LogicalVolumneOutOfSpace
Abort
FirwareTypeNotSupported
FirwareTypeNotInstalled
StoreReadFailed
StoreWriteFailed
RescueBackupNotPossible
RescueRestoreNotPossible
ConnectionRequired
ObjectNotFound
BufferToSmall
InvalidArguements
AttributeNotFound
InvalidPath
TypeConversionFailed
FileReadFailed
FileWriteFailed
OutOfResources
OutOfSpace
```

```
UnknownAddon
IncompatibleAddon
AddonsUnsupported
LicenseFailed
UnknownApp
UnknownAppAddon
UnknownReferenceApp
RuntimeMissing
RuntimeBroken
SignatureRequired
SignatureInvalid
SignatureFailure
CertificateInvalid
CertificateFailure
CertificateNotReady
CertificateExpired
CertificateRevoked
SecurityLib
WrongRuntimeVersion
MajorRuntimeDowngrade
MajorRuntimeUpgrade
MajorImageDowngrade
MajorImageUpgrade
WrongRuntime
NotEnoughMemory
ProjectCharacteristicsMissing
ProjectCharacteristicsInvalid
PanelOrientationIsPortrait
PanelOrientationIsLandscape
WrongDevicetype
NoRuntimeInstalled
RuntimeCorrupt
```

7.16.5 Language

The Language enumeration allows you to assign the language for returned string data. It contains the following values:

```
English
German
French
Spanish
Italian
```

7.16.6 OperatingState

This enumeration defines the possible states of the OperatingState property.

```
NotSupported
StopFwUpdate
StopSelfInitialization
Stop
Startup
Run
RunRedundant
Halt
LinkUp
Update
Defective
ErrorSearch
NoPower
CiR
STOPwithoutODIS
RunODIS
```

7.16.7 OperatingStateREQ

This enumeration defines the possible state transitions that can be requested, on a call to the SetOperatingState (Page 200) method.

```
Stop
Run
```

7.16.8 ProgressAction

This enumeration defines the possible argument values that can be sent to a ProgressChangedEventHandler (Page 168).

```
Invalid
Connecting
Reconnecting
Disconnecting
Initializing
Updating
Processing
Downloading
Uploading
Deleting
Reseting
Rebooting
Verifying
Formatting

Refreshing
Finished
UpdatingFirmware
InstallingRuntime
InstallingAddOns
UninstallingAddOns
UpdatingProgram
```

7.16.9 RemoteInterfaceType

This enumeration defines the possible states that can be returned from a call to the `InterfaceType` property on the IRemoteInterfaces (Page 211) interface.

```
None
Profinet
Profibus
ASi
```

7.16.10 FeatureSupport

The SIMATIC Automation Tool provides this enumeration to indicate what features each device supports.

```
Uninitialized
BackupAllowed
ChangeModeAllowed
FirmwareUpdateAllowed
MemoryResetAllowed
PasswordAllowed
ProgramUpdateAllowed
ResetToFactoryAllowed
FormatMCAllowed
NotFailsafe
RestoreAllowed
RemoteDataLogsAllowed
RemoteRecipesAllowed
Supported
FormatMCAllowed
Failsafe
ServiceDataAllowed
SetTimeAllowed
DiagBufferAllowed
```

To test whether a device supports a given feature, compare the value of the appropriate property with the `FeatureSupport` value defined for that feature.

For example, the following code checks to see if a device supports the Memory Reset feature, before attempting the operation.

```
Network myNetwork = new Network();
IProfinetDeviceCollection scannedDevices;

Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
if (retVal.Succeeded)
{
    List<ICPU> cpus = scannedDevices.FilterOnlyCpus();
    foreach (ICPU cpu in cpus)
    {
        if (cpu.MemoryResetAllowed)
        {

        }
    }
}
```

7.16.11 ProtectionLevel

The `ProtectionLevel` enumeration gives the protection level of a CPU password:

```
Unknown
Failsafe
Full
Read
HMI
NoAccess
NoPassword
```

7.16.12 ConfirmationType

This enumeration is used to indicate the status of Fail-Safe CPUs.

Description	Value
`Invalid`	0
`SafetyPasswordIsBeingUsed`	0x2f161717
`DeletingExistingSafetyProgram`	0x40232122
`ReplacingExistingSafetyProgram`	0x492a282b
`ReplacingExistingSafetyProgramWithNonSafetyProgram`	0x4a2c2b2d
`LoadingSafetyProgram`	0x46292728

7.16.13 FailsafeOperation

The `FailSafeOperation` enumeration indicates operations that are safety-relevant.

Description	Value
`Invalid`	0
`ResetToFactoryOperation`	0x2f161717
`FormatMCOperation`	0x46292728
`ProgramUpdateOperation`	0x43252224
`RestoreOperation`	0x45262427

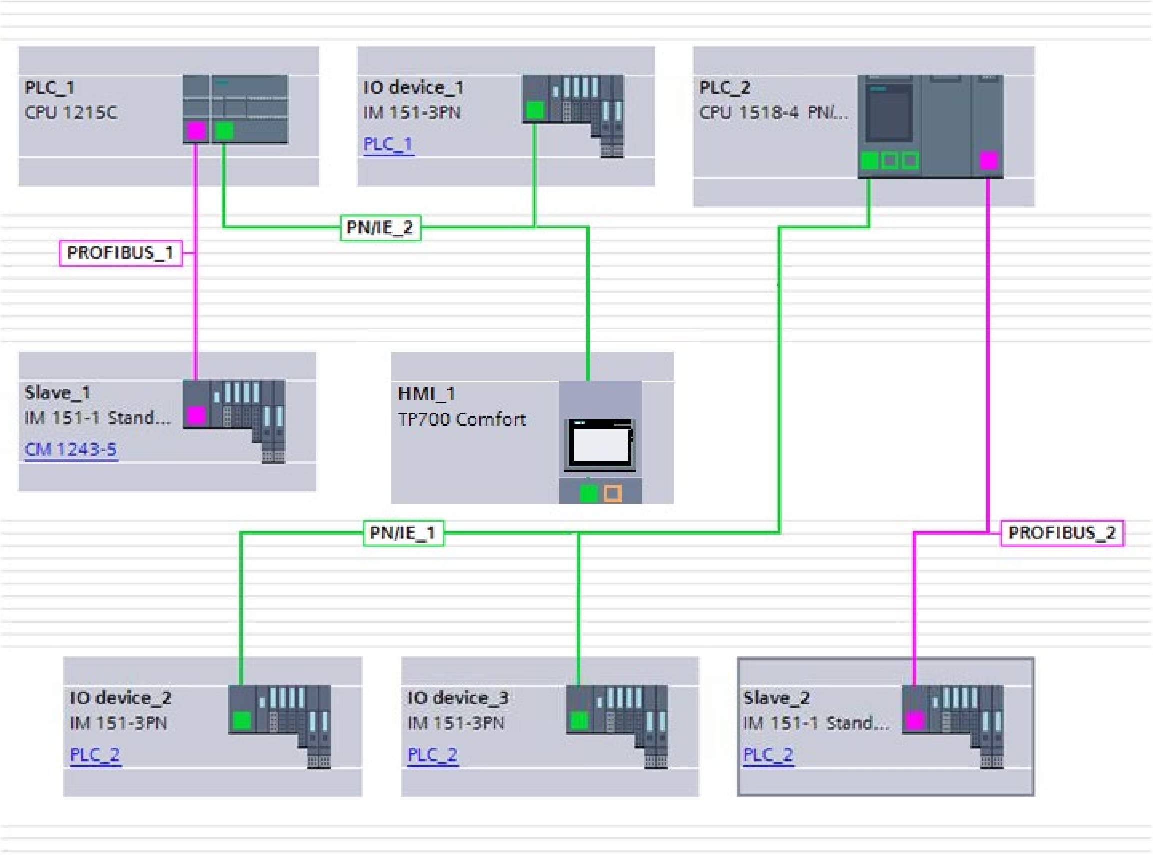
7.16.14 RemoteFolderType

The `RemoteFolderType` enumeration indicates the remote folder type.

Description	Value
`None`	0
`Recipe`	1
`Datalog`	2

7.17 Network example

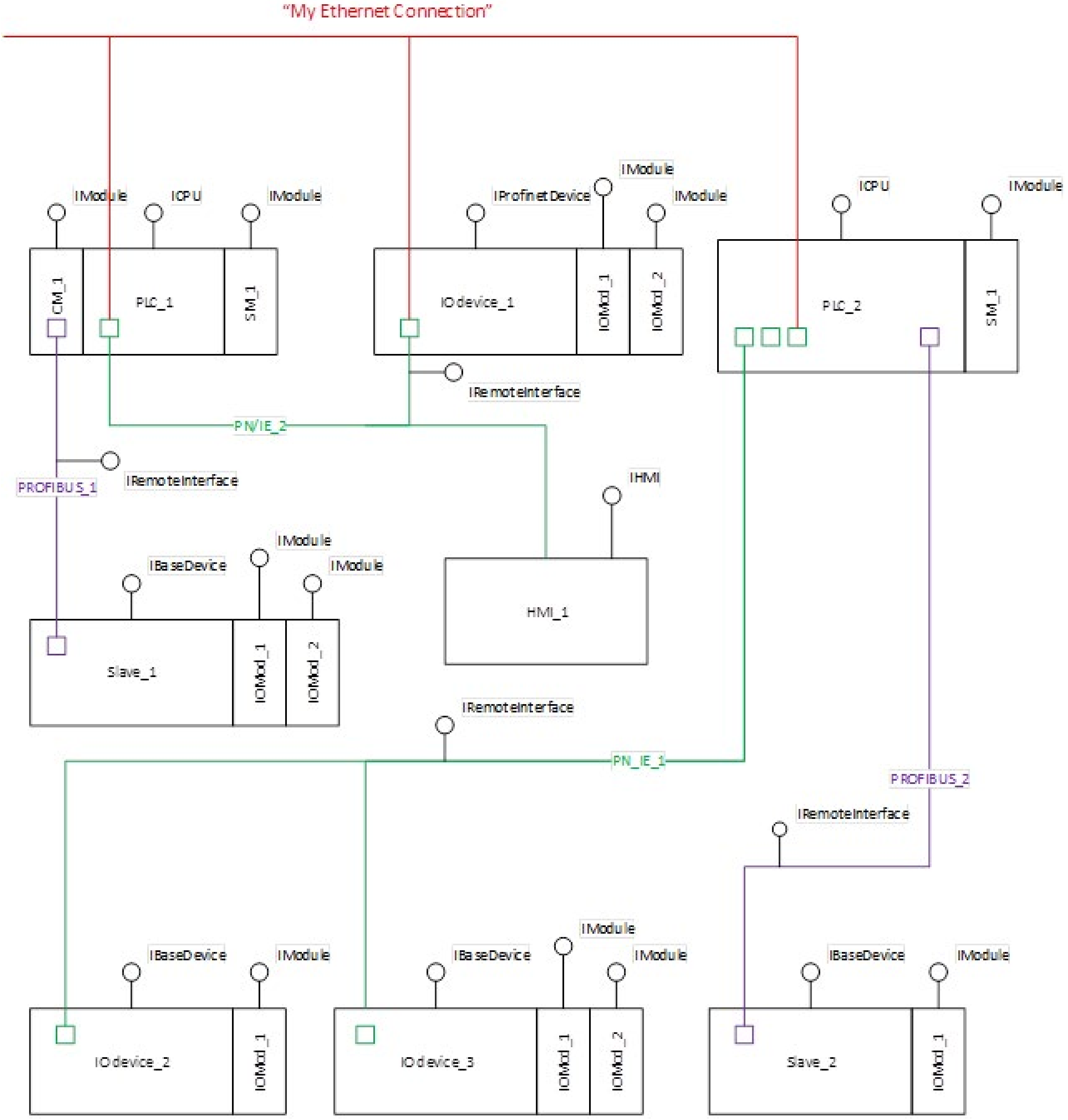
This example shows a TIA Portal network configuration and the API interfaces that represent the networked devices.



Assume that all the devices in the top row (PLC_1, IO device_1, and PLC_2) are connected to an external Ethernet network (not shown), and so can be directly accessed by the SIMATIC Automation Tool API. Further, assume that the PROFINET subnet connected to PLC_2 is not connected to the external network.

The SIMATIC Automation Tool API can provide information and operations for all the PLCs and I/O stations in this configuration.

The following diagram shows the same network configuration, and the hardware devices on the network.



In the diagram above, the "lollipop" notation shows which SIMATIC Automation Tool API interface class best represents each network component.

- CPUs directly connected to the external network are represented by the `ICPU` interface
- I/O Stations directly connected to the external network are represented by the `IProfinetDevice` interface

- Subnets originating from a CPU are represented by the `IRemoteInterface` interface
- I/O Stations not directly connected to the external network (but accessible through a CPU) are represented by the `IBaseDevice` interface.
- I/O or Communications modules connected to a CPU or IO Station are represented by the `IModule` interface

SIMATIC Automation Tool device support 8

8.1 Unrecognized firmware versions and devices

Supported Siemens devices

The device support tables show the supported Siemens devices, supported firmware versions, and supported tool operations.

Unsupported Siemens and non-Siemens devices

If you connect an unsupported device to your network, then there are two possibilities.

- The SIMATIC Automation Tool supports the Siemens device, but the firmware version is newer than the latest supported version.
  - The device is displayed in a Device table row
  - The supported device icon is displayed in the row
  - Tool operations are restricted to those operations that are supported in an earlier supported firmware version.
- The SIMATIC Automation Tool does not support the Siemens or non-Siemens device.
  - The device is displayed in a Device table row with partial information
  - The unsupported device question mark icon is displayed in the row
  - Only the MAC address based DCP operations work (Scan Entire Network, Identify, Update IP address, and Update PROFINET name) when the device is connected to the same subnet as the PG/PC running the SIMATIC Automation Tool.

8.2 ET 200

8.2.1 ET 200AL

8.2.1.1 ET 200AL IM support

ET 200AL IM operation support and firmware version

A check mark ✓ means that the operation is supported.

When the PROFINET column is supported, these PROFINET operations are supported:

- Scan for devices
- Identify device
- Set IP address
- Set PROFINET name

Article number	Module name	Firmware version	PROFINET	Factory reset	Firmware update
6ES7 157-1AA00-0AB0	IM 157-1 DP	V1.0			✓
6ES7 157-1AB00-0AB0	IM 157-1 PN	V1.0	✓	✓	✓

8.2.1.2 ET 200AL SM and IO-Link support

ET 200AL SM, IO-Link support and firmware version

A check mark (✓) means that the operation is supported.

Article number	Module name	Module type	Firmware version	Firmware update
6ES7 142-5AF00-0BA0	DQ 8x24VDC/2A 8xM12	SM	V1.0	✓
6ES7 143-5AF00-0BA0	DIQ 4+DQ 4x24VDC/0.5A 4xM12	SM	V1.0	✓
6ES7 143-5AH00-0BA0	DIQ 16x24VDC/0.5A 8xM12	SM	V1.0	✓
6ES7 143-5BF00-0BA0	DIQ 4+DQ 4x24VDC/0.5A 8xM8	SM	V1.0	✓
6ES7 144-5KD00-0BA0	AI 4xU/I/RTD 4xM12	SM	V1.0	✓
6ES7 145-5ND00-0BA0	AQ 4xU/I 4xM12	SM	V1.0	✓
6ES7 147-5JD00-0BA0	CM 4xIO-Link 4xM12	IO-Link	V1.0	✓

8.2.2 ET 200eco support

ET 200eco device operation support and firmware version

A check mark (✓) means that the operation is supported.

When the PROFINET column is supported, these PROFINET operations are supported:

- Scan for devices
- Identify device
- Set IP address
- Set PROFINET name

Article number	Module name	Firmware version	PROFINET	Factory reset	Firmware update
6ES7 141-6BF00-0AB0	8DI x 24VDC 4xM12	V6.0, 7.0	✓	✓	✓
6ES7 141-6BG00-0AB0	8DI x 24VDC 8xM12	V6.0, 7.0	✓	✓	✓
6ES7 141-6BH00-0AB0	16DI x 24VDC 8xM12	V6.0, 7.0	✓	✓	✓
6ES7 142-6BF00-0AB0	8DO x 24VDC / 1.3A 4xM12	V6.0, 7.0	✓	✓	✓
6ES7 142-6BF50-0AB0	8DO x 24VDC / 0.5A 4xM12	V6.0, 7.0	✓	✓	✓
6ES7 142-6BG00-0AB0	8DO x 24VDC / 1.3A 8xM12	V6.0, 7.0	✓	✓	✓
6ES7 142-6BH00-0AB0	16DO x 24VDC / 1.3A 8xM12	V6.0, 7.0	✓	✓	✓
6ES7 142-6BR00-0AB0	8DO x 24VDC / 2.0A 8xM12	V6.0, 7.0	✓	✓	✓
6ES7 144-6KD00-0AB0	8AI x 4U/I + 4RTD/TC 8 x M12	V6.0, 7.0	✓	✓	✓
6ES7 144-6KD50-0AB0	8AI x RTD/TC 8xM12	V7.0	✓	✓	✓
6ES7 145-6HD00-0AB0	4AO x 4U/I 4 x M12	V6.0, 7.0	✓	✓	✓
6ES7 147-6BG00-0AB0	8DI/8DO x 24VDC / 1.3A 8xM12	V6.0, 7.0	✓	✓	✓
6ES7 148-6JA00-0AB0	4IO-L + 8DI + 4DO x 24VDC / 1.3A 8xM12	V6.1, 7.0	✓	✓	✓
6ES7 148-6JD00-0AB0	4IO-L 4xM12	V1.0	✓	✓	✓

8.2.3 ET 200M IM support

ET 200M IM operation support and firmware version

A check mark ✓ means that the operation is supported.

When the PROFINET column is supported, these PROFINET operations are supported:

- Scan for devices
- Identify device
- Set IP address
- Set PROFINET name

Article number	Module name	Firmware version	PROFINET	Factory reset	Firmware update
6ES7 153-1AA03-0XB0	IM 153-1			✓	✓
6ES7 153-2BA02-0XB0	IM 153-2			✓	✓
6ES7 153-2BA10-0XB0	IM 153-2	V6.0		✓	✓
6ES7 153-2BA70-0XB0	IM 153-2 OD	V6.0		✓	✓
6ES7 153-2BA82-0XB0	IM 153-2 OD			✓	✓
6ES7 153-2BB00-0XB0	IM 153-2 FO			✓	✓
6ES7 153-4AA01-0XB0	IM 153-4 PN	V2.0, 3.0, 4.0	✓	✓	✓
6ES7 153-4BA00-0XB0	IM 153-4 PN	V3.0, 4.0	✓	✓	✓
6ES7 360-3AA01-0AA0	IM 360 IM S			✓	✓
6ES7 361-3CA01-0AA0	IM 361 IM R			✓	✓
6ES7 365-0BA01-0AA0	IM 365 IM S-R			✓	✓

8.2.4 ET 200MP IM support

ET 200MP IM operation support and firmware version

A check mark ✓ means that the operation is supported.

When the PROFINET column is supported, these PROFINET operations are supported:

- Scan for devices
- Identify device
- Set IP address
- Set PROFINET name

Article number	Module name	Firmware version	PROFINET	Factory reset	Firmware update
6AG1 155-5AA00-7AB0	IM 155-5 PN ST SIPLUS	V1.0, 2.0, 3.0	✓	✓	✓
6ES7 155-5AA00-0AA0	IM 155-5 PN BA	V4.0	✓	✓	✓
6ES7 155-5AA00-0AB0	IM 155-5 PN ST	V1.0, 2.0, 3.0	✓	✓	✓
6ES7 155-5AA00-0AC0	IM 155-5 PN HF	V1.0, 3.0	✓	✓	✓
6ES7 155-5BA00-0AB0	IM 155-5 DP ST	V2.0, 3.0			✓

8.2.5 ET 200S

ET 200S operation support and firmware version

A check mark ✓ means that the operation is supported.

When the PROFINET column is supported, these PROFINET operations are supported:

- Scan for devices
- Identify device
- Set IP address
- Set PROFINET name

Article number	Module name	Firmware version	PROFINET	Factory reset	Firmware update
6ES7 151-3AA22-0AB0	IM 151-3 PN	V5.0	✓	✓	✓
6ES7 151-3AA23-0AB0	IM 151-3 PN	V6.0, 6.1, 7.0	✓	✓	✓
6ES7 151-3BA22-0AB0	IM 151-3 PN	V5.0	✓	✓	✓
6ES7 151-3BA23-0AB0	IM 151-3 PN	V6.0, 6.1, 7.0	✓	✓	✓
6ES7 151-3BA60-0AB0	IM 151-3 PN	V3.0	✓	✓	✓
6ES7 151-3BB22-0AB0	IM 151-3 PN	V5.0	✓	✓	✓
6ES7 151-3BB23-0AB0	IM 151-3 PN	V6.1, 7.0	✓	✓	✓
6ES7 138-4FB04-0AB0	4 F-DO DC24V/2A				✓

Note

ET 200S CPU not supported

The ET 200S CPU is not supported by the SIMATIC Automation Tool

8.2.6 ET 200pro

8.2.6.1 ET 200pro CPU support (based 0n S7-1516)

ET 200pro CPU operation support and firmware version

A check mark (✓) means that the operation is supported. Standard CPU models have only the firmware version number in the column header. Fail-Safe CPU models have "Fail-Safe" in the column header.

			Fail-Safe	
CPU 1516pro-2 PN, CPU1516pro F-2 PN	**V2.0**	**V2.1**	**V2.0**	**V2.1**
Scan for devices	✓	✓	✓	✓
Identify device	✓	✓	✓	✓
Set IP address	✓	✓	✓	✓
Set PROFINET name	✓	✓	✓	✓
Put CPU in RUN/STOP	✓	✓	✓	✓
Set time to PG/PC time	✓	✓	✓	✓
Program update	✓	✓	✓	✓
Remote Recipe access	✓	✓	✓	✓
Remote Data Log access	✓	✓	✓	✓
Backup	✓	✓	✓	✓
Restore	✓	✓	✓	✓
Upload Service Data	✓	✓	✓	✓
Read Diagnostic buffer	✓	✓	✓	✓
Reset CPU memory	✓	✓	✓	✓
Reset to factory defaults	✓	✓	✓	✓
Format memory card	✓	✓	✓	✓
Firmware update	✓	✓	✓	✓

8.2.6.2 ET 200pro IM support

ET 200pro IM operation support and firmware version

A check mark ✓ means that the operation is supported.

When the PROFINET column is supported, these PROFINET operations are supported:

- Scan for devices
- Identify device
- Set IP address
- Set PROFINET name

Article number	Module name	Firmware version	PROFINET	Factory reset	Firmware update
6ES7 154-4AB10-0AB0	IM 154-4 Cu	V5.0, 6.0, 7.0, 7.1	✓		✓
6ES7 154-6AB00-0AB0	IM 154-6 IWLAN	V1.0	✓		✓
6ES7 154-6AB50-0AB0	IM 154-6 IWLAN	V1.0	✓		✓

8.2.6.3 ET 200pro IO-Link, RFID support

ET 200pro IO-Link, RFID support and firmware version

A check mark (✓) means that the operation is supported.

Article number	Module name	Module type	Firmware version	Firmware update
6ES7 147-4JD00-0AB0	CM 4xIO-Link 4xM12	IO-Link	V1.0	✓
6GT2 002-0HD00	RF170C	RFID	V1.0	✓
6GT2 002-0HD01	RF170C	RFID	V3.0	✓

8.2.7 ET 200SP

8.2.7.1 ET 200SP CPU support (based on S7-151x)

ET 200SP CPU operation support and firmware version

A check mark (✓) means that the operation is supported. Standard CPUs have only the firmware version number in the column header. Fail-Safe CPUs have "Fail-Safe" in the column header.

CPU 1510SP-1 PN **CPU 1510SP F-1 PN** **CPU 1512SP-1 PN** **CPU 1512SP F-1 PN**	**V1.6**	**V1.7**	**V1.8**	**V2.0**	**V2.1**	**Fail-Safe**				
						V1.6	**V1.7**	**V1.8**	**V2.0**	**V2.1**
Scan for devices	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Identify device	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Set IP address	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Set PROFINET name	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Change Run/STOP	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Set time to PG/PC time	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Program update	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Remote Recipe access	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Remote Data Log access	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Backup		✓	✓	✓	✓	✓	✓	✓	✓	✓
Restore		✓	✓	✓	✓	✓	✓	✓	✓	✓
Upload Service Data	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Read Diagnostic buffer	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Reset CPU memory	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Reset to factory values	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Format memory card	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Firmware update	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

8.2.7.2 ET 200SP IM and Server module support

ET 200SP IM and Server module support

A check mark ✓ means that the operation is supported.

When the PROFINET column is supported, these PROFINET operations are supported:

- Scan for devices
- Identify device
- Set IP address
- Set PROFINET name

Article number	Module name	Firmware versions	PROFINET	Factory reset	Firmware update
6AG1 155-6AU00-7BN0	IM 155-6 PN ST SIPLUS	V1.0, 1.1, 3.1, 3.3	✓	✓	✓
6AG1 193-6PA00-7AA0	Server module SIPLUS	V1.0, 1.1	✓		✓
6ES7 155-6AR00-0AN0	IM 155-6 PN BA	V3.2	✓	✓	✓
6ES7 155-6AU00-0BN0	IM 155-6 PN ST	V1.0, 1.1, 3.1, 3.3	✓	✓	✓
6ES7 155-6AU00-0CN0	IM 155-6 PN HF	V2.1, 2.2, 3.0, 3.1, 3.3	✓	✓	✓
6ES7 155-6AU00-0DN0	IM 155-6 PN HS	V4.0	✓	✓	✓
6ES7 155-6AU01-0BN0	IM 155-6 PN ST	V4.1	✓	✓	✓
6ES7 155-6BU00-0CN0	IM 155-6 DP HF	V1.1, 3.0, 3.1			✓
6ES7 193-6PA00-0AA0	Server module	V1.0, 1.1			✓

8.2.7.3 ET 200SP SM, AS-i, CM, CP, TM, IO-Link, Motorstarter support

ET 200SP SM, Motorstarter support and firmware version

A check mark (✓) means that the operation is supported.

Article number	Module name	Firmware version	Module type	Firmware update
3RK1 308-0AB00-0CP0	DS 0.3 - 1A HF 3DI/LC	V1.0, 1.1	Motorstarter	✓
3RK1 308-0AC00-0CP0	DS 0.9 - 3A HF 3DI/LC	V1.0, 1.1	Motorstarter	✓
3RK1 308-0AD00-0CP0	DS 2.8 - 9A HF 3DI/LC	V1.0, 1.1	Motorstarter	✓
3RK1 308-0AE00-0CP0	DS 4 - 12A HF 3DI/LC	V1.1	Motorstarter	✓
3RK1 308-0BB00-0CP0	RS 0.3- 1A HF 3DI/LC	V1.0, 1.1	Motorstarter	✓
3RK1 308-0BC00-0CP0	RS 0.9- 3A HF 3DI/LC	V1.0, 1.1	Motorstarter	✓
3RK1 308-0BD00-0CP0	RS 2.8- 9A HF 3DI/LC	V1.0, 1.1	Motorstarter	✓
3RK1 308-0BE00-0CP0	RS 4 - 12A HF 3DI/LC	V1.1	Motorstarter	✓
3RK1 308-0CB00-0CP0	F-DS 0.3 - 1A HF 3DI/LC	V1.0	Motorstarter	✓
3RK1 308-0CC00-0CP0	F-DS 0.9 - 3A HF 3DI/LC	V1.0	Motorstarter	✓
3RK1 308-0CD00-0CP0	F-DS 2.8 - 9A HF 3DI/LC	V1.0	Motorstarter	✓
3RK1 308-0CE00-0CP0	F-DS 4 - 12A HF 3DI/LC	V1.0	Motorstarter	✓
3RK1 308-0DB00-0CP0	F-RS 0.3 - 1A HF 3DI/LC	V1.0	Motorstarter	✓
3RK1 308-0DC00-0CP0	F-RS 0.9 - 3A HF 3DI/LC	V1.0	Motorstarter	✓
3RK1 308-0DD00-0CP0	F-RS 2.8 - 9A HF 3DI/LC	V1.0	Motorstarter	✓
3RK1 308-0DE00-0CP0	F-RS 4 - 12A HF 3DI/LC	V1.0	Motorstarter	✓
3RK7 136-6SC00-0BC1	F-CM AS-i Safety ST	V1.0	ASi	✓
3RK7 137-6SA00-0BC1	CM AS-i Master ST	V1.0, 1.1	CP	✓
6AG1 131-6BF00-7BA0	DI 8x24VDC ST SIPLUS	V1.0, 1.1	SM	✓
6AG1 131-6BH00-7BA0	DI 16x24VDC ST SIPLUS	V1.0	SM	✓
6AG1 132-6BD20-7BA0	DQ 4x24VDC/2A ST SIPLUS	V1.0, 1.1	SM	✓
6AG1 132-6BF00-7BA0	DQ 8x24VDC/0.5A ST SIPLUS	V1.0, 1.1	SM	✓
6AG1 132-6BD20-7BA0	DQ 4x24VDC/2A ST SIPLUS	V1.0, 1.1	SM	✓
6AG1 132-6BF00-7BA0	DQ 8x24VDC/0.5A ST SIPLUS	V1.0, 1.1	SM	✓
6AG1 132-6BH00-7BA0	DQ 16x24VDC/0.5A ST SIPLUS	V1.0	SM	✓
6AG1 134-6GD00-7BA1	AI 4xI 2- 4-wire ST SIPLUS	V1.0, 1.1	SM	✓
6AG1 134-6HD00-7BA1	AI 4xU/I 2-wire ST SIPLUS	V1.0, 1.1	SM	✓
6AG1 134-6JD00-2CA1	AI 4xRTD/TC 2- 3- 4-wire HF SIPLUS	V1.0, 1.1, 2.0	SM	✓
6AG1 135-6HD00-7BA1	AQ 4xU/I ST SIPLUS	V1.0, 1.1	SM	✓
6ES7 131-6BF00-0AA0	DI 8x24VDC BA	V1.0	SM	✓
6ES7 131-6BF00-0BA0	DI 8x24VDC ST	V1.0, 1.1	SM	✓
6ES7 131-6BF00-0CA0	DI 8x24VDC HF	V1.0, 1.1, 1.2, 2.0	SM	✓
6ES7 131-6BF00-0DA0	DI 8x24VDC HS	V1.0	SM	✓
6ES7 131-6BF60-0AA0	DI 8x24VDC SRC BA	V1.0	SM	✓
6ES7 131-6BH00-0BA0	DI 16x24VDC ST	V1.0, 1.1	SM	✓
6ES7 131-6FD00-0BB1	DI 4x120..230VAC ST	V1.0	SM	✓
6ES7 131-6TF00-0CA0	DI 8xNAMUR HF	V1.0	SM	✓
6ES7 132-6BD20-0BA0	DQ 4x24VDC/2A ST	V1.0, 1.1	SM	✓
6ES7 132-6BD20-0CA0	DQ 4x24VDC/2A HF	V1.0, 2.0	SM	✓
6ES7 132-6BD20-0DA0	DQ 4x24VDC/2A HS	V1.0	SM	✓

Article number	Module name	Firmware version	Module type	Firmware update
6ES7 132-6BF00-0AA0	DQ 8x24VDC/0.5A BA	V1.0	SM	✓
6ES7 132-6BF00-0BA0	DQ 8x24VDC/0.5A ST	V1.0, 1.1	SM	✓
6ES7 132-6BF00-0CA0	DQ 8x24VDC/0.5A HF	V1.0, 1.1, 1.2, 2.0	SM	✓
6ES7 132-6BF60-0AA0	DQ 8x24VDC/0.5A SNK BA	V1.0	SM	✓
6ES7 132-6BH00-0BA0	DQ 16x24VDC/0.5A ST	V1.0, 1.1	SM	✓
6ES7 132-6FD00-0BB1	DQ 4x24..230VAC/2A ST	V1.0	SM	✓
6ES7 132-6GD50-0BA0	RQ 4x24VUC/2A CO ST	V1.0	SM	✓
6ES7 132-6HD00-0BB0	RQ 4x120VDC/230VAC/5A NO ST	V1.0, 1.1	SM	✓
6ES7 132-6HD00-0BB1	RQ 4x120VDC/230VAC/5A NO ST	V1.1	SM	✓
6ES7 132-6MD00-0BB1	RQ 4x120VDC/230VAC/5A NO MA ST	V1.0	SM	✓
6ES7 134-6FB00-0BA1	AI 2xU ST	V1.0	SM	✓
6ES7 134-6FF00-0AA1	AI 8xU BA	V1.0	SM	✓
6ES7 134-6GB00-0BA1	AI 2xI 2- 4-wire ST	V1.0	SM	✓
6ES7 134-6GD00-0BA1	AI 4xI 2- 4-wire ST	V1.0, V1.1	SM	✓
6ES7 134-6GF00-0AA1	AI 8xI 2- 4-wire BA	V1.0	SM	✓
6ES7 134-6HB00-0CA1	AI 2xU/I 2- 4-wire HF	V1.0, 2.0	SM	✓
6ES7 134-6HB00-0DA1	AI 2xU/I 2- 4-wire HS	V1.0, 1.1, 2.0	SM	✓
6ES7 134-6HD00-0BA1	AI 4xU/I 2-wire ST	V1.0, 1.1	SM	✓
6ES7 134-6JD00-0CA1	AI 4xRTD/TC 2- 3- 4-wire HF	V1.0, 1.1, 2.0	SM	✓
6ES7 134-6JF00-0CA1	AI 8xRTD/TC 2-wire HF	V2.0	SM	✓
6ES7 134-6PA00-0BD0	AI EnergyMeter ST	V1.0, 2.0	SM	✓
6ES7 134-6PA01-0BD0	AI EnergyMeter 400VAC ST	V3.0	SM	✓
6ES7 134-6PA20-0BD0	AI EnergyMeter 480VAC ST	V4.0	SM	✓
6ES7 134-6TD00-0CA1	AI 4xI 2-wire 4..20mA HART	V1.0	SM	✓
6ES7 135-6FB00-0BA1	AQ 2xU ST	V1.0	SM	✓
6ES7 135-6GB00-0BA1	AQ 2xI ST	V1.0	SM	✓
6ES7 135-6HB00-0CA1	AQ 2xU/I HF	V1.0	SM	✓
6ES7 135-6HB00-0DA1	AQ 2xU/I HS	V1.0, 1.1, 2.0	SM	✓
6ES7 135-6HD00-0BA1	AQ 4xU/I ST	V1.0, 1.1	SM	✓
6ES7 136-6BA00-0CA0	F-DI 8x24VDC HF	V1.0	SM	✓
6ES7 136-6DB00-0CA0	F-DQ 4x24VDC/2A PM HF	V1.0	SM	✓
6ES7 136-6DC00-0CA0	F-DQ 8x24VDC/0.5A PP HF	V1.0	SM	✓
6ES7 136-6PA00-0BC0	F-PM-E 24VDC/8A PPM ST	V1.0	PM	✓
6ES7 136-6RA00-0BF0	F-RQ 1x24..48VDC/24..230VAC/5A	V1.0	SM	✓
6ES7 137-6AA00-0BA0	Point-to-point	V1.0	CM	✓
6ES7 137-6BD00-0BA0	CM 4xIO-Link	V1.0, 2.0, 2.1	IO-Link	✓
6ES7 138-6AA00-0BA0	TM Count 1x24V	V1.0, 1.1, 1.2	TM	✓
6ES7 138-6BA00-0BA0	TM PosInput 1	V1.0, 1.1, 1.2	TM	✓
6ES7 138-6CG00-0BA0	TM Timer DIDQ 10x24V	V1.0	TM	✓
6ES7 138-6DB00-0BB1	TM Pulse 2x24V	V1.0	TM	✓
6ES7 545-5DA00-0AB0	CM DP		CP	✓
6GK7 542-6UX00-0XE0	CP 1542SP-1	V1.0	CP	✓
6GK7 542-6VX00-0XE0	CP 1542SP-1 IRC	V1.0	CP	✓
6GK7 543-6WX00-0XE0	CP 1543SP-1	V1.0	CP	✓
7MH4 138-6AA00-0BA0	SIWAREX WP321	V1.0	TM	✓

8.3 S7-1200

8.3.1 S7-1200 CPU support

S7-1200 operation support and firmware version

A check mark (✓) means that the operation is supported. Standard CPUs have only the firmware version number in the column header. Fail-Safe CPUs have "Fail-Safe" in the column header.

	V1.x	V2.x	V3.x	V4.0	V4.1	V4.2	Fail-Safe	
							V4.1	V4.2
Scan for devices		✓	✓	✓	✓	✓	✓	✓
Identify device		✓	✓	✓	✓	✓	✓	✓
Set IP address		✓	✓	✓	✓	✓	✓	✓
Set PROFINET name		✓	✓	✓	✓	✓	✓	✓
Change Run/STOP		✓	✓	✓	✓	✓	✓	✓
Set time to PG/PC time		✓	✓	✓	✓	✓	✓	✓
Program update		✓	✓	✓	✓	✓		✓
Remote Recipe access					✓	✓	✓	✓
Remote Data Log access					✓	✓	✓	✓
Backup						✓		✓
Restore						✓		✓
Upload Service Data		✓	✓	✓	✓	✓	✓	✓
Read Diagnostic buffer		✓	✓	✓	✓	✓	✓	✓
Reset CPU memory		✓	✓	✓	✓	✓	✓	✓
Reset to factory values		✓	✓	✓	✓	✓		✓
Format memory card						✓	✓	✓
Firmware update				✓	✓	✓	✓	✓

8.3.2 S7-1200 I/O and other module support

D I/O, A I/O, SB, CM, CP, and IO link

A check mark (✓) means that the operation is supported.

Article number	Module name	Firmware version	Module type	Firmware update
021620-B	CM CANopen	V1.0	CM	✓
021730-B	CM CANopen (Ruggedized)	V1.0	CM	✓
3RK7243-2AA30-0XB0	CM 1243-2	V1.1	CM	✓
6AG1 221-1BF32-2XB0	DI 8x24VDC SIPLUS	V2.0	SM	✓
6AG1 221-1BF32-4XB0	DI 8x24VDC SIPLUS	V2.0	SM	✓
6AG1 221-1BH32-2XB0	DI 16x24VDC SIPLUS	V2.0	SM	✓
6AG1 221-1BH32-4XB0	DI 16x24VDC SIPLUS	V2.0	SM	✓
6AG1 221-3AD30-5XB0	DI 4x5VDC SIPLUS	V1.0	SignalBoard	
6AG1 221-3BD30-5XB0	DI 4x24VDC SIPLUS	V1.0	SignalBoard	
6AG1 222-1AD30-5XB0	DQ 4x5VDC SIPLUS	V1.0	SignalBoard	
6AG1 222-1BD30-5XB0	DQ 4x24VDC SIPLUS	V1.0	SignalBoard	
6AG1 222-1BF32-2XB0	DQ 8x24VDC SIPLUS	V2.0	SM	✓
6AG1 222-1BF32-4XB0	DQ 8x24VDC SIPLUS	V2.0	SM	✓
6AG1 222-1BH32-2XB0	DQ 16x24VDC SIPLUS	V2.0	SM	✓
6AG1 222-1BH32-4XB0	DQ 16x24VDC SIPLUS	V2.0	SM	✓
6AG1 222-1HF32-2XB0	DQ 8xRelay SIPLUS	V2.0	SM	✓
6AG1 222-1HF32-4XB0	DQ 8xRelay SIPLUS	V2.0	SM	✓
6AG1 222-1HH32-2XB0	DQ 16xRelay SIPLUS	V2.0	SM	✓
6AG1 222-1HH32-4XB0	DQ 16xRelay SIPLUS	V2.0	SM	✓
6AG1 222-1XF32-2XB0	DQ 8xNO/NC Relay SIPLUS	V2.0	SM	✓
6AG1 222-1XF32-4XB0	DQ 8xNO/NC Relay SIPLUS	V2.0	SM	✓
6AG1 223-0BD30-4XB0	DI 2/DQ 2x24VDC SIPLUS	V1.0	SignalBoard	
6AG1 223-0BD30-5XB0	DI 2/DQ 2x24VDC SIPLUS	V1.0	SignalBoard	
6AG1 223-1BH32-2XB0	DI 8/DQ 8x24VDC SIPLUS	V2.0	SM	✓
6AG1 223-1BH32-4XB0	DI 8/DQ 8x24VDC SIPLUS	V2.0	SM	✓
6AG1 223-1BL32-2XB0	DI 16/DQ 16x24VDC SIPLUS	V2.0	SM	✓
6AG1 223-1BL32-4XB0	DI 16/DQ 16x24VDC SIPLUS	V2.0	SM	✓
6AG1 223-1PH32-2XB0	DI 8x24VDC/DQ 8xRelay SIPLUS	V2.0	SM	✓
6AG1 223-1PH32-4XB0	DI 8x24VDC/DQ 8xRelay SIPLUS	V2.0	SM	✓
6AG1 223-1PL32-2XB0	DI 16x24VDC/DQ 16xRelay SIPLUS	V2.0	SM	✓
6AG1 223-1PL32-4XB0	DI 16x24VDC/DQ 16xRelay SIPLUS	V2.0	SM	✓
6AG1 223-1QH32-2XB0	DI/DQ 8x120VAC/DQ 8xRelay SIPLUS	V2.0	SM	✓
6AG1 223-1QH32-4XB0	DI/DQ 8x120VAC/DQ 8xRelay SIPLUS	V2.0	SM	✓
6AG1 223-3AD30-5XB0	DI 2/DQ 2x5VDC SIPLUS	V1.0	SignalBoard	
6AG1 223-3BD30-5XB0	DI 2/DQ 2x24VDC SIPLUS	V1.0	SignalBoard	
6AG1 231-4HD32-4XB0	AI 4x13BIT SIPLUS	V2.0	SM	✓
6AG1 231-4HF32-4XB0	AI 8x13BIT SIPLUS	V2.0	SM	✓
6AG1 231-5ND32-4XB0	AI 4x16BIT SIPLUS	V2.0	SM	✓
6AG1 231-5PD32-2XB0	AI 4xRTD SIPLUS	V2.0	SM	✓
6AG1 231-5PD32-4XB0	AI 4xRTD SIPLUS	V2.0	SM	✓

Article number	Module name	Firmware version	Module type	Firmware update
6AG1 231-5PF32-2XB0	AI 8xRTD SIPLUS	V2.0	SM	✓
6AG1 231-5PF32-4XB0	AI 8xRTD SIPLUS	V2.0	SM	✓
6AG1 231-5QD32-4XB0	AI 4xTC SIPLUS	V2.0	SM	✓
6AG1 231-5QF32-4XB0	AI 8xTC SIPLUS	V2.0	SM	✓
6AG1 232-4HA30-4XB0	AQ 1x12BIT SIPLUS	V1.0	SignalBoard	
6AG1 232-4HA30-5XB0	AQ 1x12BIT SIPLUS	V1.0	SignalBoard	
6AG1 232-4HB32-4XB0	AQ 2x13BIT SIPLUS	V2.0	SM	✓
6AG1 232-4HD32-2XB0	AQ 4x14BIT SIPLUS	V2.0	SM	✓
6AG1 232-4HD32-4XB0	AQ 4x14BIT SIPLUS	V2.0	SM	✓
6AG1 234-4HE32-2XB0	AI 4x13BIT/AQ 2x14BIT SIPLUS	V2.0	SM	✓
6AG1 234-4HE32-4XB0	AI 4x13BIT/AQ 2x14BIT SIPLUS	V2.0	SM	✓
6AG1 241-1AH32-2XB0	CM 1241 (RS232) SIPLUS	V2.1, V2.2	CM	✓
6AG1 241-1AH32-4XB0	CM 1241 (RS232) SIPLUS	V2.1, V2.2	CM	✓
6AG1 241-1CH30-5XB1	CB 1241 (RS485) SIPLUS	V1.0	CommunicationBoard	
6AG1 241-1CH32-2XB0	CM 1241 (RS422/485) SIPLUS	V2.1	CM	✓
6AG1 241-1CH32-4XB0	CM 1241 (RS422/485) SIPLUS	V2.1	CM	✓
6AG1 242-5DX30-2XE0	CM 1242-5 SIPLUS	V1.0	CM	
6AG1 242-7KX30-4XE0	CP 1242-7 GPRS SIPLUS	V1.4	CM	
6AG1 243-1JX30-7XE0	CP 1243-1 DNP3 SIPLUS	V1.1	CP	✓
6AG1 243-5DX30-2XE0	CM 1243-5 SIPLUS	V1.3	CM	
6AG1 278-4BD32-2XB0	4SI IO link SIPLUS	V2.0	SM	✓
6AG1 278-4BD32-4XB0	4SI IO link SIPLUS	V2.0	SM	✓
6AT8 007-1AA10-0AA0	SM 1281 Condition Monitoring	V1.0	SM	✓
6ES7 221-1BF30-0XB0	DI 8x24VDC	V1.0	SM	
6ES7 221-1BF32-0XB0	DI 8x24VDC	V2.0	SM	✓
6ES7 221-1BH30-0XB0	DI 16x24VDC	V1.0	SM	
6ES7 221-1BH32-0XB0	DI 16x24VDC	V2.0	SM	✓
6ES7 221-3AD30-0XB0	DI 4x5VDC	V1.0	SignalBoard	
6ES7 221-3BD30-0XB0	DI 4x24VDC	V1.0	SignalBoard	
6ES7 222-1AD30-0XB0	DQ 4x5VDC	V1.0	SignalBoard	
6ES7 222-1BD30-0XB0	DQ 4x24VDC	V1.0	SignalBoard	
6ES7 222-1BF30-0XB0	DQ 8x24VDC	V1.0	SM	
6ES7 222-1BF32-0XB0	DQ 8x24VDC	V2.0	SM	✓
6ES7 222-1BH30-0XB0	DQ 16x24VDC	V1.0	SM	
6ES7 222-1BH32-0XB0	DQ 16x24VDC	V2.0	SM	✓
6ES7 222-1HF30-0XB0	DQ 8xRelay	V1.0	SM	
6ES7 222-1HF32-0XB0	DQ 8xRelay	V2.0	SM	✓
6ES7 222-1HH30-0XB0	DQ 16xRelay	V1.0	SM	
6ES7 222-1HH32-0XB0	DQ 16xRelay	V2.0	SM	✓
6ES7 222-1XF30-0XB0	DQ 8xNO/NC Relay	V1.0	SM	
6ES7 222-1XF32-0XB0	DQ 8xNO/NC Relay	V2.0	SM	✓
6ES7 223-0BD30-0XB0	DI 2/DQ 2x24VDC	V1.0	SignalBoard	
6ES7 223-1BH30-0XB0	DI 8/DQ 8x24VDC	V1.0	SM	
6ES7 223-1BH32-0XB0	DI 8/DQ 8x24VDC	V2.0	SM	✓
6ES7 223-1BL30-0XB0	DI 16/DQ 16x24VDC	V1.0	SM	

Article number	Module name	Firmware version	Module type	Firmware update
6ES7 223-1BL32-0XB0	DI 16/DQ 16x24VDC	V2.0	SM	✓
6ES7 223-1PH30-0XB0	DI 8x24VDC/DQ 8xRelay	V1.0	SM	
6ES7 223-1PH32-0XB0	DI 8x24VDC/DQ 8xRelay	V2.0	SM	✓
6ES7 223-1PL30-0XB0	DI 16x24VDC/DQ 16xRelay	V1.0	SM	
6ES7 223-1PL32-0XB0	DI 16x24VDC/DQ 16xRelay	V2.0	SM	✓
6ES7 223-1QH30-0XB0	DI/DO 8x120VAC/DQ 8xRelay	V1.0	SM	
6ES7 223-1QH32-0XB0	DI/DO 8x120VAC/DQ 8xRelay	V2.0	SM	✓
6ES7 223-3AD30-0XB0	DI 2/DQ 2x5VDC	V1.0	SignalBoard	
6ES7 223-3BD30-0XB0	DI 2/DQ 2x24VDC	V1.0	SignalBoard	
6ES7 226-6BA32-0XB0	F-DI 8/16x24VDC	V2.0	SM	✓
6ES7 226-6DA32-0XB0	F-DQ 4x24VDC	V2.0	SM	✓
6ES7 226-6RA32-0XB0	F-DQ 2xRelay	V2.0	SM	✓
6ES7 228-1RC51-0AA0	Power Signal Booster Carrier Module	V2.0, V2.2	SM	
6ES7 228-1RC52-0AA0	Power Signal Booster Segment Module	V2.0, V2.2	SM	
6ES7 231-4HA30-0XB0	AI 1x12BIT	V1.0, V2.0	SignalBoard	
6ES7 231-4HD30-0XB0	AI 4x13BIT	V1.0	SM	
6ES7 231-4HD32-0XB0	AI 4x13BIT	V2.0	SM	✓
6ES7 231-4HF30-0XB0	AI 8x13BIT	V1.0	SM	
6ES7 231-4HF32-0XB0	AI 8x13BIT	V2.0	SM	✓
6ES7 231-5ND30-0XB0	AI 4x16BIT	V1.0	SM	
6ES7 231-5ND32-0XB0	AI 4x16BIT	V2.0	SM	✓
6ES7 231-5PA30-0XB0	AI 1xRTD	V1.0, V2.0	SignalBoard	
6ES7 231-5PD30-0XB0	AI 4xRTD	V1.0	SM	
6ES7 231-5PD32-0XB0	AI 4xRTD	V2.0	SM	✓
6ES7 231-5PF30-0XB0	AI 8xRTD	V1.0	SM	
6ES7 231-5PF32-0XB0	AI 8xRTD	V2.0	SM	✓
6ES7 231-5QA30-0XB0	AI 1xTC	V1.0, V2.0	SignalBoard	
6ES7 231-5QD30-0XB0	AI 4xTC	V1.0	SM	
6ES7 231-5QD32-0XB0	AI 4xTC	V2.0	SM	✓
6ES7 231-5QF30-0XB0	AI 8xTC	V1.0	SM	
6ES7 231-5QF32-0XB0	AI 8xTC	V2.0	SM	✓
6ES7 232-4HA30-0XB0	AQ 1x12BIT	V1.0	SignalBoard	
6ES7 232-4HB30-0XB0	AQ 2x14BIT	V1.0	SM	
6ES7 232-4HB32-0XB0	AQ 2x14BIT	V2.0	SM	✓
6ES7 232-4HD30-0XB0	AQ 4x14BIT	V1.0	SM	
6ES7 232-4HD32-0XB0	AQ 4x14BIT	V2.0	SM	✓
6ES7 234-4HE30-0XB0	AI 4x13BIT/AQ 2x14BIT	V1.0	SM	
6ES7 234-4HE32-0XB0	AI 4x13BIT/AQ 2x14BIT	V2.0	SM	✓
6ES7 238-5XA32-0XB0	AI Energy Meter	V2.0	SM	✓
6ES7 241-1AH30-0XB0	CM 1241 (RS232)	V1.0	CM	
6ES7 241-1AH32-0XB0	CM 1241 (RS232)	V2.0, 2,1, 2.2	CM	✓
6ES7 241-1CH30-0XB0	CM 1241 (RS485)	V1.0	CM	
6ES7 241-1CH30-1XB0	CB 1241 (RS485)	V1.0	Communica-tionBoard	
6ES7 241-1CH31-0XB0	CM 1241 (RS422/485)	V1.0	CM	
6ES7 241-1CH32-0XB0	CM 1241 (RS422/485)	V2.0. 2.1, 2.2	CM	✓

Article number	Module name	Firmware version	Module type	Firmware update
6ES7 278-4BD32-0XB0	4SI IO link	V2.0	SM	✓
6ES7 972 0MD00 0XA0	TS Module ISDN	V1.0	CP	✓
6ES7 972 0MG00 0XA0	TS Module GSM	V1.0	CP	✓
6ES7 972 0MM00 0XA0	TS Module Modem	V1.0	CP	✓
6ES7 972 0MS00 0XA0	TS Module RS232	V1.0	CP	✓
6GK7 242-5DX30-0XE0	CM 1242-5	V1.0	CM	
6GK7 242-7KX30-0XE0	CP 1242-7	V1.0, V1.3, V1.4	CM	
6GK7 242-7KX31-0XE0	CP 1242-7	V2.1, V3.0	CP	✓
6GK7 243-1BX30-0XE0	CP 1243-1	V2.0, V2.1, 3.0	CP	✓
6GK7 243-1HX30-0XE0	CP 1243-1 PCC	V2.0	CP	✓
6GK7 243-1JX30-0XE0	CP 1243-1 DNP3	V1.0. V1.1	CP	✓
6GK7 243-1PX30-0XE0	CP 1243-1 IEC	V1.1. V1.2	CP	✓
6GK7 243-7KX30-0XE0	CP 1243-7 LTE	V2.1, V3.0	CP	✓
6GK7 243-7SX30-0XE0	CP 1243-7 LTE	V2.1, V3.0	CP	✓
6GK7 243-8RX30-0XE0	CP 1243-8 IRC	V2.1, V3.0	CP	✓
6GT2 002-0LA00	RFC120C	V1.0	CM	
7MH4 960-2AA01	SIWAREX WP231		TM	
7MH4 960-4AA01	SIWAREX WP241		TM	
7MH4 960-6AA01	SIWAREX WP251		TM	

8.4 S7-1500

8.4.1 S7-1500 CPU support

S7-1500 operation support and firmware version

A check mark (✓) means that the operation is supported. Standard CPU models have only the firmware version number in the column header. Fail-Safe CPU models have "Fail-Safe" in the column header.

									Fail-Safe					
S7-1500	**V1.0**	**V1.1**	**V1.5**	**V1.6**	**V1.7**	**V1.8**	**V2.0**	**V2.1**	**V1.5**	**V1.6**	**V1.7**	**V1.8**	**V2.0**	**V2.1**
Scan for devices	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Identify device	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Set IP address	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Set PROFINET name	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Put CPU in RUN/STOP	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Set time to PG/PC time	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Program update	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓
Remote Recipe access	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Remote Data Log access	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Backup					✓	✓	✓	✓			✓	✓	✓	✓
Restore					✓	✓	✓	✓			✓	✓	✓	✓
Upload Service Data	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Read Diagnostic buffer	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Reset CPU memory	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Reset to factory defaults	✓	✓	✓	✓	✓	✓	✓	✓			✓	✓	✓	✓
Format memory card				✓	✓	✓	✓	✓			✓	✓	✓	✓
Firmware update	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Note

S7-1500 ODK (Open development kit) CPUs are not supported by the SIMATIC Automation Tool

Not supported:

6ES7 518-4AP00-3AB0 CPU 1518-4 PN/DP ODK

6ES7 518-4FP00-3AB0 CPU 1518F-4 PN/DP ODK

6ES7 518-4FP00-3AB0 CPU 1518F-4 PN/DP ODK

6ES7 518-4FP00-3AB0 CPU 1518F-4 PN/DP ODK

8.4.2 S7-1500 I/O and other module support

D I/O, A I/O, CP, TM, and PM

A check mark (✓) means that the operation is supported.

Device firmware version and firmware update operation support

S7-1500 power supply modules		Firmware update operation
6EP1 332-4BA00	PM 70W 120/230VAC	✓
6EP1333-4BA00	PM 190W 120/230VAC	✓

S7-1500 TIM module		Firmware update operation
6GK7 543-1MX00-0XE0	TIM 1531 IRC	✓ for V0.1, V1.0

S7-1500 I/O and other modules	V1.x	V2.x
Firmware update operation	✓	✓

8.5 SIMATIC HMI (Human Machine Interface)

8.5.1 HMI Basic panels support

The following SIMATIC HMI device groups are supported by the SIMATIC Automation Tool.

SIMATIC HMI panel group	Supported panel models (PROFINET models only)
Basic	KTP400 Basic
	KTP700 Basic
	KTP900 Basic
	KTP1200 Basic

HMI panel firmware version and supported operations

Operation	HMI firmware version Greater than or equal to V13.0.0.0
Program update	✓ for firmware version greater than or equal to V13.0.0.0 SIMATIC Automation Tool can update HMI device operating system and runtime software with the Program update operation.
Backup	✓ for firmware version greater than or equal to V13.0.1.0
Restore	✓ for firmware version greater than or equal to V13.0.1.0

The SIMATIC Automation Tool supports the PROFINET DCP operations:

- Scan for devices
- Identify device
- Set IP address
- Set PROFINET name

8.5.2 HMI Comfort panels support

The following SIMATIC HMI device groups are supported by the SIMATIC Automation Tool.

SIMATIC HMI panel group	Supported panel models
Comfort	KP400, KTP400 Comfort
	KP700, TP700 Comfort
	KP900, TP900 Comfort
	KP1200, TP1200 Comfort
	KP1500, TP1500 Comfort
	TP1900 Comfort
	TP2200 Comfort

A check mark (✓) means that the operation is supported.

Operation	HMI firmware version
Program update	✓ for firmware version greater than or equal to V13.0.0.0 SIMATIC Automation Tool Program update for HMI devices can update HMI firmware, operating system, and run-time project data.
Backup	✓ for firmware version greater than or equal to V13.0.1.0
Restore	✓ for firmware version greater than or equal to V13.0.1.0

The SIMATIC Automation Tool supports the PROFINET DCP operations:

- Scan for devices
- Identify device
- Set IP address
- Set PROFINET name

8.5.3 HMI Mobile panels support

The following SIMATIC HMI device groups are supported by the SIMATIC Automation Tool.

SIMATIC HMI panel group	Supported panel models
Mobile	KTP700 Mobile
	KTP900 Mobile

HMI panel firmware version and supported operations

Operation	HMI firmware version Greater than or equal to V13.0.1.0
Program update	✓ SIMATIC Automation Tool can update HMI device operating system and runtime software with the Program update operation.
Backup	✓
Restore	✓

The SIMATIC Automation Tool supports the PROFINET DCP operations:

- Scan for devices
- Identify device
- Set IP address
- Set PROFINET name

8.6 SITOP (Power supplies)

8.6.1 SITOP support (Power supply)

A check mark (✓) means that the operation is supported.

Article number	Module name	Firmware version	Module type	Factory reset	Firmware update
6EP3 436-8MB00-2CY0	PSU8600	V1.1, V1.2	IM		✓
6EP3 436-8SB00-2AY0	PSU8600	V1.1, V1.2	IM		✓
6EP3 437-8MB00-2CY0	PSU8600	V1.0, V1.1, V1.2	IM		✓
6EP3 437-8SB00-2AY0	PSU8600	V1.1, V1.2	IM		✓
6EP4 134-3AB00-2AY0	UPS1600 10A PN	V1.14, V1.22, V2.0, V2.1, V2.2	SITOP	✓	✓
6EP4 136-3AB00-2AY0	UPS1600 20A PN	V1.14, V1.22, V2.0, V2.1, V2.2	SITOP	✓	✓
6EP4 137-3AB00-2AY0	UPS1600 40A PN	V2.0, V2.1	SITOP	✓	✓
6EP4 293-8HB00-0XY0	BUF8600	V1.1, V1.2	SITOP		✓
6EP4 295-8HB00-0XY0	BUF8600	V1.1, V1.2	SITOP		✓
6EP4 297-8HB00-0XY0	BUF8600	V1.0, V1.1, V1.2	SITOP		✓
6EP4 297-8HB10-0XY0	BUF8600	V1.0, V1.1, V1.2	SITOP		✓
6EP4 436-8XB00-0CY0	CNX8600	V1.0, V1.1, V1.2	SITOP		✓
6EP4 437-8XB00-0CY0	CNX8600	V1.0, V1.1, V1.2	SITOP		✓

8.7 RFID and MOBY (Communication modules)

8.7.1 RFID (Radio Frequency Identification)

A check mark (✓) means that the operation is supported.

Article number	Module name	Firmware version	Module type	Firmware update
6GT2 002-0EF00	RF160C CM	V1.0	RFID	✓
6GT2 002-0HD00	RF170C CM	V1.0	RFID	✓
6GT2 002-0HD01	RF170C CM	V3.0	RFID	✓
6GT2 002-0JD00	RF180C CM	V2.0	RFID	✓

8.7.2 MOBY (DeviceNet interface)

A check mark (✓) means that the operation is supported.

Article number	Module name	Firmware version	Module type	Firmware update
6GT2 002-0EB00	MOBY interface ASM 450	V3.0	IM	✓
6GT2 002-0ED00	MOBY CM ASM 456	V5.0	IM	✓
6GT2 002-0GA10	MOBY CM ASM 475		SM	✓

Index