SIEMENS

SIMATIC

S7/HMI
SIMATIC Automation Tool V3.0 user guide

Manual

V3.0, 06/2017

A5E33042676-AE

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

℗ 06/2017 Subject to change

Preface

Service and support

In addition to our documentation, Siemens offers technical expertise on the Internet and on the customer support web site (https://www.siemens.com/automation/).

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. All translated manuals refer back to the English manual as the authoritative and/or original source. Siemens identifies the English manual as the authoritative and/or original source in the case of discrepancies between the translated manuals.

Security information

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

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

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

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

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

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

Table of contents

Software license and product updates

1

1.1 Software license

Software license requirement

SIMATIC Automation Tool version 3.0 and later require a software license for full feature operation.

The TIA (Totally integrated Automation) ALM (Automation License Manager) software is installed/updated during the SIMATIC Automation Tool installation process.

At startup, the ALM checks for a V3 license. If no license is found, then SIMATIC Automation Tool starts in "unlicensed" mode.

The licensed and unlicensed versions can use all new V 3.0 features, and expanded SIMATIC device support.

Unlicensed operation

- Only one Device table row may be selected (checked) at one time. Multiple device and file processing is disabled.
- The API (Application Interface) for custom application programming is disabled.

Licensed operation

- Multiple Device table row selection is possible. Multiple communication connections and processing threads are used to process group tasks.
- The API is enabled for custom application programming.

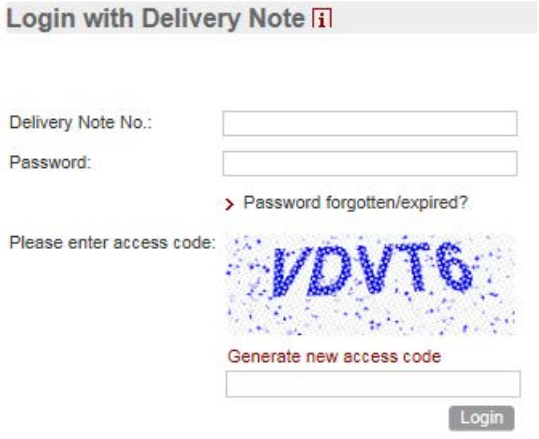
Getting a license

The license can be purchased through the Siemens Mall, and installed with the Automation License Manager 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 the Web license Key download icon. This action opens the SIEMENS Online Software Delivery page for viewing in the ALM window.



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

You have the option to install the TIA Software updater software 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.

This 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 a control program is created and verified with the Siemens TIA Portal software, the SIMATIC Automation Tool can be used in the field for configuring, operating, maintaining, and documenting automation networks.

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.

Automation Tool operations

- Scan the network and create a table that maps the accessible devices on the network. Unconfigured and configured CPUs, modules, HMIs, and other Siemens devices are included in the table data that is stored in a secure *.sat project file, or an open text .csv file.
- Flash device LEDs or HMI screens to physically locate a device
- Download addresses (IP, subnet, gateway) to a device
- Download PROFINET name (station name) to a device
- Put a CPU in RUN or STOP mode
- Set the time in a CPU to the current time in your PG/PC (Programmer/Personal Computer)
- Download a new program to a CPU or HMI device
- Upload, download, or delete Recipe data from a CPU
- Upload or delete Data log data from a CPU
- Backup/Restore data to/from a backup file for CPU and HMI device
- Upload service data from a CPU
- Read the diagnostic buffer of a CPU
- Perform a CPU Memory reset
- Reset devices to factory default values
- Download a firmware update to a device

SIMATIC device support

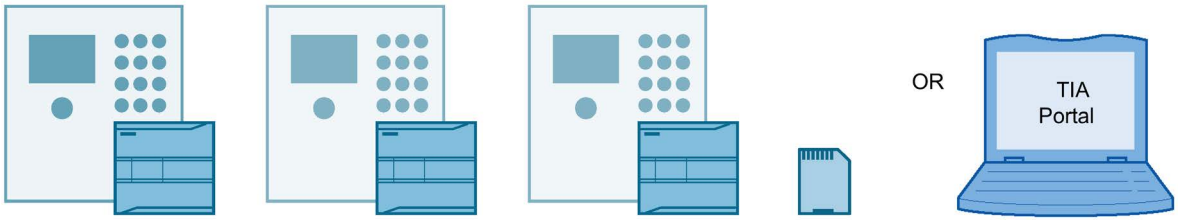
The SIMATIC Automation Tool supports standard and fail-safe CPUs, HMIs, and I/O devices.

The safety relevant operations Reset to factory defaults, Restore from backup file, and Program update are not supported on fail-safe CPUs.

For some devices, some SIMATIC Automation Tool operations are not supported.

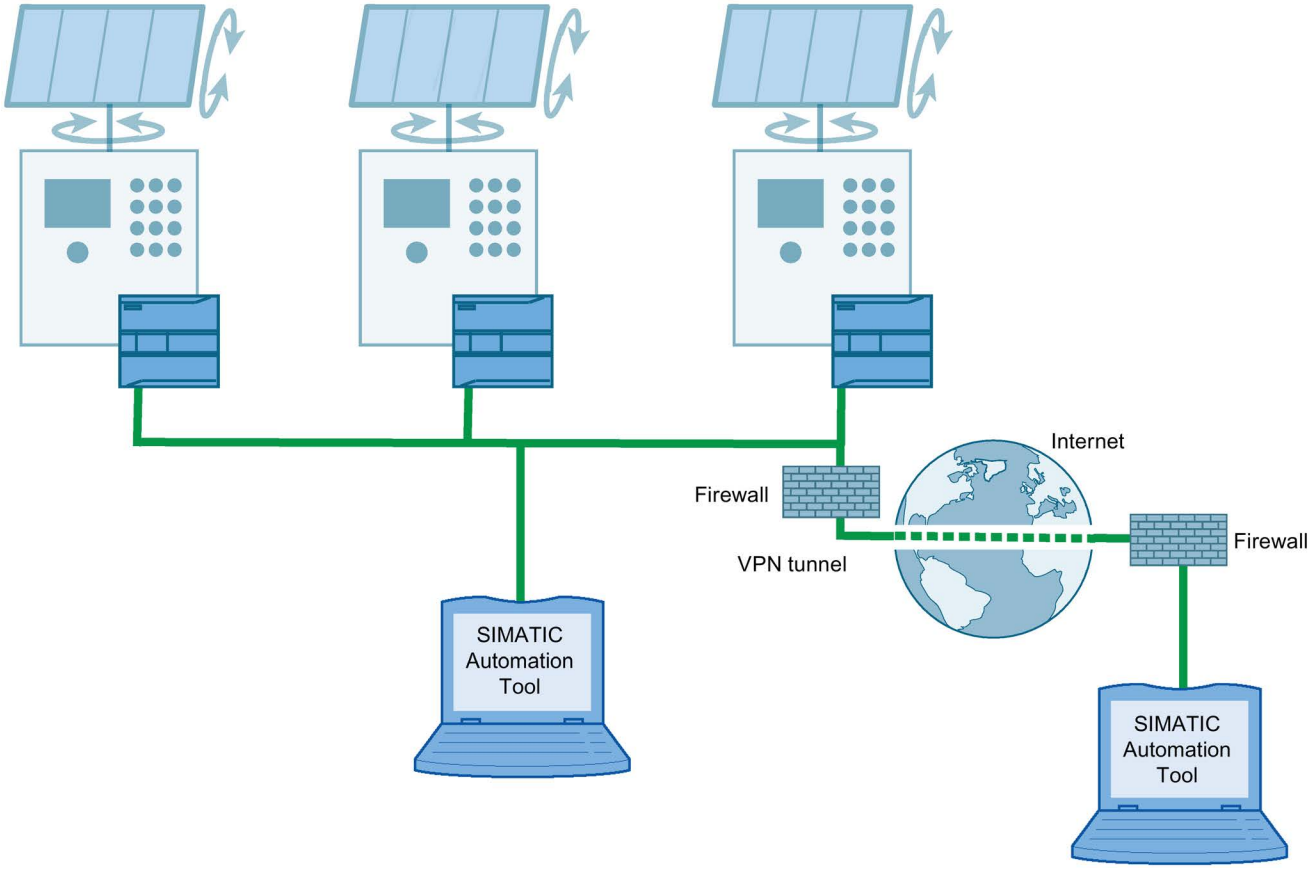
Click on a device type for support details.

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.
- The panel shop uses a memory card or the TIA Portal to load the CPU program.



SIMATIC Automation Tool configuration, operation, and maintenance

- Network configuration
  - Scan a network to find CPUs, HMIs, modules, and other Siemens devices
  - Flash LEDs and HMI screens to identify a device
  - Assign IP addresses and PROFINET names for each device, when you set up your network
- System configuration
  - Set the time in CPUs to the current time in your PG/PC
  - Load user programs for CPU and HMI devices
  - Upload, Download, or delete Recipe data from a CPU
  - Upload or delete Data Log data from a CPU
  - Update firmware in devices
- Operation
  - Put CPUs in RUN or STOP mode
- System diagnostics and maintenance
  - Backup/Restore data to/from a backup file for CPU and HMI devices
  - Read a CPU diagnostic buffer and upload service data
  - Reset CPU memory
  - Reset devices to factory default values
  - Document and save your network information in a standard text .csv file or a password protected .sat file.

2.2 Network access

Communicating with Siemens devices

PROFINET devices connected directly to a network are discovered 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 PROFINET devices are discovered whether they are configured with an IP, subnet, and gateway address, or not configured (addresses are 0.0.0.0).

A directly connected CPU must have a valid IP address and a valid project hardware configuration before communication modules, signal modules, and decentralized I/O that are connected behind the CPU become visible in the SIMATIC Automation Tool device table.

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 are connected. You must assign a valid IP address to the S7-1500 CPU and successfully compile and download your project's device configuration before the decentralized I/O network is visible in the SIMATIC Automation Tool device table.

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
Flash LED/HMI screen on devices	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	-	
Put CPUs in RUN or STOP	IP	Yes	• **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	
Download programs to CPU and HMI devices	IP	Yes	
Upload, download, or delete Recipe data from a CPU	IP	Yes	
Upload or delete Data log data from a CPU	IP	Yes	
Backup/restore CPU and HMI data	IP	Yes	
Upload service data from CPUs	IP	Yes	
Read CPU Diagnostic buffer	IP	Yes	
Reset CPU memory	IP	Yes	
Reset devices to factory default values	IP	Yes	
Download new firmware to devices	IP	Yes	

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 subnet assignments.

The type of network interface protocol that you select ("TCPIP" or "TCPIP.Auto") can affect whether Siemens devices are discovered during the SIMATIC Automation Tool Network scan.

See the example in the Communication setup topic (Page 29).

2.3 Network options

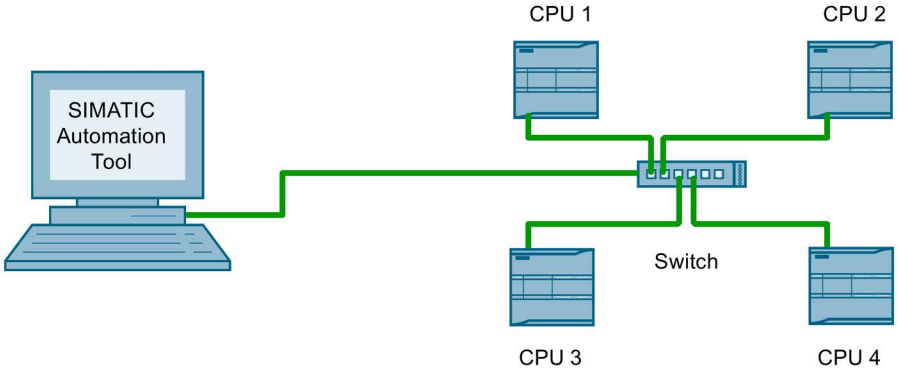
Local and remote networks

The following examples show local and remote networks that the SIMATIC Automation Tool can use. The diagrams are simplified to 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.

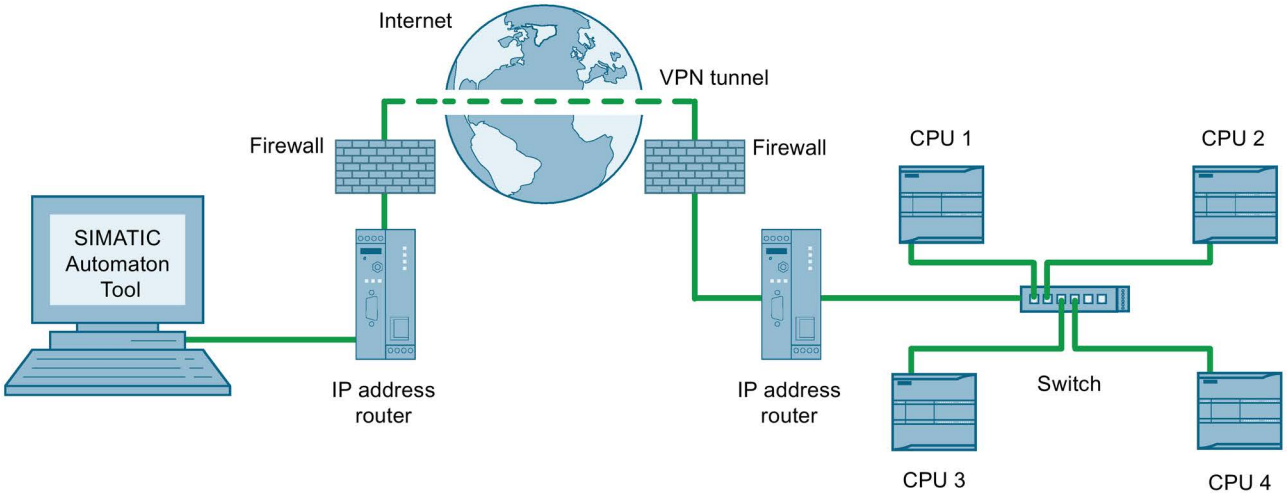
IP addressing and therefore valid SIMATIC device IP address configuration is necessary to get access to devices behind Ethernet IP routers.

You can fill the SIMATIC Automation Tool device table by automatically scanning a network (Page 41) or by importing a list (Page 74) that identifies devices that are behind a router.

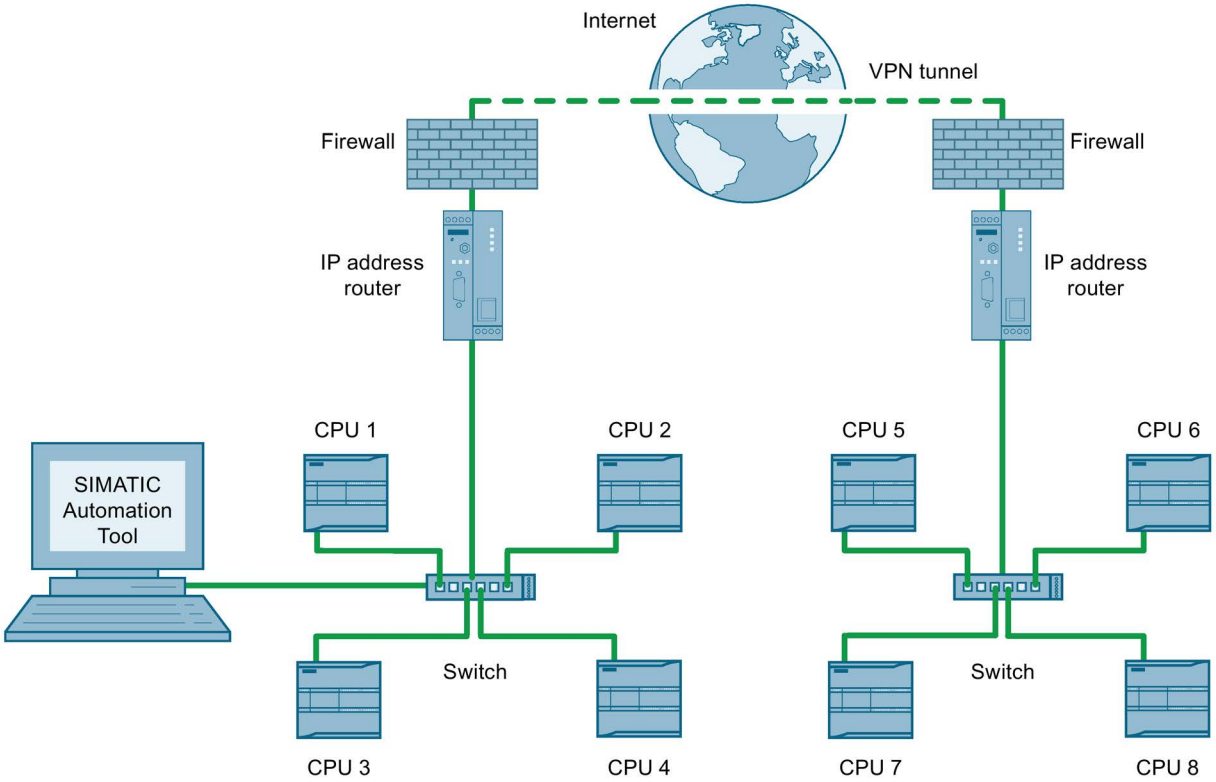
Example 1: S7-1200 local network



Example 2: S7-1200 remote network



Example 3: S7-1200 combined local and remote networks



2.4 .NET API (application interface) .dll file

The Microsoft .NET API used by the SIMATIC Automation Tool is documented in this user guide. You can create your own application software that uses the API 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 installation also provides HMI related files that your application uses to interact with HMI devices.

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

Software license required for V3.0 and later versions

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

The API is enabled when operating V3.0 and later versions and a license is detected, by the Automation License Manager.

API .dll file

The AutomationToolAPI.dll file is located in the folder where the SIMATIC Automation Tool is installed.

HMI related files

These files are required for working with HMI devices and must exist in same directory where the AutomationToolAPI.dll file is stored.

DeviceManagerClient.dll
hmitr.dm.client.proxy.dll
hmitr.dm.client.stub.exe
hmitr.ipc.dll

See also SIMATIC Automation Tool API for .NET framework (Page 89)

Prerequisites and communication setup 3

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

Microsoft Windows operating systems not supported by V3.0

- No support for Windows operating systems older than Windows 7 and SIMATIC Automation Tool installation is blocked.
- No support for Windows 32-bit operating systems and SIMATIC Automation Tool installation is blocked.

Microsoft Windows 64-bit operating systems supported by V3.0

SIMATIC Automation Tool V3.0 is tested and approved to work 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)

SIMATIC Automation Tool may install and work correctly with versions of Windows 64-bit operating systems that are not tested and approved.
There is no guarantee that untested versions work and technical support is not provided.

You can install the SIMATIC Automation Tool and use the unlicensed version to test the operations yourself, for untested Windows 64-bit operating systems.

Virtual machine software support

SIMATIC Automation Tool V3.0 is tested and approved to work with the following VM (Virtual Machine) software

- VMware Workstation 12.5
- VMware Player 12.5

Virus and security software support

SIMATIC Automation Tool V3.0 is tested and approved to work 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)

3.2 Installing the SIMATIC Automation Tool

Tool installation

1. Save all your work in progress and close all PG/PC applications, before installing the SIMATIC Automation Tool.
2. Execute the Start.exe file to begin the installation.

V3.0 installation rules

Only one version of the SIMATIC Automation Tool can be installed on a PG/PC. If you have a previously installed version (V1.0, V2.0, V2.1, V2.1.1) this version must be uninstalled before installing V3.0.

The setup program for SIMATIC Automation Tool V3.0 checks for a previous installation. This check results in one of the following behaviors:

- If no version of SIMATIC Automation Tool is found, the setup can proceed.
- If a previous version of the application (V1.0, V2.0, or V2.1.1) is found, you are informed that you must uninstall the older version. You cannot proceed with the installation until you close and restart the installer.
- If a V3.0 installation is found, the setup presents options to modfiy/upgrade, repair, or uninstall the previous installation.

3.3 Starting the SIMATIC Automation Tool

Start options for 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".
- Start the Windows command prompt (cmd.exe) and enter the executable file name and optional project parameter.
  AutomationTool.exe [projectname.sat].

3.4 CPU configuration requirements

Ethernet address configuration

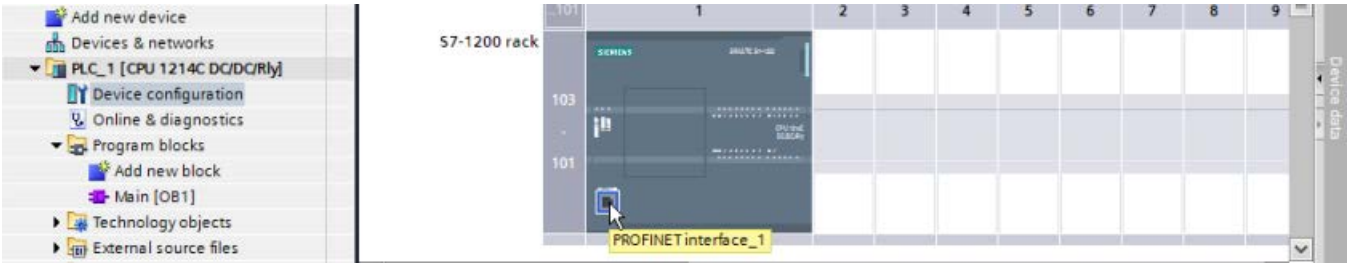
If you want the SIMATIC Automation Tool to set the IP address or PROFINET name of a CPU, then your TIA portal project must enable these actions in the CPU device configuration. Use the TIA portal to view and modify a program's IP protocol setting, as shown in the following S7-1200 example.

IP address and PROFINET name change

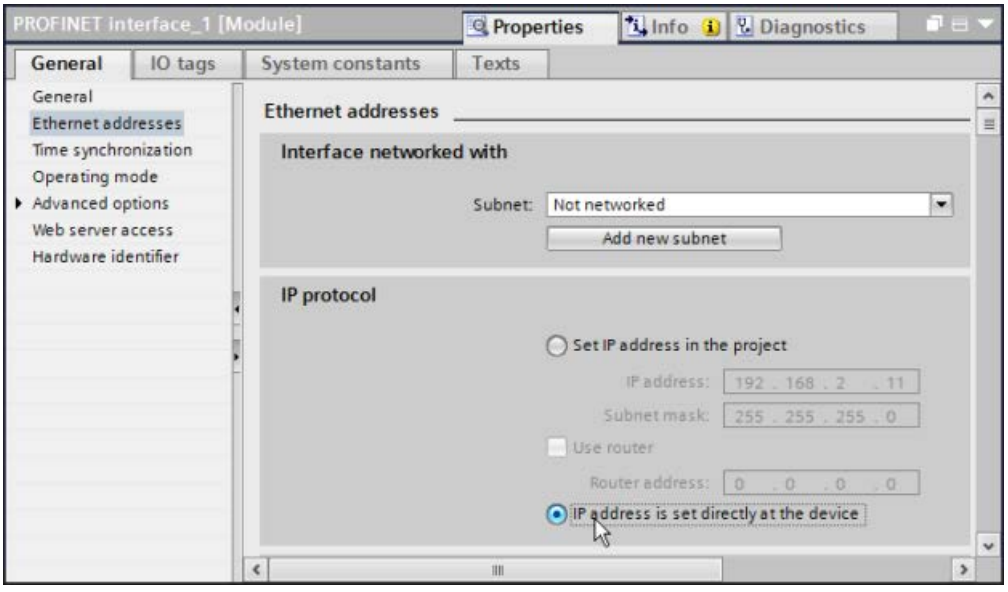
- **Possible** for PROFINET devices (CPUs, HMIs, decentralized I/O, and other devices) directly connected to the network that is connected to the SIMATIC Automation Tool, including connection through an Ethernet switch.
- **Not possible** for PROFINET devices with an indirect connection through a CP module, or a CPU's secondary Ethernet port, when the direct connection is to the primary 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.

S7-1200 example configuration

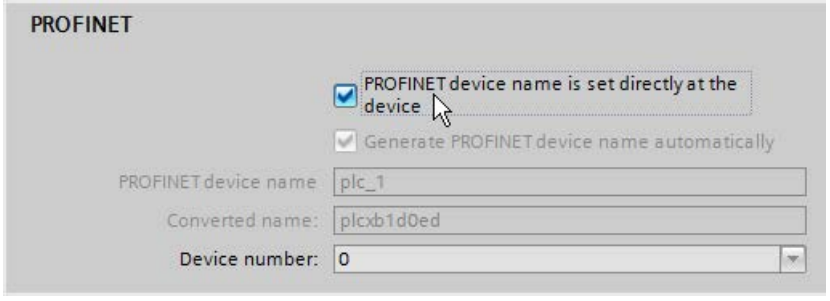
1. Click the PROFINET port on the device configuration CPU image, to view the port parameters.



2. On the **Properties** tab, click the **General** tab to view the **Ethernet addresses** options. Click the **SET IP address 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.



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.



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

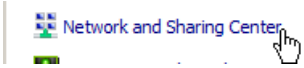
After you connect your PG/PC to a network, then you can use the Windows control panel to see the name of the network interface card.

In the following example, S7-1200 CPUs are connected to a USB port on a PC running Windows 7. The network interface card is a USB to Ethernet converter device. The options that you actually see on your PG/PC depend on your network hardware.

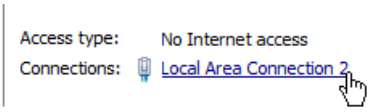
Use the Windows Control Panel to identify the name of the device.

1. Open the Windows Control Panel

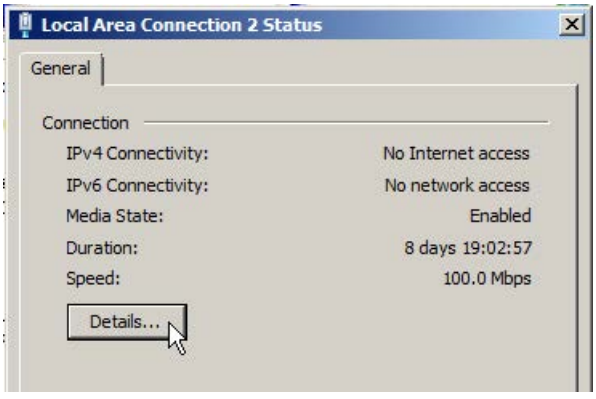
2. Click on the Network and Sharing center.



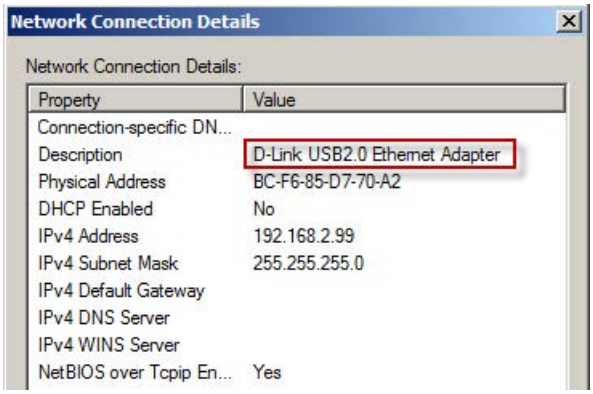
3. View your active networks and click on the network that is connected to the S7-1200 CPUs.



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



5. View the description of the network interface.



Assigning the network interface card in the SIMATIC Automation Tool

You must assign the network interface card to a new project, before communication can begin. Start the SIMATIC Automation Tool, click on the Network Interface Card drop-down list, and select the network card that is connected to your Siemens device network.

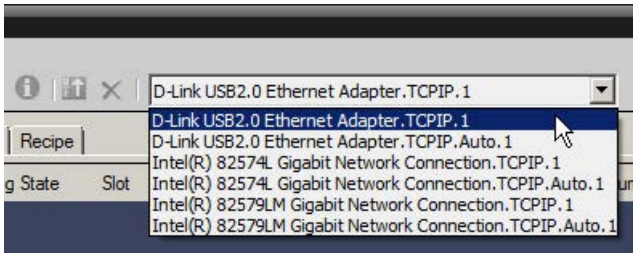
You may see different network cards from those shown in the following image, because the list shows the network interfaces that are available in your PG/PC.

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

MAC address operations

- Scan for network devices
- Flash LEDs to locate devices
- Set Ethernet IP addresses
- Set PROFINET names
- Reset to factory default values (for PROFINET I/O devices only)

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 values that work 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 difficult to diagnose communication errors for some parts of your network.

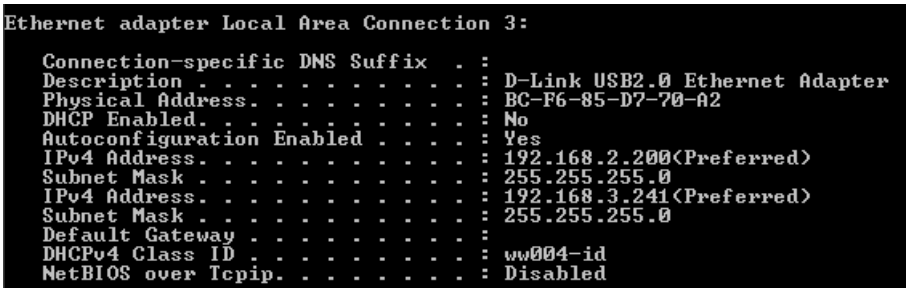
Example use of TCPIP and TCPIP.Auto protocols

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`.

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.



   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 download a new 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 reset and 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, then you can use the SIMATIC Automation Tool operations that use an IP address.

IP address operations

- Put CPUs in RUN or STOP
- Set CPU time to PG/PC time
- Program update for CPU and HMI devices
- Upload, download, or delete Recipe data from a CPU
- Upload or delete Data log data from a CPU
- Backup/restore CPU and HMI data
- Upload service data from CPUs
- Read CPU Diagnostic buffer
- Reset CPU memory
- Read Diagnostic buffer
- Reset devices to factory default values
- Download new firmware to devices

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 is displayed for that CPU. However, other devices are communicating and executing the operation as expected. If you have this problem, then reduce the maximum number of (threads/connections) that is assigned in the **Tools>Options>Communications** simultaneous operations. Close and restart the SIMATIC Automation Tool application, then 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 time delay value that is assigned in the **Tools>Options>Communications** Timeout for communications operations.

Tool operations

4

4.1 CPU passwords

If password security is configured in a CPU, then you must enter the password for the security level that allows the SIMATIC Automation Tool to perform the tool operation that you want to execute. Some tool operations require only the "Read access" password. A "Full access" password (read and write access) allows all tool operations. You provide a password in the device table column titled "Password".

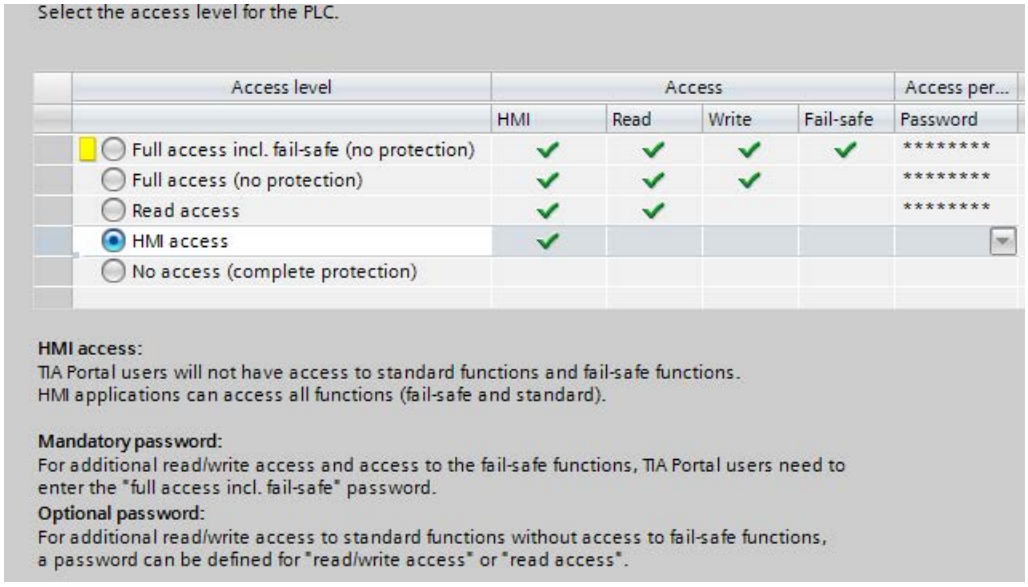
Note that the password entered in the "Password" column refers to the password protection that currently exists in the target CPU hardware and not passwords that are configured in a TIA portal project that you want to download to the CPU.

For example: A new CPU in a packing box from Siemens has no program and no hardware configuration. The new CPU has no password protection. When you perform a SIMATIC Automation Tool program update operation to load a TIA portal project that contains password protection, you must leave the "Password" column empty, because no password protection exists in the CPU. After a new project is loaded in the CPU, then you must use the passwords that are configured in that project.

A standard CPU has four security 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 level 2 "HMI access" or level 1 "No access" protection. You must configure a "Read access" or "Full access" password and then enter that password into the device table for the target CPU.

SIMATIC Automation Tool V3.0 cannot make a connection to a fail-safe CPU using the level 5 fail-safe password. For this reason, if you enter a level 5 fail-safe password for a fail-safe CPU, then a message box is displayed that warns you not to use a fail-safe password. If you ignore this warning (or did not know you used the fail-safe password) and enter the fail-safe password anyway, the password is rejected and deleted. You can connect to and work with a fail-safe CPU using a level 4 or level 3 password.

The 5 level fail-safe TIA portal password configuration is shown in the following image. Only the lower four levels are available in a standard CPU.

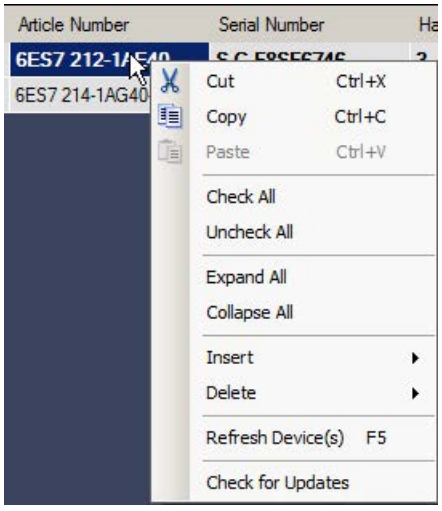


4.2 Working with the Device table and Event log

Working with the device table

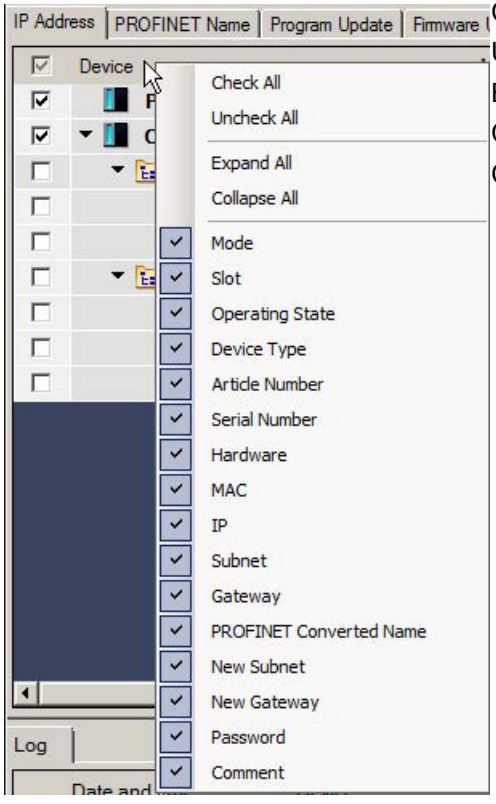
- Use Microsoft Excel compatible cell selection and copy/paste clipboard inside the device table or outside to/from another application.
- Click on a column header to sort or reverse sort the rows, by that column's data.
- Right-click a column header to show/hide any column.
- Click the check box column header or use the edit menu to Check All or Uncheck all rows.
- You can create row filters for the Device, Device type, and Article number columns.

Right-click menu for table cells



Cut current selection
Copy current selection
Paste current selection
Check all device rows (first column)
Uncheck all device rows (first column)
Expand all rows in the Device column
Collapse all rows in the Device column
Insert a new device row
Delete selection or checked device(s)
Refresh checked devices
Open SIEMENS industry support web page for current row's article number. Check the latest device information for firmware updates

Right-click menu for column headers



Check all device rows (first column)
Uncheck all device rows (first column)
Expand all rows in the Device column
Collapse all rows in the Device column
Check boxes to show/hide information columns

Filtering the displayed rows

You can fiter 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 131-6BF00-0BA0 and 6ES7 132-6BD20-0BA0. When you click the OK button, the device table only displays rows that have these article number values.

Filtering unsupported devices

You can use the Tools>Options>General (Page 80) dialog box to enable/disable the display of unsupported devices.

Scan rules for existing table entries

- If a MAC address already exists in the table, then the row for that MAC address has the IP address, Subnet, and Gateway fields updated. The data in all other fields remains.
- If a MAC address is not listed, then a new row is created. The MAC address, IP Address, Subnet, and Gateway are added. For a new row, all other fields are empty.

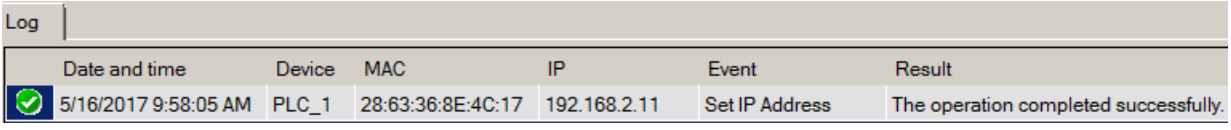
Working with the Event log

The event log is shown in the window area below the device table. When you select devices and perform operations, status information about successful and unsuccessful results is displayed in the Event log window.

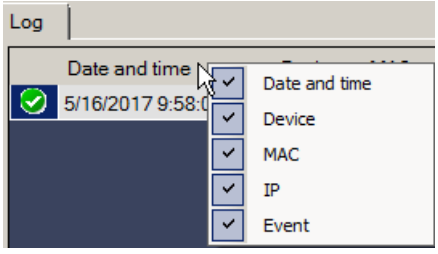
The SIMATIC Automation Tool can also automatically log operation status to a file. You can enable this feature on the Tools>Options>Log (Page 85) dialog box. To enable continuous logging, first assign the path where the file should be created and select the checkbox for "Automatically save log file". While this option is enabled, any actions that output a result to the Event log window are copied to the file "EventLogFile.csv". When you close and re-open the SIMATIC Automation Tool, logging to the file automatically resumes.

From the Options dialog, you can clear the content of the log file by clicking the "Clear Log" button. This clears the contents of the file, but does not delete it.

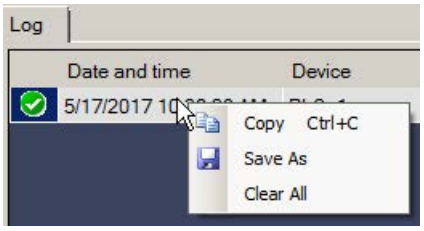
Event log row showing successful operation:



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



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



Note

Event log and user interface language change

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

4.3 Scan a network

Basic operation

The device table for a new SIMATIC Automation Tool project is empty. To begin work with the SIMATIC Automation Tool, there must be one or more device rows visible in the device table. You can automatically fill the device table with a network scan or manually assign devices using the Insert > Device command.

Row icons help you identify the device table rows:

PROFINET device not supported. The row's address text is grayed and no entry or operations are possible.

 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 ASi master devices

 Folder containing Data log and/or Recipe data

 Data log data

 Recipe data

Device text is displayed in black when not selected and **bold black** when the device is selected.

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

You can enter text in cells with a light gray background

You cannot enter text in cells with a dark gray background. A disabled dark gray cell in a New value column indicates the operation is not supported for the selected device type/firmware version.

After valid IP addresses are entered in the device table, you can use the "Download" command to transfer the address assignments into selected directly connected devices.

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

Scan your network

1. Select the Network>Scan menu or click the Scan button on the toolbar.
2. The SIMATIC Automation Tool will fill a new device table or update an existing table with information from your network devices.

The initial scan shows devices that have a direct connection to the SIMATIC Automation Tool and are shown at the top level (left-most) in the device hierarchy.

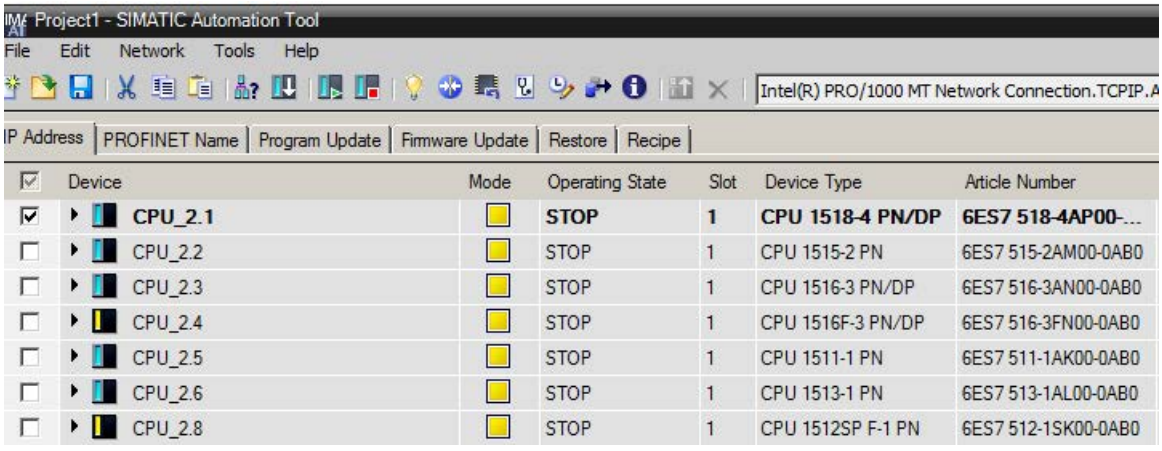
Directly connected devices (including connection through an Ethernet switch)

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), if the device firmware supports the operations.

PROFINET I/O

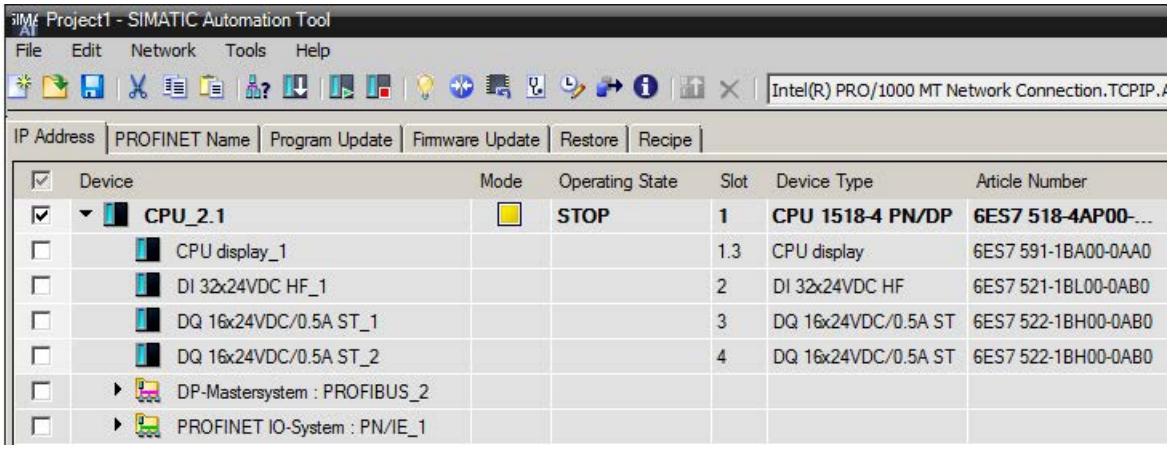
PROFINET I/O devices can be listed twice in the device table. The device is shown once on a top level row, where direct connection with the tool allows all supported SIMATIC Automation Tool operations. The device is also shown in a lower level row behind a CPU (with valid IP address and hardware configuration), 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.

Example initial scan result:



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 menu or Edit menu to expand/collapse all levels.



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 (for example, head module and I/O modules) are visible in the device tree.

See also

Log options (Page 85)

General options (Page 80)

4.4 Download new IP, subnet, and gateway addresses

Change IP addresses

1. Click the "IP Address" tab.
2. Click the left-side check box on devices to include in the operation. You can use the top check box, right-click shortcut menu, or the Edit menu for "Select All" and "Unselect All" commands.
3. Enter address changes in the "New IP", "New subnet", and "New Gateway" columns.
4. Select Download from the Network menu or click the Download button on the toolbar
5. The Download operation sets the IP, subnet, and gateway addresses in the selected devices.
6. The Event log below the device table shows the results of this operation.

IP	Subnet	Gateway	New IP	New Subnet	New Gateway
192.168.2.11	**255.255.255.0**	**0.0.0.0**	192.168.2.10	255.255.255.0	192.168.2.140
192.168.2.12	**255.255.255.0**	**0.0.0.0**	192.168.2.11	255.255.255.0	192.168.2.140

Duplicate IP addresses

When two or more devices have the same IP address, the addresses will be shown in red text, as shown in the following image.

IP Address | PROFINET Name | Program Update | Firmware Update | Restore | Recipe

	Device	Article Number	Serial Number	Hardware	MAC	IP
☐	S7-1200	6ES7 212-1AE40-0XB0	S C-F8SF6746	0	28:63:36:8E:4C:17	192.168.2.10
☑	**S7-1200**	**6ES7 212-1AE40-...**	**S C-F8SF6746**	**0**	**28:63:36:A7:44:5C**	**192.168.2.10**

4.5 Download new PROFINET device names

Changing PROFINET device names

PROFINET name rules

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

A maximum of 63 characters is allowed. 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.

Duplicate names

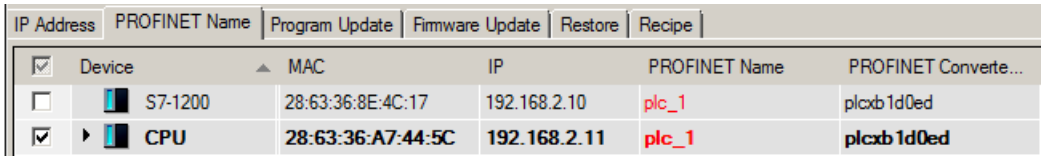
When two or more devices have duplicate PROFINET names, these will be indicated with red text. SIMATIC Automation Tool supports full functionality for these devices and displays all other information. Both the "PROFINET Name" field and the "PROFINET Converted Name" field show this error condition.

Duplicate PROFINET names are displayed in red text for both supported and unsupported devices. However, you can only update the PROFINET name for supported devices.

Change PROFINET name

1. Click the "PROFINET name" tab.
2. Click the left-side check box on devices to include in the operation. You can use the top check box, right-click shortcut menu, or the Edit menu for "Select All" and "Unselect All" commands.
3. Enter a new PROFINET name in the "New PROFINET Name" column.
4. Select Download from the Network menu or click the Download button on the toolbar.
5. The Download operation sets new PROFINET names in the selected devices.
6. The Event log below the device table shows the results of this operation.

Duplicate PROFINET names are colored red:

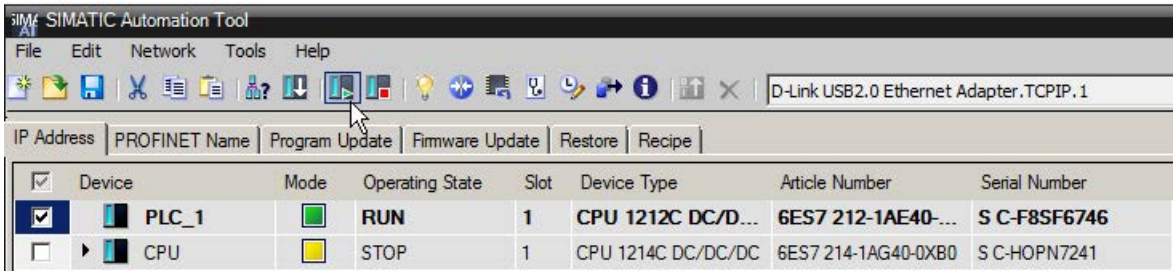


4.6 Set CPUs to RUN or STOP mode

Change CPUs to RUN or STOP mode

1. Click the left-side check box on devices to include in the operation. You can use the top check box, right-click shortcut menu, or the Edit menu for "Select All" and "Unselect All" commands.
2. Select RUN from the Network 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 Network menu or click the STOP toolbar button.
3. Selected CPUs are set to RUN or STOP mode.
4. The Mode and Operating state columns in the device table indicate the current CPU state. Yellow means STOP mode. Green means RUN mode, and RED means CPU fault.
5. The Event log below the device table shows the results of this operation.

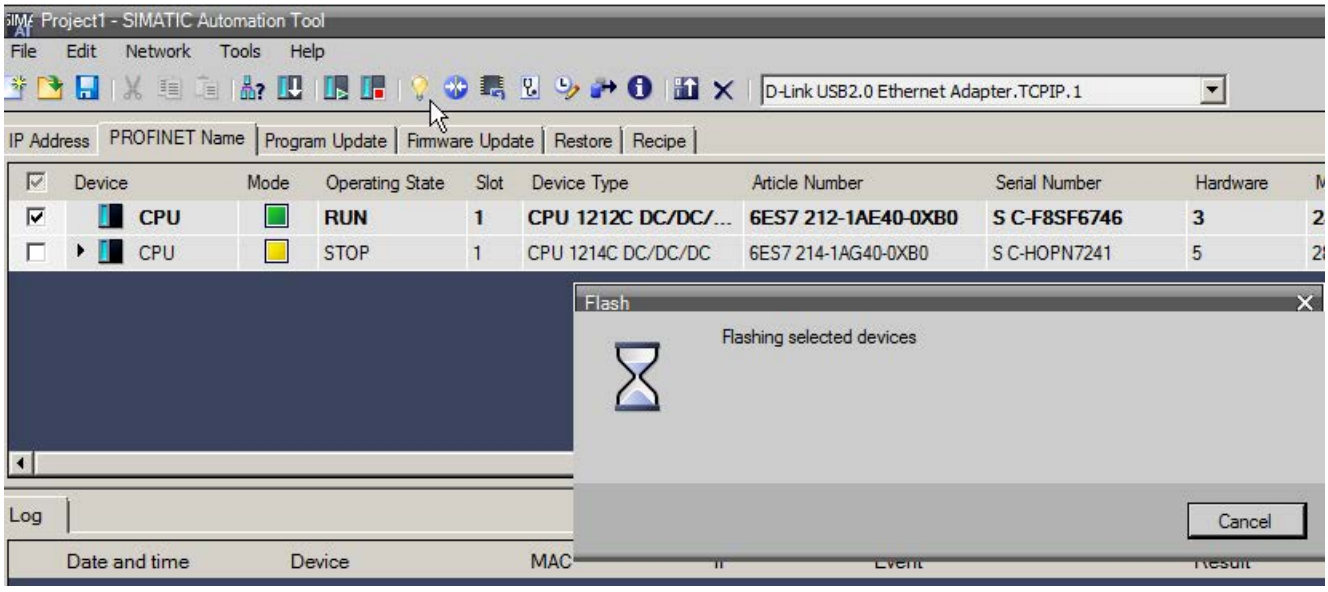


4.7 Flash LED or HMI panel to identify device

Locate a device by flashing an LED or HMI display

The Flash button will flash a light on selected devices. The Flash operation helps you physically locate which device has a specific MAC address. You can use the flash operation in RUN mode and STOP mode.

1. Click the left-side check box on devices to include in the operation. You can use the top check box, right-click shortcut menu, or the Edit menu for "Select All" and "Unselect All" commands.
2. Select flash from the Network menu or click the Flash toolbar button.
3. Selected devices flash a light and show their location.
4. Flashing continues until you click the cancel button.



4.8 Download new programs to devices

Preparing a program for use with the SIMATIC Automation Tool

A TIA portal program must be transferred by the TIA portal software to a SIMATIC memory card, USB flash drive, or another partition of your PG/PC hard drive before the program is usable with the SIMATIC Automation Tool. Refer to the TIA portal documentation about how to transfer a program to a storage device.

You can also update software in an HMI device with the SIMATIC Automation Tool Program Update operation. An HMI project data folder is created on the PG/PC running the SIMATIC Automation Tool and the path to that folder is selected in the HMI device row's Program Update column. HMI project data can include firmware, operating system, add-ons, and runtime software. You do not have the option to select a partial update. The SIMATIC Automation Tool updates all data components as necessary, for a consistent download.

Note

For safety reasons, the Program Update operation is not allowed on Fail-Safe devices.

After the TIA portal transfers program data to a storage device, you can use the Windows 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. Start the SIMATIC Automation Tool and view the Tools>Options>Program Update path assignment for the program update folder.
   The default path is C:\Users\MyAccount\SIMATIC Automation Tool\Programs.
   Your path may have a different drive letter and "MyAccount" represents the login name of the current user.
   You can browse to another folder and assign a different path.
2. Create subfolders under the folder assigned in the Options dialog. 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.
3. Use the Windows Explorer to copy the "SIMATIC.S7S" folder (including all subfolders and files) to the folder assigned in the SIMATIC Automation Tool. Optional recipe files are downloaded using the Recipe tab operation. A TIA portal program (a "SIMATIC.S7S" folder) can be put in a zip file archive and sent to a remote location.

Note

TIA portal program data is protected

Details like the project name or target CPU, of a TIA portal program, cannot be discovered 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 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.

Download new programs to a CPU or project data to a HMI device

WARNING

Verify that the CPU is not actively running a process before downloading a new program

Installing a new program 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.

After program data are copied to the program update folder, you can use the SIMATIC Automation Tool to load new programs in one or more devices.

1. Click the "Program Update" tab.
2. Click the left-side check box on devices to include in the operation. You can use the top check box, right-click shortcut menu, or the Edit menu for "Select All" and "Unselect All" commands.
3. For each selected device, use the "Program" column drop-down list to select a folder name. The drop-down list will show the folders that you created in the program update path.
   You can also use the browse button ... and navigate to the folder where a program is stored on your PG/PC. The program you browsed to is added to the drop-down list. If the selected file has the same name as one of the files already listed, a number is added to the new file to make the names unique. To help you identify files, a tooltip displays the entire path and filename.
4. Select the Download Command from the Network menu, or click the Download toolbar button to start the operation.
5. The Event log below the device table shows the results of this operation.

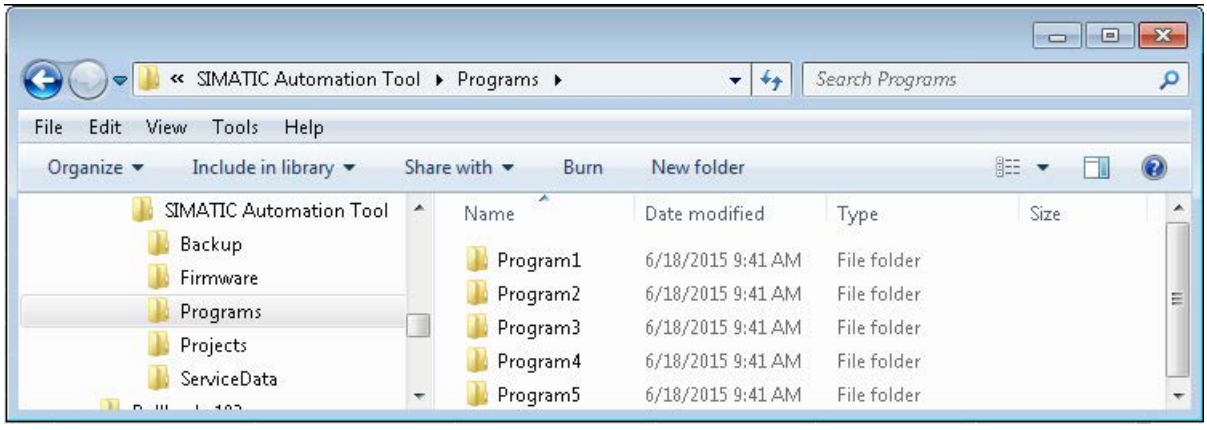
Example CPU program update

The default path in the options dialog for program update is C:\Users\MyAccount\SIMATIC Automation Tool\Programs.

If you want five different programs available for download, then you must create and name five folders under the path that is assigned in the Options dialog. Copy the entire "SIMATIC.S7S" folders to the five corresponding folders.

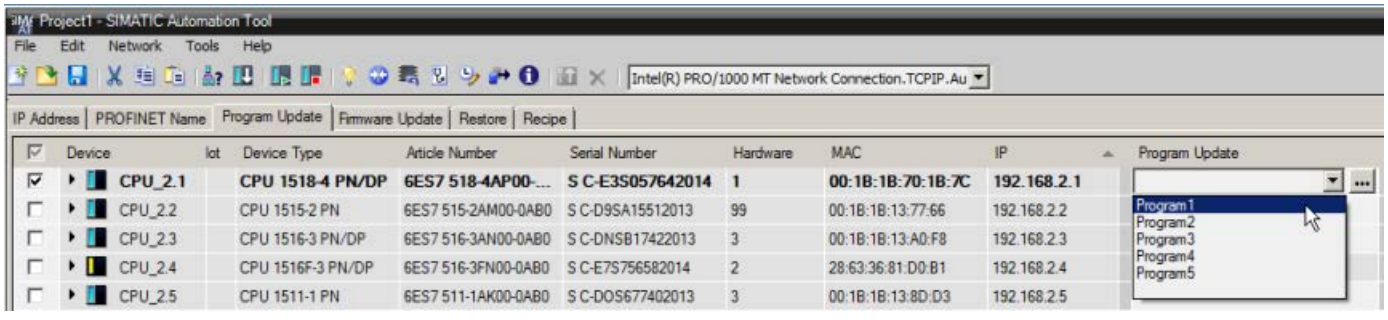
In this example, the folder names "Program1", "Program2", "Program3", "Program4", and "Program5" identify the available programs. You can use any folder name you want. The folder name could refer to a program function, or CPU location.

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



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

Select the Download command on the Network menu, or click the toolbar Download button, to start the program update operation.



CPU Program update rules

The program update operation is supported for standard CPUs and is not supported for Fail-Safe CPUs.

Program update rules:

1. 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.
2. For the S7-1200, S7-1500, and ET 200SP (S7-1500) CPUs, the program update operation is possible if the project's assigned CPU firmware version is supported as shown in the following tables.
3. Due to an S7-1200 V3.0 to V4.0 hardware and firmware change, combinations marked with * are not possible.

Program update support tables

- Program update is possible for combinations where ✓ is displayed.
- Program update is not possible where an empty cell or * is displayed.

S7-1500 ET 200SP CPU		CPU firmware version configured in project for download to CPU						
		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	✓	✓	✓	✓	✓	✓	✓

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 download to CPU						
		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	*	*	*	*	✓	✓	✓

4.9 Upload, download, and delete Recipes in CPUs

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

Recipe upload, download, and delete operations work with CPUs in RUN and STOP mode.

- **Recipe Upload** copies recipe files from a CPU to your PG/PC.
- **Recipe Delete** erases recipe files in a CPU.
- **Recipe Download** copies recipe files from your PG/PC to a CPU.

You can download multiple recipes in a single operation, if the recipes are all on different CPUs. You cannot download more than one recipe into one CPU, for each Download operation.

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

When a recipe file or recipe folder is selected, the device table row is displayed in **bold text**. When one or more Recipes are selected, the Upload and Delete functions are enabled.

For Recipe downloads, Click on the Recipe tab and select a CPU's Recipe folder row. When a recipe file that you prepared and stored in the Tools>Options>Recipes path is entered in the Recipe name column of a Recipe folder row, the Download button is enabled.

SIMATIC Automation Tool creates a unique folder name for each CPU, to store uploaded recipe 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 recipe file twice, a number is appended to the filename, to make all filenames unique.

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

Toolbar actions

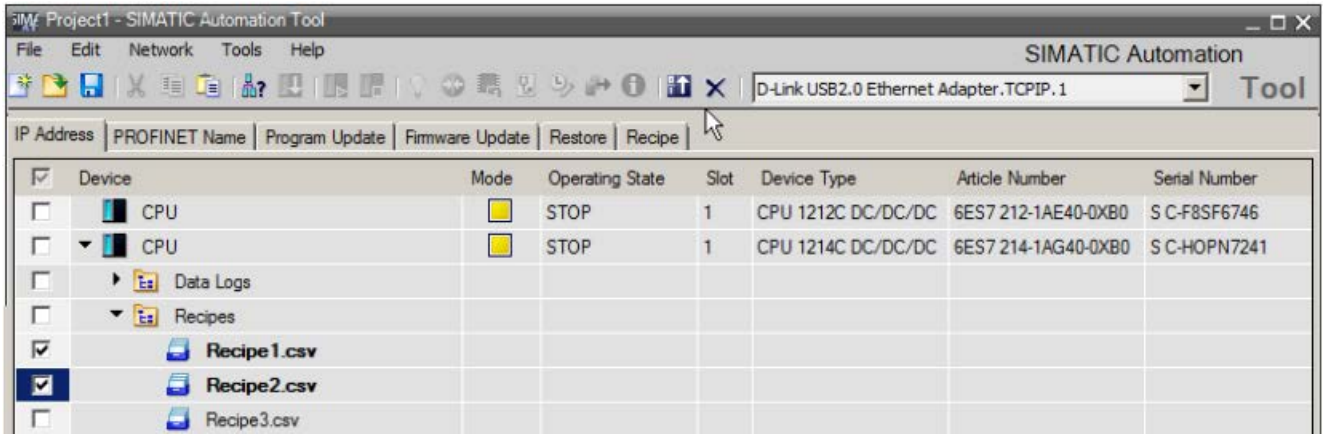
Upload selected file(s) Copies selected recipe files from CPU to PG/PC. Recipes are copied to the PG/PC directory path assigned in the Tools>Options>Recipes dialog box.

Delete selected file(s) Deletes selected recipe files that are stored in a CPU.

Download checked items Copies selected recipe files from your PG/PC to a CPU.

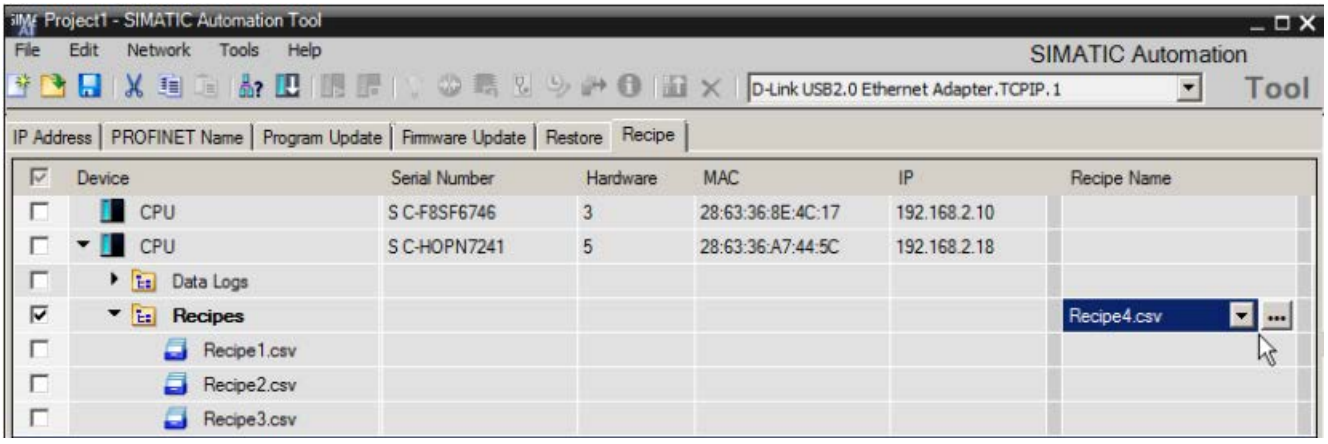
Upload or Delete recipe files

1. Expand a CPU row and make a recipe folder visible
2. Expand a recipe folder and then click the left-side check box on recipe files to include in the operation.
3. Select the menu command Network>Remote file(s)>Upload/Delete
   or click Upload selected file(s) from a CPU
   or click Delete selected file(s) in a CPU
4. The Event log below the device table shows the results of this operation



Download recipe files

1. Copy Recipe data .csv files that you want to download to the directory path assigned in the Tools>Options>Recipes dialog box.
2. Click the Recipe tab.
3. Expand a CPU row and make a recipe folder visible.
4. Click the left-side check box on a target CPU's recipe folder row.
5. For each recipe folder row selected, use the "Recipe Name" column drop-down list and select a recipe file name. The drop-down list will show the names of .csv files that exist in the directory path assigned in the Tools>Options>Recipes dialog box.
   You can also use the browse button ... and navigate to the folder where recipe files are stored on your PG/PC. The file you browsed to is added to the drop-down list. If the selected file has the same name as one of the files already listed, a number is added to the new file to make the names unique. To help you identify files, a tooltip displays the entire path and filename.
6. Select the menu command Network>Download or click the download button.
7. The Event log below the device table shows the results of this operation



WARNING

Security note

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.

4.10 Upload and delete Data logs in CPUs

The Data log upload operation works for CPUs in RUN and 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. When a Data log file is selected, the device table row is displayed in **bold text**. When one or more Data log files are selected, the Upload and Delete functions are enabled.

SIMATIC Automation Tool creates a unique folder name for each CPU, to store uploaded Datalog 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.

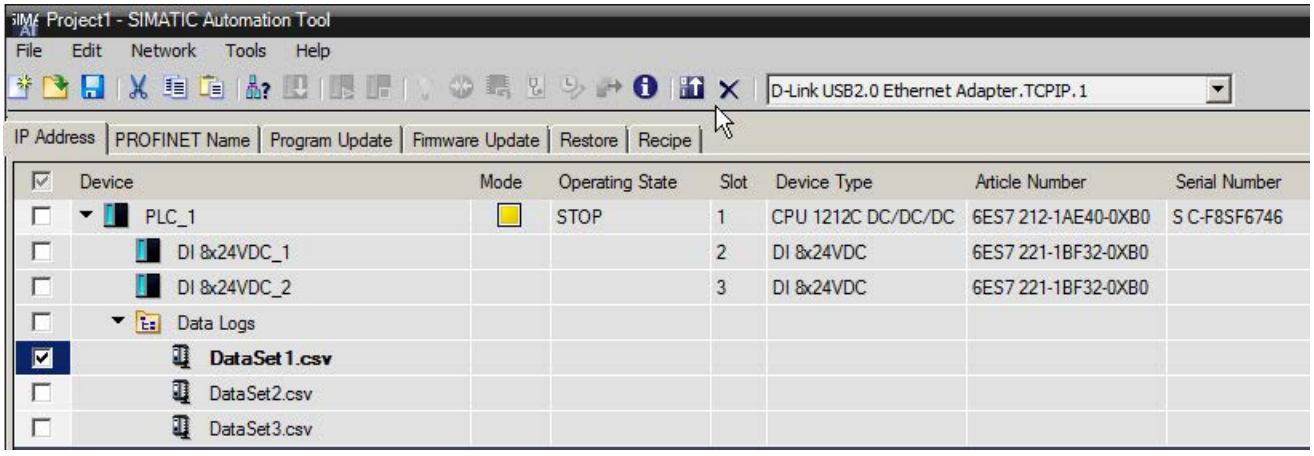
Toolbar actions

Upload selected file(s) Uploads a copy of selected Data log file(s) from CPU to PG/PC. The files are copied to the directory assigned in the Tools>Options>Data logs dialog box.

Delete selected file(s) Deletes selected Data log records that are stored in a CPU.

Upload or Delete Data log files

1. Expand a CPU row and make any Data log folders visible
2. Expand a Data log folder and select Data log files
3. Select the menu command Network>Remote file(s)>Upload/Delete
   or click Upload selected file(s) from a CPU
   or click Delete selected file(s) in a CPU
4. The Event log below the device table shows the results of this operation.



WARNING

Security note

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.

4.11 Install new firmware in devices

Firmware updates

TIA Portal

A SIMATIC memory card can 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 firmware, operating system, and runtime software updates

You must use the SIMATIC Automation Tool **Program Update** operation to update firmware, operating system, and runtime software for HMI devices. 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

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.

By default, the firmware update operation does not replace firmware with the same firmware version and allows only a single processing thread that must complete execution, before the next device operation is executed. To change this behavior to multi-thread processing, see the Tools>Options>Communications (Page 81) topic. Be aware of the risk of communication disruption that is described in that topic.

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 the S7-1200 CM, you must complete a TIA portal CM configuration and download the configuration, before you can 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 may 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.

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 go directly to a device's customer support web page by right-clicking with the mouse cursor on a device row and then selecting the menu item "Check for updates". 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 will 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 **232-4Hhd32-0XB0**_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 underneath the C:\Users\MyAccount\SIMATIC Automation Tool\Firmware folder. 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.

1. Run the SIMATIC Automation Tool and view the Tools>Options>Firmware Update setting and note the folder assignment for firmware update files. The default path is C:\Users\MyAccount\SIMATIC Automation Tool\Firmware. You can modify the default setting.
   Your path may have a different drive letter and "MyAccount" represents the login name of the current user.
2. Copy all the .upd files you need to the firmware folder assigned in the Tools>Options>Firmware Update dialog box.

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.

Download firmware updates to CPUs and modules

After .upd files are copied to the firmware update folder, you can use the SIMATIC Automation Tool to update the firmware in CPUs and modules.

1. Click the "Firmware Update" tab.
2. Click the left-side check box on devices to include in the operation. You can use the top check box, right-click shortcut menu, or the Edit menu for "Select All" and "Unselect All" commands.
3. For each device row selected, use the "Firmware Version" column drop-down list and select a firmware version for either a CPU or module. The drop-down list will show the names of the .upd files that you copied to the firmware update path. If new firmware versions (.upd files) are available in the firmware update folder, then the latest version is automatically entered in the "Firmware version" column cell for a device.

   You can also use the browse ... button and navigate to a folder where a firmware update files is stored on your PG/PC. The file you browsed to is added to the drop-down list. If the selected file has the same name as one of the files already listed, a number is added to the new file to make the names unique. To help you identify files, a tooltip displays the entire path and filename.

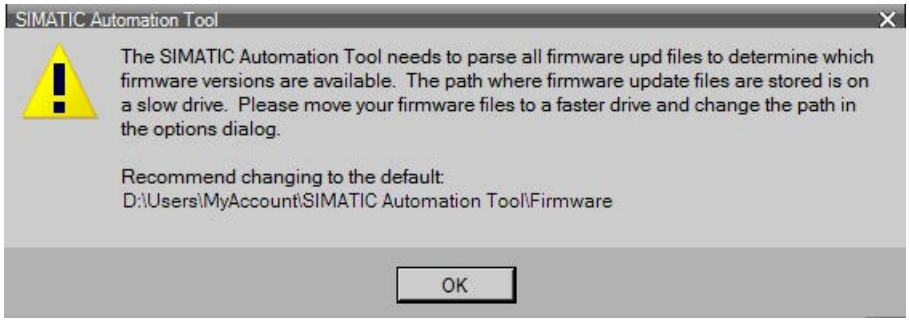
4. Select the Download Command from the Network menu, or click the Download toolbar button to start the operation.
5. Allow time for the firmware update to complete. Wait until the CPU lights stop flashing before attempting another operation with this CPU

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

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

If you see the following error message box, then more than 5 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 is related to data access time and the number of .upd files in the folder.

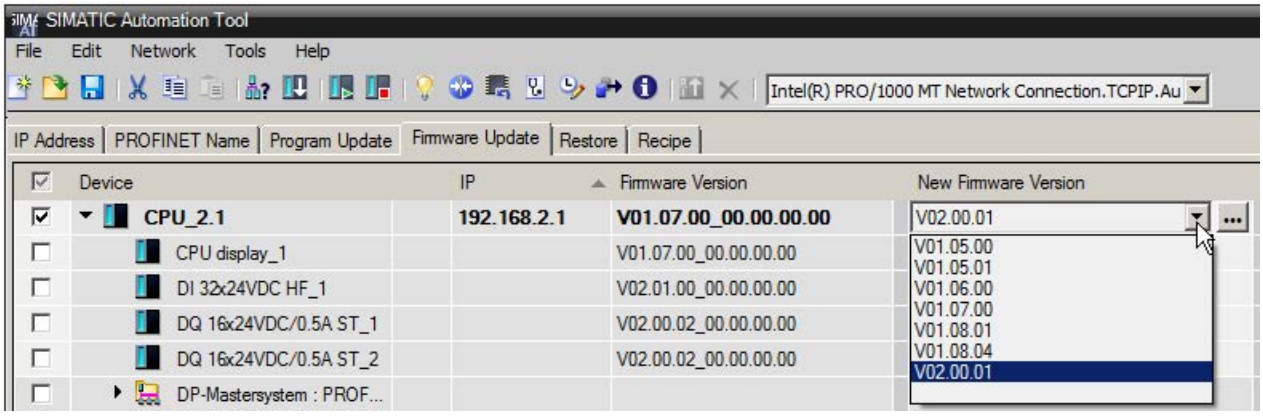
- This timeout error can occur when communication with a remote storage device that is too slow.
- To prevent this problem, assign a faster firmware data storage path with the Tools>Options>Firmware Update dialog box. Copy the .upd files you need to a faster local storage device and try the operation again.



Example firmware update

The default path for firmware update is C:\Users\MyAccount\SIMATIC Automation Tool\Firmware.

- If you want many different firmware versions available for downloading, then you must copy each .upd file to the firmware subfolder.
- On the Firmware update tab, the SIMATIC Automation Tool lists the available versions (.upd files) in the "New Firmware Version" column drop-down list. One CPU was checked in the select column. So, you must use the FW Version drop-down list and assign a file for this CPU. If more than one device is selected, then you must repeat the process and assign the correct update file for each selected device.
- If multiple I/O modules of the same model exist, then one module firmware update will update all similar modules. I/O module firmware can be updated separately without updating firmware in the rack's CPU.
- Select the Download command on the Network menu, or click the toolbar Download button, to start the operation.



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.

Modules that have article numbers containing xxx**30** or xxx**31** cannot be updated with the Automation Tool and you must use a SIMATIC memory card. This is only for S7-1200 modules installed on the left or right side of the CPU. For example, the middle part of the article number 6ES7232-4HD30-0XB0 contains 4HD**30** and you cannot update the firmware of this module with the SIMATIC Automation Tool.

4.12 Backup and Restore CPU or HMI data

Backup device

New data backup files are created and copied to the assigned folder.

The default file path is C:\Users\MyAccount\SIMATIC Automation Tool\Backup.
Your path may have a different drive letter and "MyAccount" represents the login name of the current user.

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

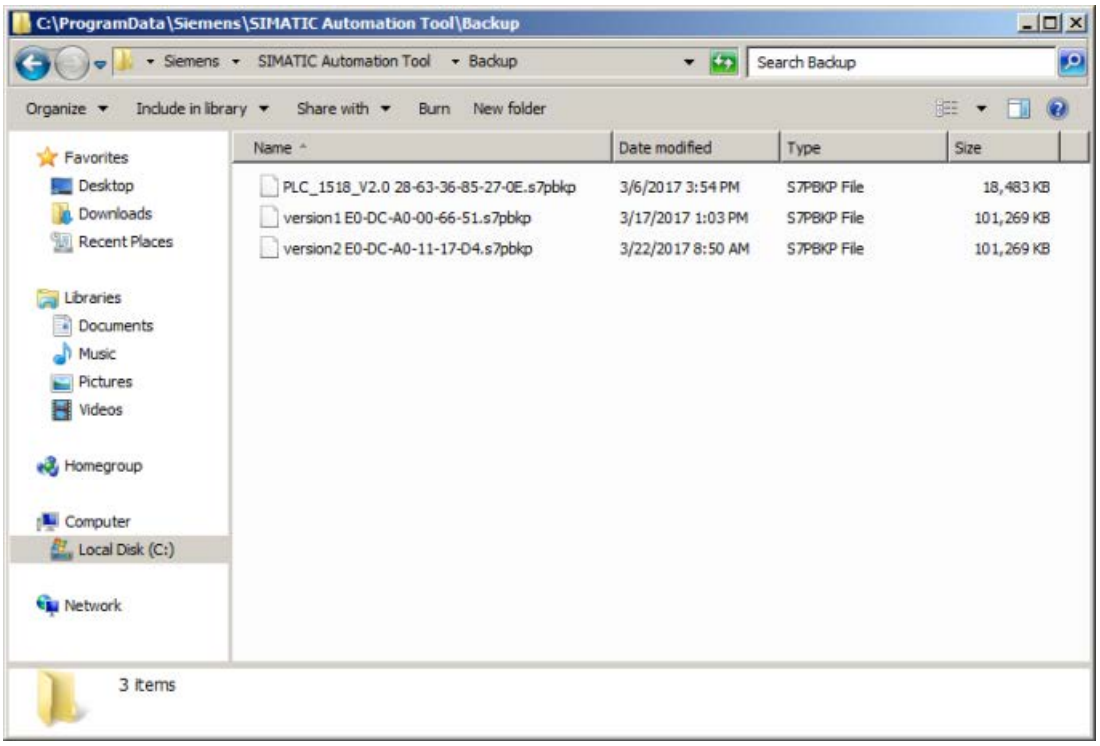
Note

For safety reasons, the Restore from backup file operation is not allowed on fail-safe CPUs.

You can start the backup operation from any tab selection.

1. Click the left-side check box on devices to include in the operation. You can use the top check box, right-click shortcut menu, or the Edit menu for "Select All" and "Unselect All" commands.
2. Select "Backup Device" from the Network menu or click the "Backup device to file" toolbar button.
3. New CPU backup files are created and copied to the backup file path assigned in the Tools>Options>Backup/Restore dialog box.
4. The Event log below the device table shows the results of this operation.

SIMATIC Automation Tool creates a backup file name for S7 and HMI devices, by combining the project name, MAC address, and .s7pbkp.



Restore name

Restore CPU data from a backup file. You can create backup files using the TIA Portal or the SIMATIC Automation Tool. S7 and HMI backup files have the extension name "s7pbkp" that must exist in the backup file path assigned in the Tools>Options>Backup/Restore dialog box. The default file path is C:\Users\MyAccount\SIMATIC Automation Tool\Backup.

You can create Backup files using the TIA Portal or the SIMATIC Automation Tool. These files have the extension name "s7pbkp". Use the cells in the "Restore Name" column to enter the backup file name. A backup file contains the data you want to restore to a selected device.

By default, the restore operation allows only a single processing thread that must complete execution, before the next device operation is executed. To change this behavior to multi-thread processing, see the Tools>Options>Communications (Page 81) topic. Be aware of the risk of communication disruption that is described in that topic.

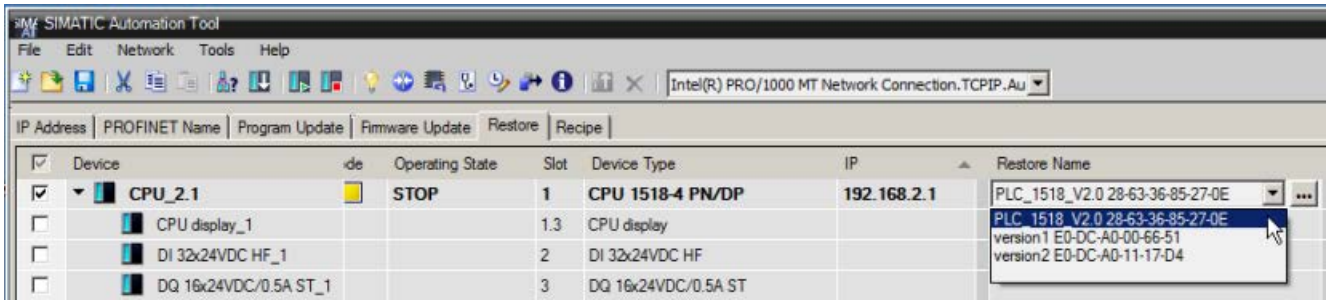
Use the following steps to restore selected devices from a backup file.

1. Click the "Restore Name" tab.
2. Click the left-side check box on devices to include in the operation. You can use the top check box, right-click shortcut menu, or the Edit menu for "Select All" and "Unselect All" commands.
3. For each device row selected, use the "Restore Name" column drop-down list and select a backup file name. The drop-down list will show the names of the .s7pbkp files that exist in the file path assigned in the Tools>Options>Backup/Restore dialog box.

   You can also use the browse ... button and navigate to the folder where a back-up files is stored on your PG/PC. The file you browsed to is added to the drop-down list. If the selected file has the same name as one of the files already listed, a number is added to the new file to make the names unique. To help you identify files, a tooltip displays the entire path and filename.

4. Select the Download Command from the Network menu, or click the Download toolbar button to start the operation.
5. The Event log below the device table shows the results of this operation.

In the following image, one CPU is selected, so only one backup file selection is required in the "Restore Name" column. If more than one device is selected, then you must repeat the backup file assignment for each selected device.



4.13 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 IP address that exists before the operation is retained, so your network IP assignments are preserved.

Note

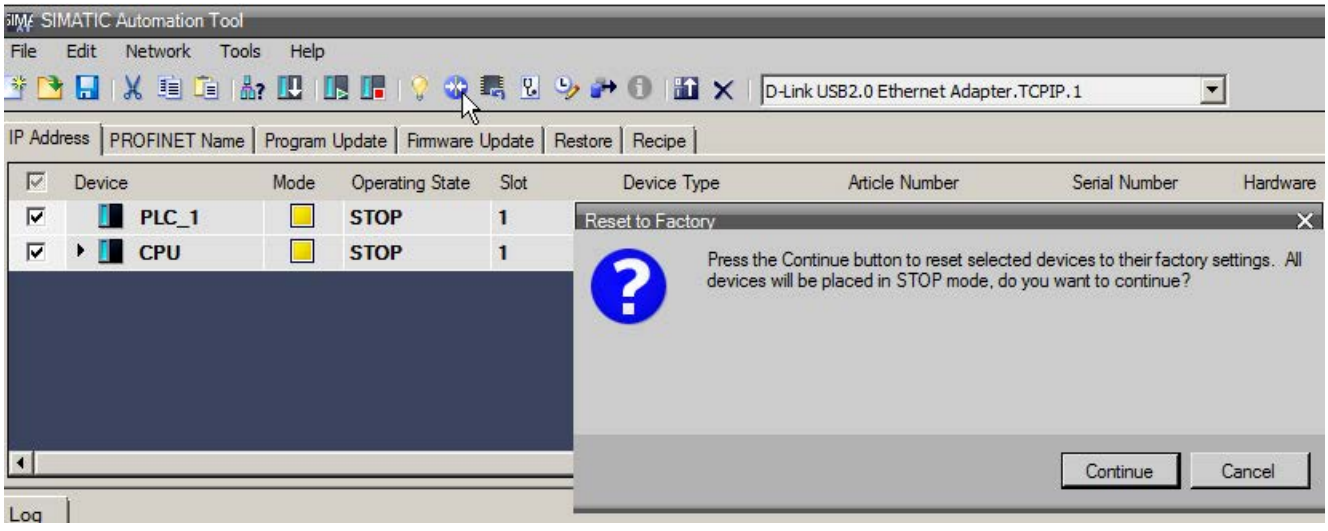
For safety reasons, the Reset to factory defaults operation is not allowed on fail-safe devices.

By default, the reset factory defaults operation allows only a single processing thread that must complete execution, before the next device operation is executed. To change this behavior to multi-thread processing, see the Tools>Options>Communications (Page 81) topic. Be aware of the risk of communication disruption that is described in that topic.

Use the following steps to reset selected devices to factory default values.

1. Click the left-side check box on devices to include in the operation. You can use the top check box, right-click shortcut menu, or the Edit menu for "Select All" and "Unselect All" commands.
2. Select "Reset Factory Defaults" from the Network menu or click the "Reset Factory Defaults" toolbar button.
3. Click the "Continue" button on the "Reset to Factory" dialog box.
4. Selected devices are reset to factory default values.
5. The Event log below the device table shows the results of this operation.
6. Allow time for the reset to complete. Wait until the device lights stop flashing before attempting another operation.

The selected devices in the following image are reset to factory default values, when the Reset Factory Defaults command is executed.



4.14 Reset CPU memory

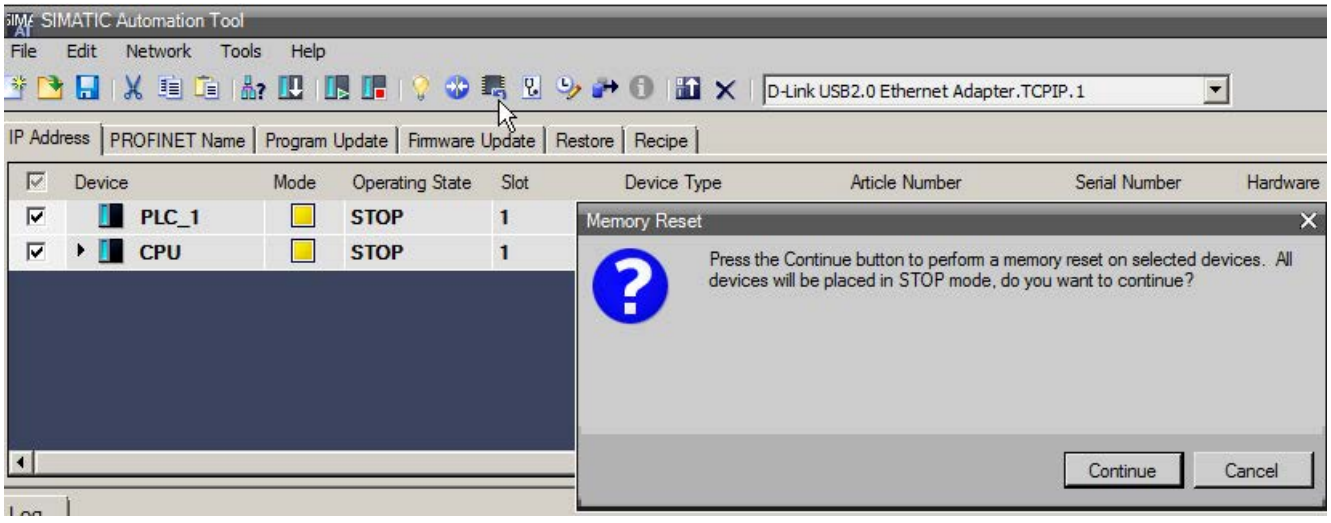
Reset memory on selected CPUs

Use the following steps to reset CPU memory on selected devices.

By default, the memory reset operation allows only a single processing thread that must complete execution, before the next device operation is executed. To change this behavior to multi-thread processing, see the Tools>Options>Communications (Page 81) topic. Be aware of the risk of communication disruption that is described in that topic.

1. Click the left-side check box on devices to include in the operation. You can use the top check box, right-click shortcut menu, or the Edit menu for "Select All" and "Unselect All" commands.
2. Select "Memory Reset" from the Network menu or click the "Memory Reset" toolbar button.
3. Click the "Continue" button on the "Memory Reset" dialog box.
4. Selected CPUs perform a memory reset operation.
5. The Event log below the device table shows the results of this operation.

The selected devices in the following image perform a memory reset when the Memory Reset command is executed.



4.15 Upload service data from CPUs

Get service data from selected CPUs

When a CPU enters a defective state, fault information is saved in the CPU 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.

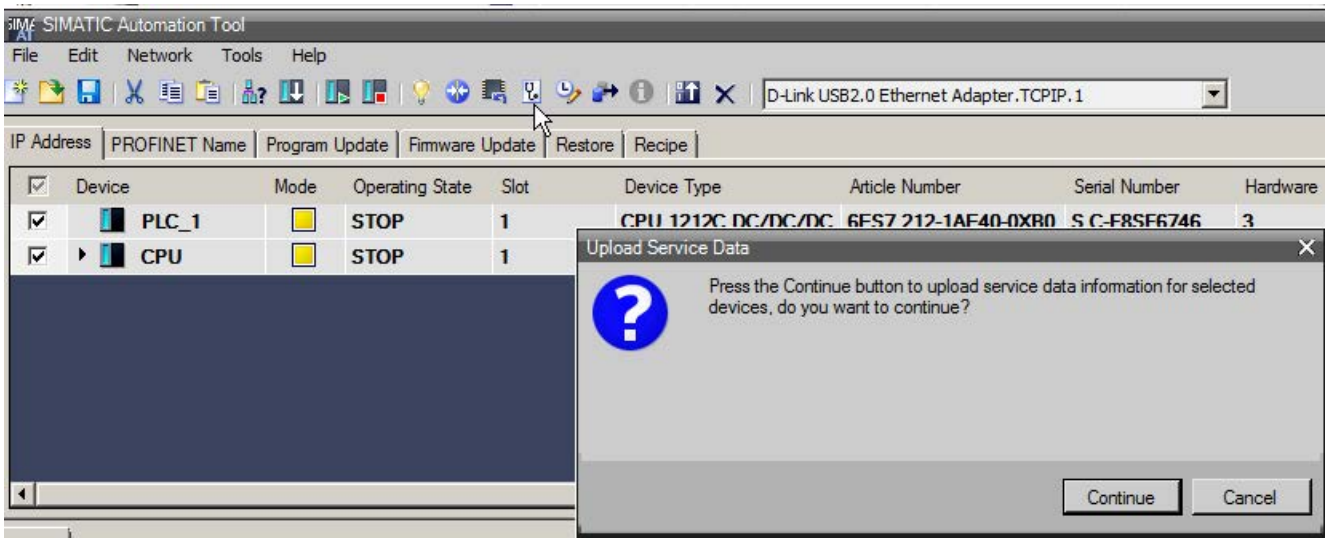
You can upload Service data in STOP or RUN mode. The Service data contains multiple files that are compressed into a single .zip file with a file name based on the PLC name, date, and time. A unique number in parentheses is appended to the file name to avoid duplicate file names.

The default Service data path is C:\Users\MyAccount\SIMATIC Automation Tool\ServiceData.
Your path may have a different drive letter and "MyAccount" represents the login name of the current user.

Use the following steps to upload service data from selected CPUs.

1. Click the left-side check box on devices to include in the operation. You can use the top check box, right-click shortcut menu, or the Edit menu for "Select All" and "Unselect All" commands.
2. Select "Upload Service data" from the Network menu or click the "Upload service data" toolbar button.
3. Click the "Continue" button on the "Upload Service Data" dialog box.
4. Selected CPUs upload service data to the file path assigned with the Tools>Options>Service Data dialog box.
5. The Event log below the device table shows the results of this operation.

The selected devices in the following image upload service data when the Upload Service Data command is executed.



Service data files

Note

Uploading 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 upload the service data files. Enter CPU passwords in the SIMATIC Automation Tool's "Password" column, before the upload service data operation is executed.

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

.zip file contents:

ResourceStats.txt

RAM.img

PLCInformation.txt

NAND.img

General.txt

Fault.bin

DNN.txt

CommBuffers.txt

ASLog.txt

Alarms.txt

Note

Service data is stored in clear text

A malicious user could use the service data files to obtain status and configuration details about the control system. The service data files are stored in clear text on the CPU (binary encoding). A CPU password can control access to this information.

Use the TIA portal device configuration to set up CPU protection with a strong password. Strong passwords are at least eight 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.

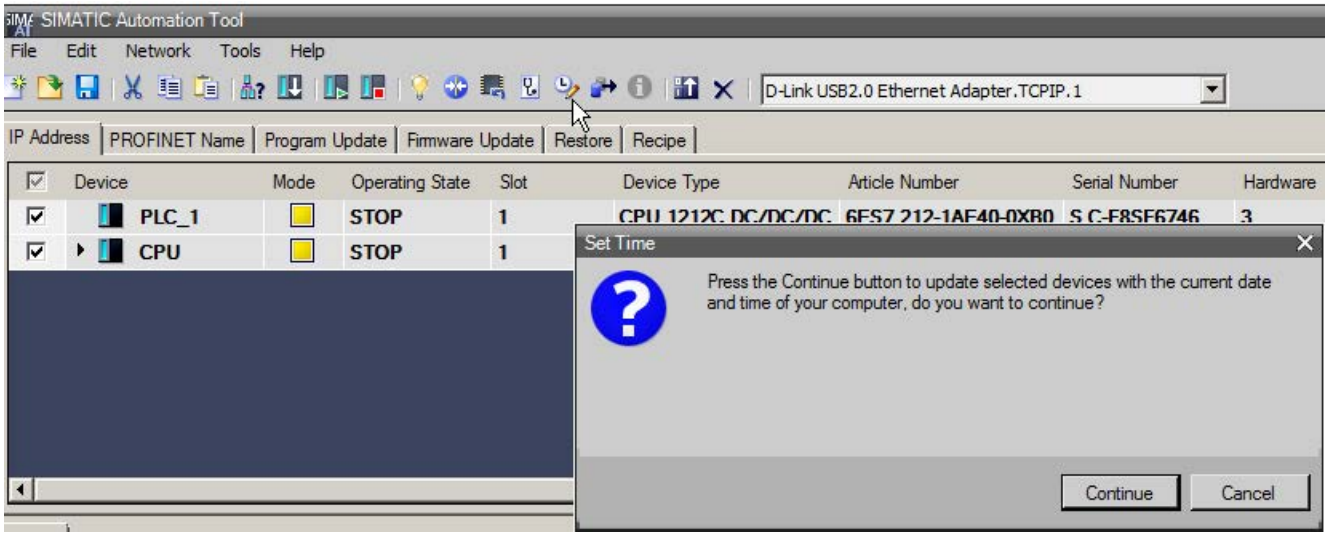
4.16 Set time in CPUs

Set time in CPUs to current PG/PC time

The Time button will set 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.

1. Click the left-side check box on devices to include in the operation. You can use the top check box, right-click shortcut menu, or the Edit menu for "Select All" and "Un-select All" commands.
2. Select "Set Time" from the Network menu or click the "Set time in devices to PC time" toolbar button.
3. Click the "Continue" button on the "Set Time" dialog box.
4. The system time on selected CPUs is set to your current PG/PC time.
5. The Event log below the device table shows the results of this operation.

The selected devices in the following image set their time to your current PG/PC time when the Set Time command is executed.



4.17 Read diagnostic buffer in a CPU

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 entries are displayed in chronological order with the most recent event at the top. Up to 50 most recent events are available in this log. When the log is full, a new event replaces the oldest event in the log. When power is lost, the events are saved.

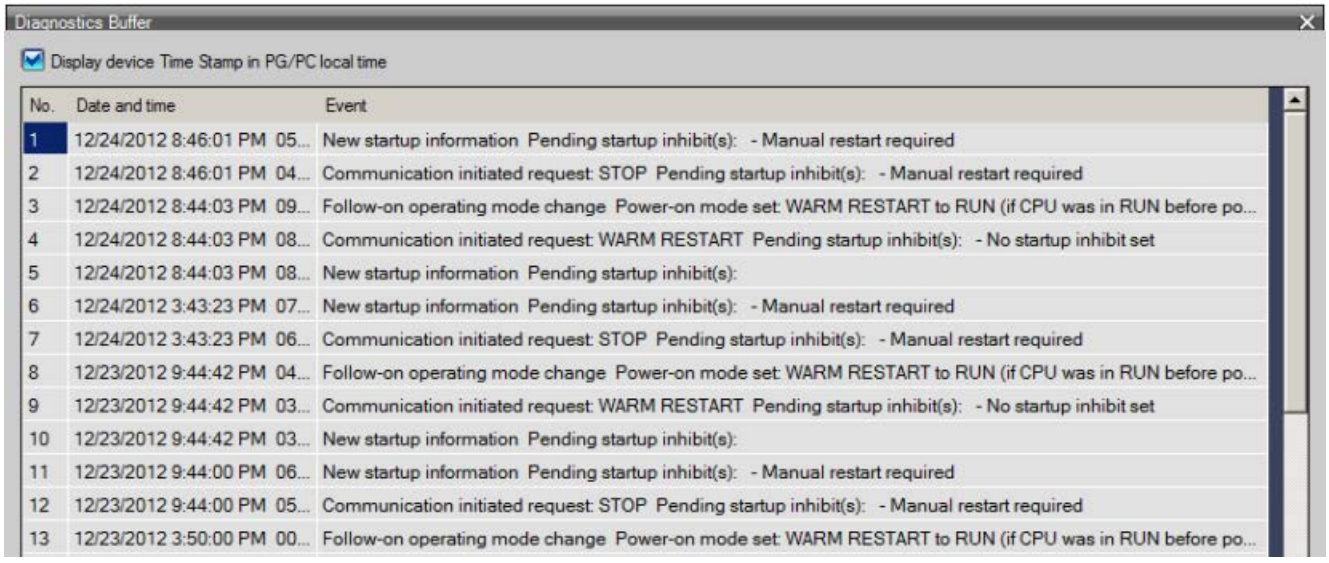
1. Click a check box in the "Select" column, for **one** CPU. You can use the right-click shortcut menu or the Edit menu, for the "Unselect All" command.
2. Select "Diagnostics buffer" from the Network menu or click the "Diagnostics" button on the toolbar.
3. The SIMATIC Automation Tool will display the contents of a CPU's diagnostics log.

Example diagnostic log

The following types of events are recorded in the diagnostics buffer.

- 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).



4.18 Execution order of operations

Operations are initiated with a toolbar button or menu item. For each toolbar button press, a single operation is added to the operations queue, for each selected device row. For example, if 20 different CPUs are selected and the RUN button is pressed, then 20 RUN operations are added to the queue.

For better performance, separate threads can run independently to initiate and execute the operations contained in the queue. The number of concurrent threads allowed is assigned in the Tools>Options>Communications dialog box. Separate threads are not allowed to simultaneously start jobs on one CPU, to avoid race conditions where one job is putting the CPU in STOP and another job is placing the same CPU in RUN.

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 **Save/Save as** commands or click the Save button to store your device table information in a protected .sat file. Once the SIMATIC Automation Tool project is saved, you can use the **File>Open** command to restore this project's device table information.

- The .sat file save path is assigned by the Tools>Options>Projects dialog box.
  The default path is C:\Users\MyAccount\SIMATIC Automation Tool\Projects.
  Your path may have a different drive letter and "MyAccount" represents the login name of the current user.
  You can modify this path
- 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 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.

SIMATIC Automation Tool password rules

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

Integrity check for .sat file

Before opening a project, an internal checksum test verifies that the file data has not changed, since the last SIMATIC Automation Tool save operation.

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

- 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 12 "," comma characters on each line of text. 12 comma characters separate the 13 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.

Note

CPU passwords are not exported

When you **export** a device table the tenth .csv file column (Password) is empty for security reasons.

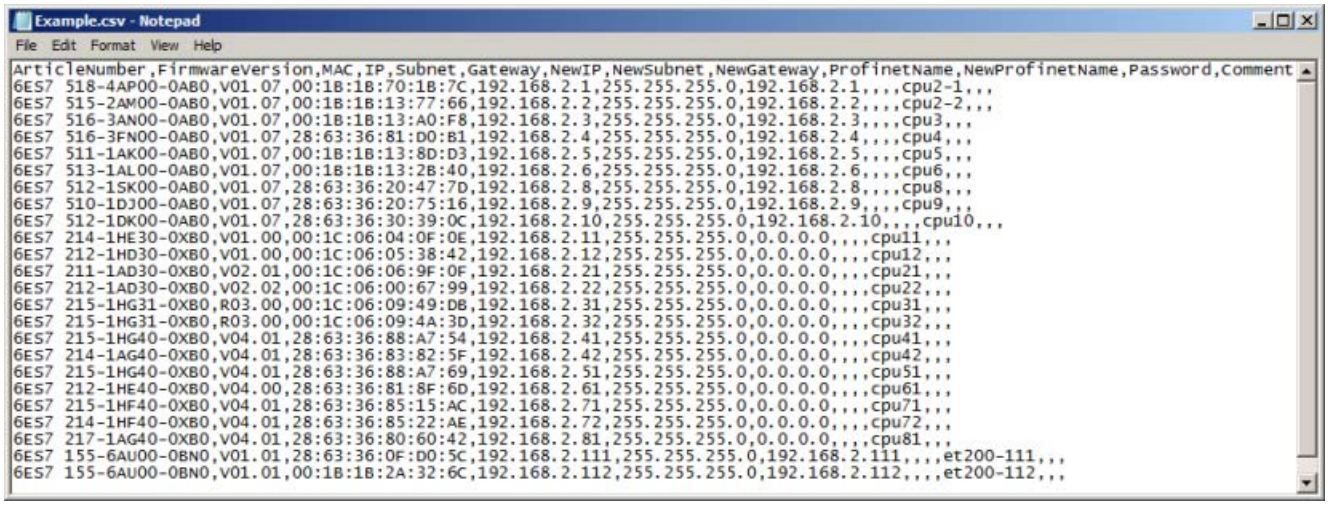
You can edit a .csv file, add in the passwords, and then **import** the .csv file. The passwords then appear in the SIMATIC Automation Tool device table.

The .csv file path for import and export operations is assigned by the Tools>Options>Projects dialog box.

The default path is C:\Users\MyAccount\SIMATIC Automation Tool\Projects.
Your path may have a different drive letter and "MyAccount" represents the login name of the current user.
You can modify this path.

Export example

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



Example.csv - Notepad

File Edit Format View Help

```
ArticleNumber,FirmwareVersion,MAC,IP,Subnet,Gateway,NewIP,NewSubnet,NewGateway,ProfinetName,NewProfinetName,Password,Comment
6ES7 518-4AP00-0AB0,V01.07,00:1B:1B:70:1B:7C,192.168.2.1,255.255.255.0,192.168.2.1,,,,cpu2-1,,,
6ES7 515-2AM00-0AB0,V01.07,00:1B:1B:13:77:66,192.168.2.2,255.255.255.0,192.168.2.2,,,,cpu2-2,,,
6ES7 516-3AN00-0AB0,V01.07,00:1B:1B:13:A0:F8,192.168.2.3,255.255.255.0,192.168.2.3,,,,cpu3,,,
6ES7 516-3FN00-0AB0,V01.07,28:63:36:81:D0:B1,192.168.2.4,255.255.255.0,192.168.2.4,,,,cpu4,,,
6ES7 511-1AK00-0AB0,V01.07,00:1B:1B:13:8D:D3,192.168.2.5,255.255.255.0,192.168.2.5,,,,cpu5,,,
6ES7 513-1AL00-0AB0,V01.07,00:1B:1B:13:2B:40,192.168.2.6,255.255.255.0,192.168.2.6,,,,cpu6,,,
6ES7 512-1SK00-0AB0,V01.07,28:63:36:20:47:7D,192.168.2.8,255.255.255.0,192.168.2.8,,,,cpu8,,,
6ES7 510-1DJ00-0AB0,V01.07,28:63:36:20:75:16,192.168.2.9,255.255.255.0,192.168.2.9,,,,cpu9,,,
6ES7 512-1DK00-0AB0,V01.07,28:63:36:30:39:0C,192.168.2.10,255.255.255.0,192.168.2.10,,,,cpu10,,,
6ES7 214-1HE30-0XB0,V01.00,00:1C:06:04:0F:0E,192.168.2.11,255.255.255.0,0.0.0.0,,,,cpu11,,,
6ES7 212-1HD30-0XB0,V01.00,00:1C:06:05:38:42,192.168.2.12,255.255.255.0,0.0.0.0,,,,cpu12,,,
6ES7 211-1AD30-0XB0,V02.01,00:1C:06:06:9F:0F,192.168.2.21,255.255.255.0,0.0.0.0,,,,cpu21,,,
6ES7 212-1AD30-0XB0,V02.02,00:1C:06:00:67:99,192.168.2.22,255.255.255.0,0.0.0.0,,,,cpu22,,,
6ES7 215-1HG31-0XB0,R03.00,00:1C:06:09:49:DB,192.168.2.31,255.255.255.0,0.0.0.0,,,,cpu31,,,
6ES7 215-1HG31-0XB0,R03.00,00:1C:06:09:4A:3D,192.168.2.32,255.255.255.0,0.0.0.0,,,,cpu32,,,
6ES7 215-1HG40-0XB0,V04.01,28:63:36:88:A7:54,192.168.2.41,255.255.255.0,0.0.0.0,,,,cpu41,,,
6ES7 214-1AG40-0XB0,V04.01,28:63:36:83:82:5F,192.168.2.42,255.255.255.0,0.0.0.0,,,,cpu42,,,
6ES7 215-1HG40-0XB0,V04.01,28:63:36:88:A7:69,192.168.2.51,255.255.255.0,0.0.0.0,,,,cpu51,,,
6ES7 212-1HE40-0XB0,V04.00,28:63:36:81:8F:6D,192.168.2.61,255.255.255.0,0.0.0.0,,,,cpu61,,,
6ES7 215-1HF40-0XB0,V04.01,28:63:36:85:15:AC,192.168.2.71,255.255.255.0,0.0.0.0,,,,cpu71,,,
6ES7 214-1HF40-0XB0,V04.01,28:63:36:85:22:AE,192.168.2.72,255.255.255.0,0.0.0.0,,,,cpu72,,,
6ES7 217-1AG40-0XB0,V04.01,28:63:36:80:60:42,192.168.2.81,255.255.255.0,0.0.0.0,,,,cpu81,,,
6ES7 155-6AU00-0BN0,V01.01,28:63:36:0F:D0:5C,192.168.2.111,255.255.255.0,192.168.2.111,,,,et200-111,,,
6ES7 155-6AU00-0BN0,V01.01,00:1B:1B:2A:32:6C,192.168.2.112,255.255.255.0,192.168.2.112,,,,et200-112,,,
```

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
1	ArticleNumber	FirmwareVersion	MAC	IP	Subnet	Gateway	NewIP	NewSubnet	NewGateway	ProfinetName	NewProfinetName	Password	Comment
2	6ES7 518-4AP00-0AB0	V01.07	00:1B:1B:70:1B:7C	192.168.2.1	255.255.255.0	192.168.2.1				cpu2-1			
3	6ES7 515-2AM00-0AB0	V01.07	00:1B:1B:13:77:66	192.168.2.2	255.255.255.0	192.168.2.2				cpu2-2			
4	6ES7 516-3AN00-0AB0	V01.07	00:1B:1B:13:A0:F8	192.168.2.3	255.255.255.0	192.168.2.3				cpu3			
5	6ES7 516-3FN00-0AB0	V01.07	28:63:36:81:D0:B1	192.168.2.4	255.255.255.0	192.168.2.4				cpu4			
6	6ES7 511-1AK00-0AB0	V01.07	00:1B:1B:13:8D:D3	192.168.2.5	255.255.255.0	192.168.2.5				cpu5			
7	6ES7 513-1AL00-0AB0	V01.07	00:1B:1B:13:2B:40	192.168.2.6	255.255.255.0	192.168.2.6				cpu6			
8	6ES7 512-1SK00-0AB0	V01.07	28:63:36:20:47:7D	192.168.2.8	255.255.255.0	192.168.2.8				cpu8			
9	6ES7 510-1DJ00-0AB0	V01.07	28:63:36:20:75:16	192.168.2.9	255.255.255.0	192.168.2.9				cpu9			
10	6ES7 512-1DK00-0AB0	V01.07	28:63:36:30:39:0C	192.168.2.10	255.255.255.0	192.168.2.10				cpu10			
11	6ES7 214-1HE30-0XB0	V01.00	00:1C:06:04:0F:0E	192.168.2.11	255.255.255.0	0.0.0.0				cpu11			
12	6ES7 212-1HD30-0XB0	V01.00	00:1C:06:05:38:42	192.168.2.12	255.255.255.0	0.0.0.0				cpu12			
13	6ES7 211-1AD30-0XB0	V02.01	00:1C:06:06:9F:0F	192.168.2.21	255.255.255.0	0.0.0.0				cpu21			
14	6ES7 212-1AD30-0XB0	V02.02	00:1C:06:00:67:99	192.168.2.22	255.255.255.0	0.0.0.0				cpu22			
15	6ES7 215-1HG31-0XB0	R03.00	00:1C:06:09:49:DB	192.168.2.31	255.255.255.0	0.0.0.0				cpu31			
16	6ES7 215-1HG31-0XB0	R03.00	00:1C:06:09:4A:3D	192.168.2.32	255.255.255.0	0.0.0.0				cpu32			
17	6ES7 215-1HG40-0XB0	V04.01	28:63:36:88:A7:54	192.168.2.41	255.255.255.0	0.0.0.0				cpu41			
18	6ES7 214-1AG40-0XB0	V04.01	28:63:36:83:82:5F	192.168.2.42	255.255.255.0	0.0.0.0				cpu42			
19	6ES7 215-1HG40-0XB0	V04.01	28:63:36:88:A7:69	192.168.2.51	255.255.255.0	0.0.0.0				cpu51			
20	6ES7 212-1HE40-0XB0	V04.00	28:63:36:81:8F:6D	192.168.2.61	255.255.255.0	0.0.0.0				cpu61			
21	6ES7 215-1HF40-0XB0	V04.01	28:63:36:85:15:AC	192.168.2.71	255.255.255.0	0.0.0.0				cpu71			
22	6ES7 214-1HF40-0XB0	V04.01	28:63:36:85:22:AE	192.168.2.72	255.255.255.0	0.0.0.0				cpu72			
23	6ES7 217-1AG40-0XB0	V04.01	28:63:36:80:60:42	192.168.2.81	255.255.255.0	0.0.0.0				cpu81			
24	6ES7 155-6AU00-0BN0	V01.01	28:63:36:0F:D0:5C	192.168.2.111	255.255.255.0	192.168.2.111				et200-111			
25	6ES7 155-6AU00-0BN0	V01.01	00:1B:1B:2A:32:6C	192.168.2.112	255.255.255.0	192.168.2.112				et200-112			

Menu, toolbar, and shortcut key reference

6

6.1 Main menu

6.1.1 File menu

Tool icon	Menu command	Description
	**New**	Creates a new SIMATIC Automation Tool project
	**Open**	An "Open" dialog is displayed that can browse to a folder, select an .sat project file, and provide a password to open a protected project file. The path is assigned in the Tools>Options>Projects dialog box.
	**Save**	The device table data is saved in a .sat file. If no filename and password are assigned, then this operation uses the "Save As" command.
	Save As...	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. The path is assigned in the Tools>Options>Projects dialog box.
	**Import...**	Fill the device table with data from a file in .csv format.
	**Export...**	Save the device table data to a file in .csv format.
	Exit	Close the application. If the project was modified since the last save operation, then the "Save" operation is performed.

See also

Save/Save as - Device table stored in protected .sat format (Page 73)

Import/Export - Device table loaded from/stored in open .csv format (Page 74)

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.
	Check All	Check (select) all rows of data on the visible tab.
	Uncheck All	Uncheck all rows on the visible tab.
	Expand All	Expand all rows for devices and modules.
	Collapse All	Collapse the rows for devices and modules.
	**Insert Device**	Insert a new device row at the selected row and push the following device rows downward. When a device cannot be discovered by a network scan, you can use this command to add the device to the device table. If you use this command to insert a device, the device name is colored blue. The blue color means that the MAC address based operations (flash LEDs, set IP address, and set PROFINET name) are not possible and the corresponding Device table cells are disabled.
	Delete	
	• **Device**	Delete one or more checked device rows.
	• **Selection**	Delete current selection in the device table.
	Refresh Devices F5	Refresh the checked devices.
	Check for Updates	Open the Siemens support Internet web page for the selected device.

6.1.3 Network menu

Tool icon	Menu command	Description	
	Scan	Scan the selected network interface for accessible CPUs and modules.	
	Download	Download data entered in the SIMATIC Automation Tool to CPUs on the network. The type of download depends on the current tab selection, Download types: • IP address parameters • PROFINET name • Program update • Firmware update • Restore device data from named .s7pbkp backup file • Recipe files	
	RUN	Put selected CPUs in RUN mode.	
	STOP	Put selected CPUs in STOP mode.	
	Flash LEDs	Flash the LEDs on selected devices. Use this feature to identify the physical location of a device.	
	Reset to factory defaults	Perform reset to factory default values on selected devices.	
	Memory reset	Perform a memory reset on selected CPUs.	
	Upload service data	Upload service information from CPUs.	
	Set time	Set time in selected CPUs to your PG/PC time.	
	Backup	Perform a backup operation on selected CPUs of all CPU data.	
	Diagnostics buffer	Read the diagnostics log from one CPU.	
	Remote file(s)	**Upload**	Upload Data log or Recipe file(s) from a CPU
		Delete	Delete Data log or Recipe file(s) in a CPU CPU must be in STOP mode for Delete Data log operation

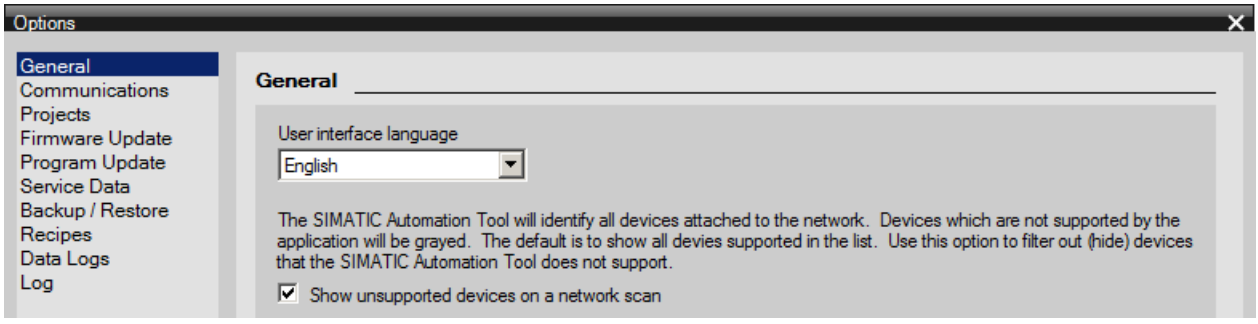
6.1.4 Tools options menu

6.1.4.1 General options

You can select the user interface language. English, German, French, Spanish, and Italian are available in V3.0.

Enable the check box to show unsupported devices on a network scan. Unsupported devices are displayed as disabled grayed rows in the device table.

Disable the check box to filter out unsupported devices so they are not displayed in the device table.



Note

Event log and user interface language change

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

6.1.4.2 Communications options

Allow multiple threads for firmware update, reset to factory defaults, memory reset, and restore device

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 enable the multiple threads option.

If your network has a chain topology, you should disable this option to prevent one CPU from disrupting the communication to other devices. For example, you have a chain connection (PG/PC to CPUa to CPUb to CPUc to ...). An ongoing operation for CPUb is disrupted when another thread causes a restart of CPUa.

Simultaneous operations

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

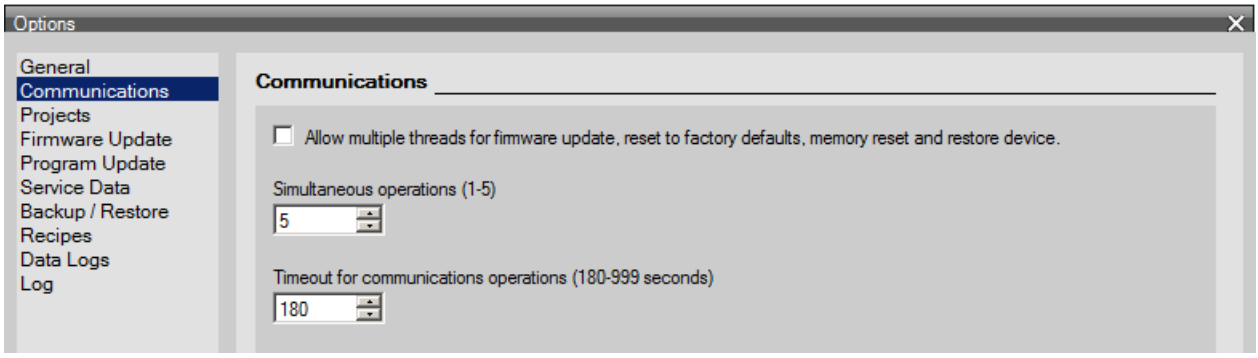
Note

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 maximum number of (threads/connections) that is assigned in the Tools>Options>Communications configuration for simultaneous operations. Close and restart the SIMATIC Automation Tool application, then try the group operation again.

Timeout for communications operations

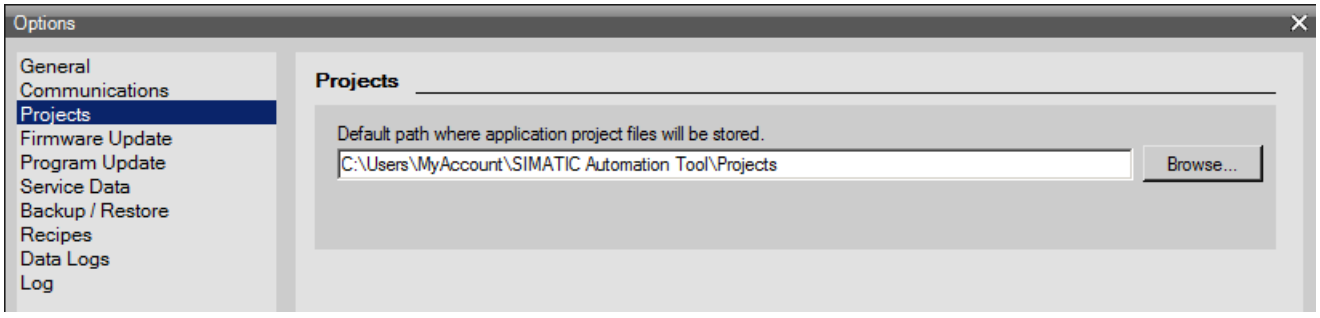
If you send an operation command to a device and the connection has a very slow data transfer rate, then you may get a communication timeout error. If you have this problem, then increase the time delay value that is assigned in the Tools>Options>Communications configuration for the communications timer.



6.1.4.3 Projects options

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

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

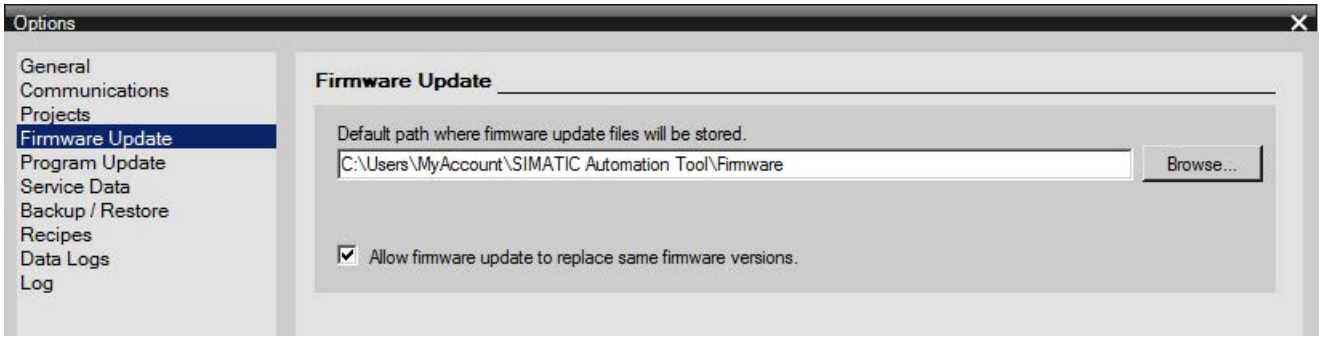


6.1.4.4 Firmware update options

You can accept the default path to firmware update files or assign a different path.

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

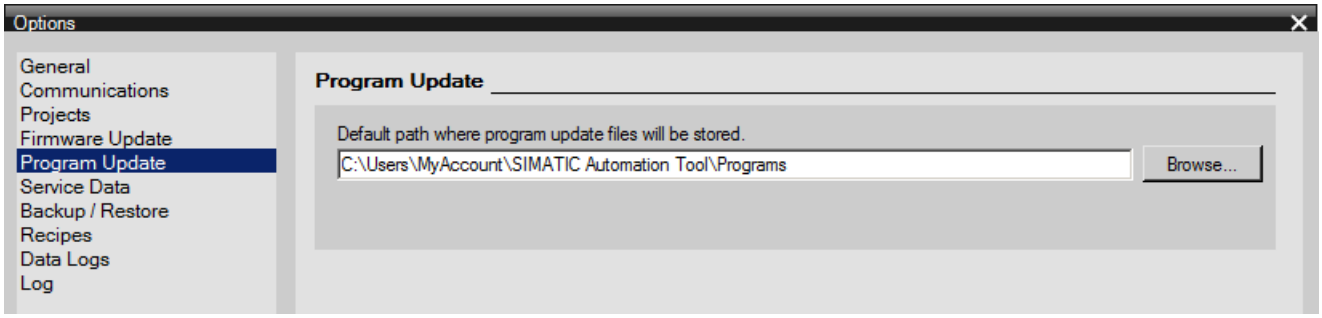
Click the check box to allow/disallow a firmware update with the same firmware version. This option can save time, by preventing unnecessary operations.



6.1.4.5 Program update options

You can accept the default path to program files or assign a different path.

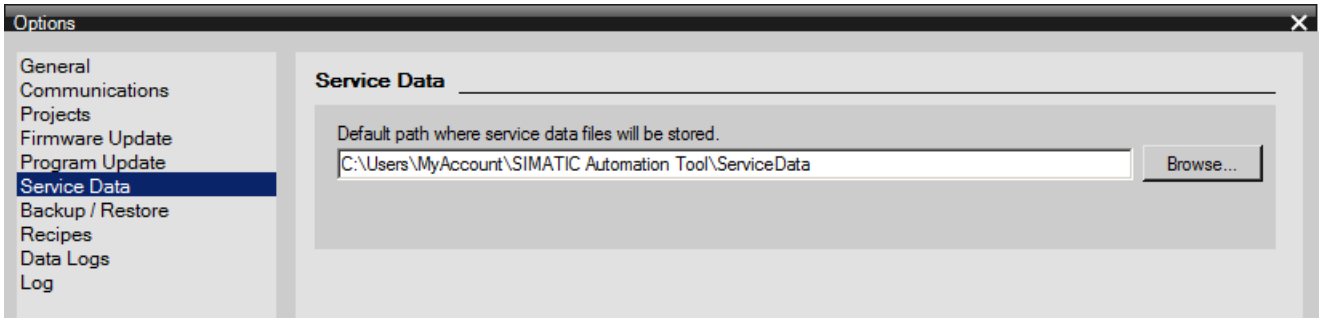
Your path may have a different drive letter and "MyAccount" represents the login name of the current user.



6.1.4.6 Service data options

You can accept the default path to Service Data files or assign a different path.

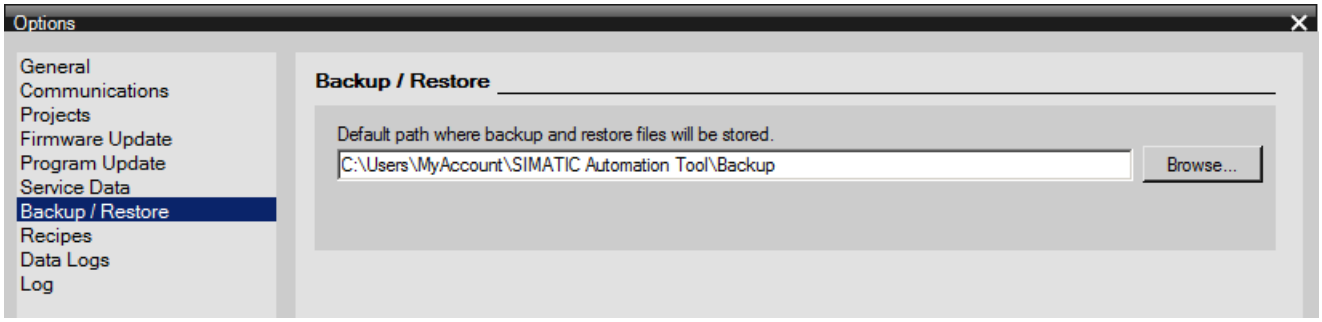
Your path may have a different drive letter and "MyAccount" represents the login name of the current user.



6.1.4.7 Backup/Restore options

You can accept the default path to Backup / Restore files or assign a different path.

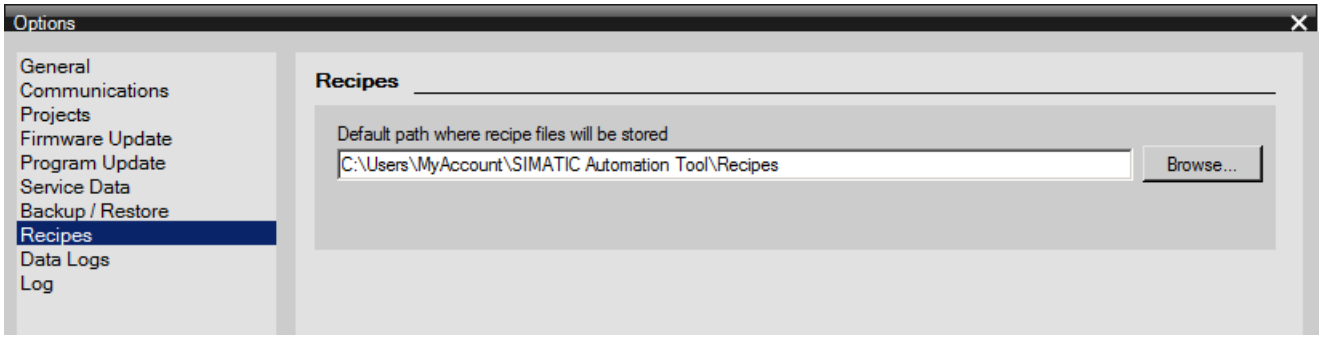
Your path may have a different drive letter and "MyAccount" represents the login name of the current user.



6.1.4.8 Recipes options

You can accept the default path to recipe files or assign a different path.

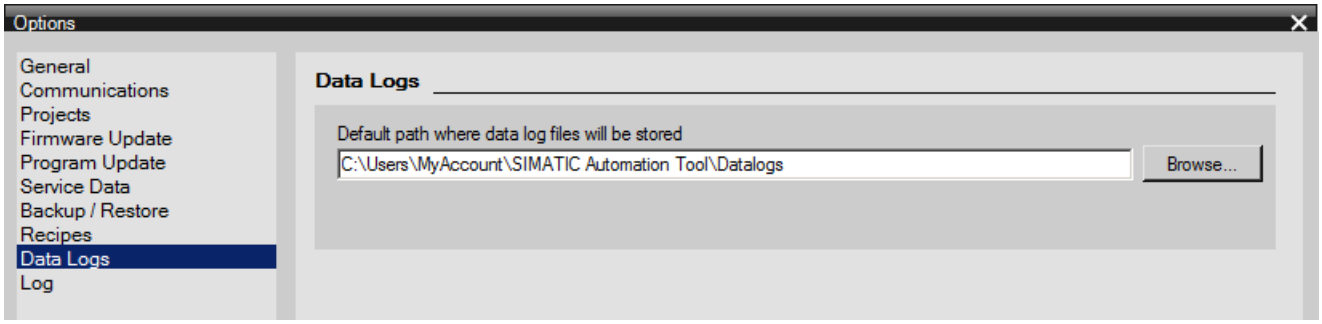
Your path may have a different drive letter and "MyAccount" represents the login name of the current user.



6.1.4.9 Data Logs options

You can accept the default path to Data Log files or assign a different path.

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



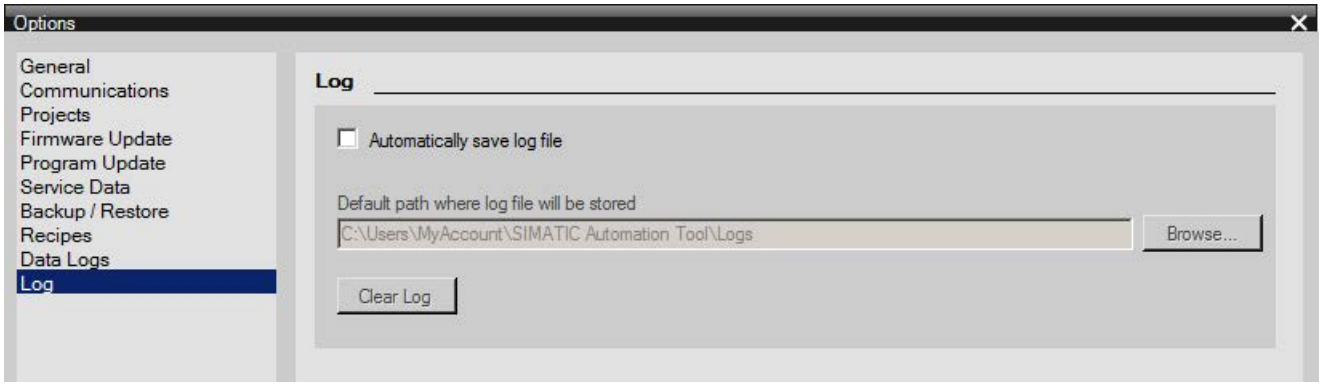
6.1.4.10 Log options

Enable the check box to create an event log file that shows the status of your SIMATIC Automation Tool operations.

When the log file check box is enabled, 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.

While this option is enabled, any actions that output a result to the log window are also mirrored to the file "EventLogFile.csv". When you close and re-open the SIMATIC Automation Tool, logging automatically resumes in the log file.

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



6.1.5 Help menu

Tool icon	Menu command	Description
	View user guide	Open the SIMATIC Automation Tool user guide.
	**About**	Displays the About dialog that contains: • Product name • Version

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 interface for accessible CPUs and modules.
	Download data entered in the SIMATIC Automation Tool to devices on the network. Depending on the current tab selection, either IP addresses, PROFINET names, program updates, firmware updates, restore data from a backup file, or recipe files are downloaded.
	RUN: Put selected CPUs in RUN mode.
	STOP: Put selected CPUs in STOP mode.
	Flash device LEDs or HMI screens, on selected devices. Use this feature to identify the physical location of a device.
	Reset factory default values in selected devices.
	Memory reset: Reset the memory on selected devices.
	Upload service data: Upload service information from a CPU.
	Set time: Set the system time in selected CPUs to current PG/PC time.
	Backup Device: Create Backup data for CPUs and HMI devices.
	Diagnostics: Read a CPU diagnostic buffer
	Upload selected CPU files (Recipes and Data logs). CPU can be in RUN or STOP mode.
	Delete selected CPU files (Recipes and Data logs). CPU must be in STOP mode.
	Network interface drop-down list: Select the Ethernet network interface that is connected to the industrial control network.

6.3 Shortcut keys

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 selected cells
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 of cells
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

SIMATIC Automation Tool API for .NET framework 7

7.1 API software license and version compatibility

API (Application Programming Interface)

The SIMATIC Automation Tool API allows you to create custom applications based on the functionality available in the SIMATIC Automation Tool application. A custom application can combine operations in a specific sequence and create workflows that are optimized for your industrial automation network. The following sections show the operations and data types that are provided through the API.

Software license required for V3.0 and later versions

The API is disabled for V3.0 or later versions, in unlicensed mode.

The API is enabled for V3.0 and later versions, when a license is detected by the Automation License Manager.

Compatibility with previous versions

The SIMATIC Automation Tool API was redesigned for V2.1. Programs written for previous versions of the API must be refactored.

The V3.0 API is backwards-compatible to the V2.1.1 API. New V3.0 API methods support managing CPU data files (Data log and Recipe), HMI device support, and multi-lingual operations.

Most applications written against the V2.1.1 API do not need changes to work with the V3.0 API.

One change that may be required concerns the property `IProfinetDevice.Supported`. This property is provided to indicate that a device is supported by the SIMATIC Automation Tool. In applications written against the SAT API V2.1.1, this property was a simple Boolean value. For V3.0, this has been added to the FeatureSupport enum.

As an example, the following code would have been used in V2.1.1 to iterate a device collection and ignore those devices that are not supported.

```
IProfinetDeviceCollection scannedDevices = new IProfinetDeviceCollection();
Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
if (retVal.Succeeded)
{
    foreach (IProfinetDevice dev in scannedDevices)
    {
        if (!dev.Supported)
            {
                continue;
            }
    }
}
```

Using the V3.0 API, this code would need to change to

```
IProfinetDeviceCollection scannedDevices = new IProfinetDeviceCollection();
Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
if (retVal.Succeeded)
{
    foreach (IProfinetDevice dev in scannedDevices)
    {
        if (dev.Supported != FeatureSupport.Supported)
        {
            continue;
        }
    }
}
```

See also

Network example (Page 163)

7.2 Architectural overview

Networks

The `.NET` class `Network` is used to perform 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 interface card to use. All SIMATIC Automation Tool communications to the industrial network use the network interface selected in this manner.

- Network constructor (Page 96)
- QueryNetworkInterfaceCards method (Page 97)
- SetCurrentNetworkInterface method (Page 98)
- CurrentNetworkInterface property (Page 98)
- ScanNetworkDevices method (Page 99)

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.

For example:

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

`IModuleCollection` – A collection that may represent all the IO modules plugged to a given CPU or I/O station.

`IHardwareCollection` – This collection may represent a CPU and all its I/O modules.

- IProfinetDeviceCollection class (Page 100)
- IProfinetDevice interface (Page 108)

- IProfinetDevice methods
  - RefreshStatus (Page 111)
  - FirmwareUpdate (Page 112)
  - FlashLED (Page 114)
  - Reset (Page 115)
  - SetIP (Page 116)
  - SetProfinetName (Page 117)
- IProfinetDevice events (Page 118)
- IModuleCollection class (Page 120)
- IModule interface (Page 121)
- ICPU interface (Page 122)
- ICPU methods
  - Backup (Page 123)
  - DownloadRecipe (Page 125)
  - DeleteDataLog (Page 126)
  - DeleteRecipe (Page 128)
  - GetCurrentDateTime (Page 130)
  - GetDiagnosticsBuffer (Page 131)
  - GetOperatingState (Page 134)
  - MemoryReset (Page 135)
  - ProgramUpdate (Page 136)
  - ResetToFactory (Page 137)
  - Restore (Page 138)
  - SetOperatingState (Page 139)
  - SetCurrentDateTime (Page 141)
  - UploadDataLog (Page 142)
  - UploadRecipe (Page 144)
  - UploadServiceData (Page 146)
- IRemoteInterface properties (Page 147)
- IHMI interface (Page 151)
- IHMI methods
  - Backup (Page 152)
  - ProgramUpdate (Page 153)
  - Restore (Page 154)

Note

See the example (Page 163) industrial network and the SIMATIC Automation Tool API classes that are used to represent each network component.

7.3 Referencing the API in a customer application

The API is delivered as a single DLL:

```
AutomationToolAPI.dll
```

This .dll file was created with Microsoft Visual Studio 2015 SP2 Update 3 using the .NET framework 4.6.1. It can be used with applications created with this version of Visual Studio or later. 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 `using` statement referencing the API namespace.

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

HMI related files

These files are required for working with HMI devices and must exist in the same directory where the `AutomationToolAPI.dll` file is stored.

```
DeviceManagerClient.dll
hmitr.dm.client.proxy.dll
hmitr.dm.client.stub.exe
hmitr.ipc.dll
```

In order 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 will 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, the API dll (AutomationToolAPI.dll) can be placed in any folder on the PG/PC and used successfully.

7.4 Common support classes

7.4.1 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 an 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.

This class provides a way to encrypt a plain-text password that you can use to legitimize a CPU connection. Many of the code examples show a typical usage of this class. Most code examples instantiate an `EncryptedString` directly where it is used, as follows:

```
Result retVal = devAsCpu.RefreshStatus(new EncryptedString("password"));
```

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

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

Result retVal = devAsCpu.RefreshStatus(pwd);
retVal = devAsCpu.GetCurrentDateTime(pwd, out curTime);
```

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:

```
Result retVal = devAsCpu.RefreshStatus(new EncryptedString(""));
```

Or

```
Result retVal = devAsCpu.RefreshStatus(new EncryptedString(String.Empty));
```

The `EncryptedString` object does not store the user-assigned plain-text password. However, if your application codes passwords as literal strings (for example, `new EncryptedString("myPassword")`) the plain-text "`myPassword`" will be compiled into the user application, and may be visible to others using .NET reflection tools.

7.4.2 Fail-Safe password

You are not allowed to legitimize a connection to an S7 CPU using the fail-safe password. If you specify a fail-safe password for any `ICPU` method that accepts a password, the SIMATIC Automation Tool API method will fail and return the error:
`ErrorCode.LegitimizationFailsafeLevelNotAllowed`.

7.4.3 Result class

The Result Class

This 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.

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:

```
Result retVal = dev.RefreshStatus(new EncryptedString(""));
if (retVal.Succeeded)
{
    //------------------------
    // Continue operations....
    //------------------------
}
```

In other cases, you may need more information about the failure. To inspect the specific error, use the `Code` property, as follows:

```
Result retVal = devAsCpu.RefreshStatus(new EncryptedString(""));
if (retVal.Succeeded)
{
    //------------------------
    // Continue operations....
    //------------------------
}
else
{

    //--------------
    // What happened
    //--------------
switch (retVal.Code)
  {
      case ErrorCode.AccessDenied:
           break;
      case ErrorCode.TooManySessions:
           break;
  }
}
```

The Result class can also provide a string description of the error condition. The property `ErrorDescription` returns an English language string descriptor for the error condition.

```
String strError = result.ErrorDescription;
```

The Result class also provides a language-specific version of this information. The `GetDescription` method uses a Language value as a parameter.

```
String strError = result.GetDescription(Language language);
```

See also ErrorCode values (Page 157)

See also

Language (Page 160)

7.5 Network class

7.5.1 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.

Constructor

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 select the network interface to use.

7.5.2 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 95)

7.5.3 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.

```
Result retVal = myNetwork.SetCurrentNetworkInterface(interfaces[0]);
if (retVal.Succeeded)
{
  //------------------------------------------------
  // The action succeeded. Continue with operations.
  //------------------------------------------------
}
```

7.5.4 CurrentNetworkInterface property

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

```
string currentInterface = myNetwork.CurrentNetworkInterface;
```

Note

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

7.5.5 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.

```
IProfinetDeviceCollection scannedDevices = new IProfinetDeviceCollection();

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.

7.6 IProfinetDeviceCollection class

7.6.1 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.

Consider the example code from the `ScanNetworkDevices` method:

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

For those programmers that prefer array-like syntax, the items in `scannedDevices` can be accessed as follows:

```
if (retVal.Succeeded)
{
  for (int deviceIdx = 0; deviceIdx < scannedDevices.Count; deviceIdx++)
  {
  //-------------------------------------------------------------
  // Each item in the collection is an IProfinetDevice.
  //-------------------------------------------------------------
  IProfinetDevice dev = scannedDevices[deviceIdx];
  }
}
```

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

```
foreach (IProfinetDevice dev in scannedDevices)
{
  //-------------------------------------------------------------
  // The variable "dev" now represents the next collection item
  //-------------------------------------------------------------
}
```

7.6.2 Filtering items in the collection

7.6.2.1 Collection items

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

The collection may also contain different "categories" of devices (i.e. 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.

7.6.2.2 FilterByDeviceFamily method

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.

```
List<DeviceFamily> fams = new List<DeviceFamily> {
DeviceFamily.CPU1200,DeviceFamily.CPU1500 };
```

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

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

IProfinetDeviceCollection 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>` will result in the return of an empty collection.

7.6.2.3 FilterOnlyCPUs method

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.

```
IProfinetDeviceCollection scannedDevices = new IProfinetDeviceCollection();
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 122) chapter.

7.6.3 Finding a specific device in the collection

7.6.3.1 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.

```
uint targetIPAddress = 0xC0A80001; // 192.168.0.1
retVal = myNetwork.ScanNetworkDevices(out scannedDevices);

if (!retVal.Succeeded)
    return;

IProfinetDevice dev = scannedDevices.FindDeviceByIP(targetIPAddress);

if (dev != null)
{
  // Found it!
}
```

7.6.3.2 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"

retVal = myNetwork.ScanNetworkDevices(out scannedDevices);

if (!retVal.Succeeded)
  return;

IProfinetDevice dev = scannedDevices.FindDeviceByMAC(targetMAC);

if (dev != null)
{
  // Found it!
}
```

7.6.4 Serialization

7.6.4.1 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.

7.6.4.2 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:

```
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.

Note

No SIMATIC Automation Tool API classes provide storage for user-entered passwords. Therefore, when the contents of the `IProfinetDeviceCollection` are serialized to a data file, no password information is included in this serialization.

7.6.4.3 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 = new IProfinetDeviceCollection();

FileStream f = File.OpenRead("myDataFile.SAT");

retVal = devices.ReadFromStream(f);

f.Close();
```

7.6.5 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.

7.6.5.1 AddDeviceByIP method

Return type	Method name
`Result`	`AddDeviceByIp`

Parameters			
Name	Data type	Parameter type	Description
`ipAddress`	`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:

```
IProfinetDeviceCollection scannedDevices = new IProfinetDeviceCollection();
retVal = network.ScanNetworkDevices(out scannedDevices);

if (!retVal.Succeeded)
  return;

UInt32 missingDeviceIPAddress = 0xC0A80001; // 192.168.0.1

retVal = scannedDevices.AddDeviceByIp(missingDeviceIPAddress);
```

7.6.5.2 AddOfflineDevice method

Return type	Method name
Result	AddOfflineDevice

Parameters			
Name	Data type	Parameter type	Description
strArticleNumber	String	In	The article number for the device to add
strVersion	String	In	The firmware version for the device to add
mac	ulong	In	The MAC address for the device to add
ip	uint	In	The IP Address of the device to add
subnetMask	uint	In	The subnet mask address of the device to add
defaultGateway	uint	In	The gateway address of the device to add
strProfinetName	String	In	The Profinet name of the device to add

The following code scans the network, and then manually adds a device with specific information:

```
IProfinetDeviceCollection scannedDevices = new IProfinetDeviceCollection();
retVal = network.ScanNetworkDevices(out scannedDevices);

if (!retVal.Succeeded)
  return;

String orderNumber = @"6ES7 214-1AG40-0XB0";
String version = @"V4.2";
ulong missingMAC = 0x112233445566;             // "11:22:33:44:55:66"
uint missingDeviceIPAddress = 0xC0A80001;      // 192.168.0.1
uint missingDeviceSubnetAddress = 0xFFFFFF00;  // 255.255.255.0
uint missingDeviceGatewayAddress = 0xC0A80021; // 192.168.0.33
String profinetName = @"PLC_1";

retVal = scannedDevices.AddOfflineDevice(orderNumber, version, missingMAC,
                                         missingDeviceIPAddress,
                                         missingDeviceSubnetAddress,
                                         missingDeviceGatewayAddress,
                                         profinetName);
```

7.7 IProfinetDevice interface

7.7.1 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`	The order or MLFB number
`BackupAllowed`	`FeatureSupport`	Does the device support the Backup feature?
`ChangeModeAllowed`	`FeatureSupport`	Does the device support changing the mode (RUN/STOP)?
`Comment`	`string`	You can assign a comment for the device. This is used in the SIMATIC Automation Tool user interface and is not relevant to API operations.
`Configured`	`bool`	Does the DNN for the device show that it has been configured?
`DefaultGateway`	`uint`	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`	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`	A description of the hardware item based on the article number. This is the same description that you see in the TIA Portal. (i.e. "CPU-1215 DC/DC/DC")
`Failsafe`	`FeatureSupport`	Based on the Article number, is this a fail-safe device?
`Family`	`DeviceFamily`	What is the "family" of the device? See also: Device family enumeration (Page 156)
`FirmwareUpdateAllowed`	`FeatureSupport`	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.
`HardwareNumber`	`Int16`	The hardware version (FS Functional State) for the device

Property Name	Return Type	Description
IP	uint	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" SIMATIC Automation Tool V3.0 supports only IPv4 addresses. IPv6 addressing is not supported.
IPString	string	The IP Address of the device, represented as a string in the form "xx.xx.xx.xx"
MAC	ulong	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"
MACString	string	The unique MAC assigned to the device, represented as a string in the form "11:22:33:44:55:66".
MemoryResetAllowed	FeatureSupport	Does the device support a memory reset?
Modules	IModuleCollection	A collection of the modules plugged on the station. See also IModuleCollection class (Page 156)
Name	string	The name of the device.
NewFirmwareVersion	string	These properties are used in the SIMATIC Automation Tool user interface are not relevant for API operations.
NewDefaultGateway	string	
NewIP	string	
NewProfinetName	string	
NewProgramName	string	
NewRestoreName		
ProfinetName	string	The PROFINET name for the device.
ProgramUpdateAllowed	FeatureSupport	Does the device support a program update?
ResetToFactoryAllowed	FeatureSupport	Does the device support reset to factory settings?
RestoreAllowed	FeatureSupport	Does the device support the restore features?
Selected	bool	This property is used in the SIMATIC Automation Tool user interface and is not relevant to API operations.
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 and is not relevant to API operations.
StationNumber	uint	The station number of the device
SubSlot	uint	The subslot of the device: This value is used for pluggable submodules such as SB-1200.(Signal Board).
Supported	FeatureSupport	Is the detected network device supported by current SIMATIC Automation Tool API operations?

Property Name	Return Type	Description
`SubnetMask`	`uint`	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`	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

IModuleCollection class (Page 120)

7.7.2 IProfinetDevice methods

7.7.2.1 RefreshStatus method

Return type	Method name
`Result`	`RefreshStatus`

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

When the `IProfinetDeviceCollection` collection is created by calling the `ScanNetworkDevices` method, only a minimal amount of information is learned about each device. In order 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.

```
IProfinetDeviceCollection scannedDevices = new IProfinetDeviceCollection();

Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);
if (retVal.Succeeded)
{
  foreach (IProfinetDevice dev in scannedDevices)
  {
    Result retVal = dev.RefreshStatus(new EncryptedString(""));
    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 above passes an empty password to the method. This would only be appropriate for a device with no password protection. The example shows the `EncryptedString` class. This class is provided by the API to correctly encrypt a plain-text password before using it to legitimize the connection with the device.

See also The EncryptedString class (Page 94)

7.7.2.2 FirmwareUpdate method

Return type	Method name
Result	FirmwareUpdate

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.
strFile	string	In	A fully qualified path and filename for the update file.
hardwareID	uint	In	The hardware identifier of the module
bUpdateSameVersion	Bool	In	If true, the method will proceed with the update, even if the update file indicates that it 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.

```
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)
{
  Result retVal = dev.FirmwareUpdate(new EncryptedString(""),
                                     updateFile, 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.

```
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(new EncryptedString(""));
  if (!retVal.Succeeded)
    return;
  //-------------------------------
  // Search the modules on the CPU.
  //-------------------------------
  IModuleCollection mods = dev.Modules;

  foreach (IModule mod in mods)
  {
    if (mod.ArticleNumber == targetModule)
    {
      //---------------------------------------
      // Update firmware for matching module(s)
      //---------------------------------------
      dev.FirmwareUpdate(new EncryptedString(""),
                         updateFile, mod.ID, true);
    }
  }
}
```

Notice that the `FirmwareUpdate` method is called on the CPU. The `hardwareID` passed to the method indicates which module should be updated.

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.

7.7.2.3 FlashLED method

Return type	Method name
Result	FlashLED

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 for the device that uses the IP address 192.168.0.1.

```
uint targetIPAddress = 0xC0A80001; // 192.168.0.1
IProfinetDeviceCollection scannedDevices = new IProfinetDeviceCollection();

Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);

if (retVal.Succeeded)
{
  //-----------------------------------------------
  // Search for the device at that IP, and flash LED
  //-----------------------------------------------
  IProfinetDevice dev = scannedDevices.FindDeviceByIP(targetIPAddress);

  if (dev != null)
  {
    retVal = dev.FlashLED();
  }
}
```

7.7.2.4 Reset method

Return type	Method name
Result	Reset

This method is used to reset a device to its factory settings.

The following example calls the Reset method for a device at a specific IP address.

```
uint targetIPAddress = 0xC0A80001; // 192.168.0.1
IProfinetDeviceCollection scannedDevices = new IProfinetDeviceCollection();

Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);

if (retVal.Succeeded)
{
  //-------------------------------------------------------
  // Search for the device at that IP, and reset the device
  //-------------------------------------------------------
  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.

7.7.2.5 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 IP address on the device". This option may be named "SET IP address using a different method", depending on the TIA portal version that you use.

The following example searches for a device at a specified MAC address, and sets its IP address.

```
ulong targetMACAddress = 0x112233445566; // 11:22:33:44:55:66
IProfinetDeviceCollection scannedDevices = new IProfinetDeviceCollection();

Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);

if (retVal.Succeeded)
{
  //-------------------------------------------------
  // Search for the device at that MAC, and Set IP
  //-------------------------------------------------
  IProfinetDevice dev = scannedDevices.FindDeviceByMAC(targetMAC);

  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); // encodedIP can now be used
```

7.7.2.6 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 "Set PROFINET device name on the device option".

```
ulong targetMACAddress = 0x112233445566; // 11:22:33:44:55:66
IProfinetDeviceCollection scannedDevices = new IProfinetDeviceCollection();

Result retVal = myNetwork.ScanNetworkDevices(out scannedDevices);

if (retVal.Succeeded)
{
  //-----------------------------------------------------------
  // Search for the device at that MAC, and Set PROFINET Name
  //-----------------------------------------------------------
  IProfinetDevice dev = scannedDevices.FindDeviceByMAC(targetMAC);

  if (dev != null)
  {
    retVal = dev.SetProfinetName("new name");
  }
}
```

7.7.3 IProfinetDevice events

7.7.3.1 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)
  {
  }
}
```

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 156)

7.7.3.2 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 then automatically called when there is a change in the progress of the operation.

The following example shows how this can be used. 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.FirmwareUpdate(new EncryptedString(""), updateFile, 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)
    {
      percent = (double)e.Index / (double)e.Count;
      string sPercent = e.Action.ToString() + " -> " +
                        "Index = " +
                        e.Index.ToString() +
                        " Count = " +
                        e.Count.ToString() +
                        " progress " +
                        (percent * 100).ToString("0.##") + "%";
    }
}
```

The ProgressChangedEventArgs class

The `ProgressChanged` event handler will be 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 161)
`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

7.8 IModuleCollection class and module properties

7.8.1 IModuleCollection class

The `IProfinetDevice` interface provides information about any modules (signal modules, signal boards, CMs, CPs, etc) 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).

```
//-------------------------------------------------------
// To ensure the information is current and complete,
// first call RefreshStatus()
//-------------------------------------------------------
Result retVal = networkDevice.RefreshStatus(new EncryptedString(""));
if (retVal.Succeeded)
{
  //-------------------------------------------------------
  // The Modules property returns a collection of IModule
  //-------------------------------------------------------
  IModuleCollection modules = networkDevice.Modules;
  foreach (IModule mod in modules)
  {
    //-------------------------------------------------------
    // Get article number for every module on central station
    //-------------------------------------------------------
    string displayArticleNum = mod.ArticleNumber;
  }
}
```

7.8.2 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 or MLFB number
`Comment`	`string`	This allows the user to assign a comment for the device. This is used in the SIMATIC Automation Tool user interface and is not relevant to API operations.
`Configured`	`bool`	Does the DNN for the device show that it has been configured?
`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 ArticleNumber, Is this a fail-safe device?
`FirmwareUpdateAllowed`	`FeatureSupport`	Does this device support firmware update?
`FirmwareVersion`	`string`	The current firmware version of the device
`HardwareID`	`uint`	The unique identifier for every device and module in the station. This is used as the unique identifier when executing the `FirmwareUpdate` method.
`Name`	`string`	The name of the device
`NewFirmwareVersion`	`string`	This property is used in the SIMATIC Automation Tool user interface. It is not relevant for API operations.
`Selected`	`bool`	This property is used in the SIMATIC Automation Tool user interface. It is not relevant for API operations.
`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 and is not relevant to 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 (Signal Board).
`Supported`	`FeatureSupport`	Is the detected network device supported by current SIMATIC Automation Tool operations?

7.9 ICPU interface

7.9.1 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, decentralized I/O stations, HMI devices, SITOP power supplies, and other devices.

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.

```
IProfinetDeviceCollection scannedDevices = new IProfinetDeviceCollection();
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.

      // Use the ICPU interface 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`. This topic only provides details for properties/methods that are unique to the `ICPU` interface.

7.9.2 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.
`DataLogFolder`	`IRemoteFolder`	Information about Data logs stored on the CPU's removable SIMATIC Memory Card.
`RecipeFolder`	`IRemoteFolder`	Information about Recipes stored on the CPU's removable SIMATIC Memory Card.

See also RemoteInterfaces property (Page 147)

7.9.3 ICPU methods

7.9.3.1 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.

7.9.3.2 Backup method (ICPU interface)

Return type	Method name
`Result`	`Backup`

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.
`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. Not all CPUs support the backup/restore feature. The property `BackupAllowed` can be checked 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.

```
uint targetIPAddress = 0xC0A80001; // 192.168.0.1
string bkFile = @"C:\MyCPUBackupFile.s7pbkp";
IProfinetDeviceCollection devices = new IProfinetDeviceCollection();

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 == FeatureSupport.BackupAllowed)
       )
    {
      retVal = devAsCpu.Backup(new EncryptedString(""), bkFile);
    }
  }
}
```

7.9.3.3 DownloadRecipe method

Return type	Method name
`Result`	`DownloadRecipe`

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.
`strFile`	`string`	In	The complete path and filename of the recipe file to download (transfer from PG/PC to CPU memory card).

Use this method to add or replace a recipe .CSV file on the CPU 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 writes a recipe file to a CPU memory card.

```
uint targetIPAddress = 0xC0A80001; // 192.168.0.1
string rcpFile = @"C:\NewRecipe.csv";
IProfinetDeviceCollection devices = new IProfinetDeviceCollection();

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 ==
                                    FeatureSupport.RemoteRecipesAllowed)
           )
        {
            retVal = devAsCpu.DownloadRecipe(new EncryptedString(""), rcpFile);
        }
    }
}
```

NOTE: If a recipe with the same name already exists on the CPU memory card, then it is replaced.

7.9.3.4 DeleteDataLog method

Return type	Method name
`Result`	`DeleteDataLog`

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.
`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.

```
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 ==
FeatureSupport.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)
                    {
                        //----------------------------------------
                        // Delete the data log. The PLC CPU goes to STOP.
                        //----------------------------------------
                        devAsCpu.DeleteDataLog(new EncryptedString(), datalog.Name);
                    }
                }
            }
        }
    }
}
```

7.9.3.5 DeleteRecipe method

Return type	Method name
`Result`	`DeleteRecipe`

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.
`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.

```
Result retVal = myNetwork.ScanNetworkDevices(out devices);
if (retVal.Succeeded)
{
    foreach (IProfinetDevice dev in devices)
    {
        ICPU devAsCpu = dev as ICPU;
        if (devAsCpu != null)
        {
            if (devAsCpu.RemoteRecipesAllowed == FeatureSupport.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)
                    {
                        //----------------------------------------
                        // Delete the recipe
                        //----------------------------------------
                        devAsCpu.DeleteRecipe(new EncryptedString(),recipe.Name);
                    }
                }
            }
        }
    }
}
```

7.9.3.6 GetCurrentDateTime method

Return type	Method name
`Result`	`GetCurrentDateTime`

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.
`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 time.

```
uint targetIPAddress = 0xC0A80001; // 192.168.0.1

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))
  {

    DateTime curTime = new DateTime();

    retVal = devAsCpu.GetCurrentDateTime(new EncryptedString(""),
                                         out curTime);

    }
}
```

7.9.3.7 GetDiagnosticsBuffer method

Return type	Method name
`Result`	`GetDiagnosticsBuffer`

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.
`aDiagnosticsItems`	`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 a CPU. Each entry is represented as a `DiagnosticsItem`.

The following example searches the `IProfinetDeviceCollection` for a CPU at a specific IP address. When found, the diagnostics information is read from the CPU.

```
uint targetIPAddress = 0xC0A80001; // 192.168.0.1
List<DiagnosticsItem> aLogs = new List<DiagnosticsItem>();
IProfinetDeviceCollection devices = new IProfinetDeviceCollection();

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))
    {

      retVal = devAsCpu.GetDiagnosticsBuffer(new EncryptedString(""),
                                             out aLogs);

        if (retVal.Succeeded)
        {

          for (int idxLog = 0; idxLog < aLogs.Count; idxLog++)
          {

            string descr = aLogs[idxLog].Description1;

          }
      }
    }
  }
}
```

Note: The example above returns all strings in English. See the next example for returning strings in another supported language.

The DiagnosticsItem class

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

7.9.3.8 GetDiagnosticsBuffer method (language-specific)

Return type	Method name
`Result`	`GetDiagnosticsBuffer`

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.
`aDiagnosticsItems`	`List<DiagnosticsItem>`	Out	A collection of Diagnostics Items: Each item in the collection represents an entry in the diagnostics buffer.
`Language`	`Language`	In	Assigns the language used in returned string information.

This method reads the current diagnostics entries from a CPU. Each entry is represented as a `DiagnosticsItem`.

This is a language-specific version of the previous `GetDiagnnostisBuffer` example. The operation is the same as the previous example, except that you can assign a language for returned string data.

The Language enum is described in the API enumerations section.

The following example searches the `IProfinetDeviceCollection` for a CPU at a specific IP address. When found, the diagnostics information is read from the CPU (in German text).

```
uint targetIPAddress = 0xC0A80001; // 192.168.0.1
List<DiagnosticsItem> aLogs = new List<DiagnosticsItem>();
IProfinetDeviceCollection devices = new IProfinetDeviceCollection();

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))
        {
            retVal = devAsCpu.GetDiagnosticsBuffer(new EncryptedString(""),
                                                   out aLogs,
                                                   Language.German);
            if (retVal.Succeeded)
            {
                for (int idxLog = 0; idxLog < aLogs.Count; idxLog++)
                {
                    string descr = aLogs[idxLog].Description1;
                }
            }
        }
    }
}
```

7.9.3.9 GetOperatingState method

Return type	Method name
`OperatingState`	`GetOperatingState`

This method returns an `OperatingState`.

See also OperatingState enumeration (Page 160)

Note

Unlike other methods on the `ICPU` interface, `GetOperatingState` does not require a legitimized connection to the CPU. Therefore, no password is needed.

The following example queries for the current operating state for all CPUs on the industrial network. If any CPU is not in RUN, an error is shown.

```
IProfinetDeviceCollection devices = new IProfinetDeviceCollection();

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

{
  foreach (IProfinetDevice dev in devices)
  {

    ICPU devAsCpu = dev as ICPU;
    if (devAsCpu != null)

    {

      if (devAsCpu.GetOperatingState() != OperatingState.Run)
      {

        // DisplayError!

      }
    }
  }
}
```

7.9.3.10 MemoryReset method

Return type	Method name
`Result`	`MemoryReset`

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 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.

```
uint targetIPAddress = 0xC0A80001; // 192.168.0.1

IProfinetDeviceCollection devices = new IProfinetDeviceCollection();

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)
    {
      retVal = devAsCpu.MemoryReset(new EncryptedString(""));
    }
  }
}
```

7.9.3.11 ProgramUpdate method

Return type	Method name
`Result`	`ProgramUpdate`

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
`strPath`	`string`	In	A fully-qualified path to the folder containing the program card contents.

This method performs a program update on the CPU. The parameter `strPath` specifies a folder containing the program to load. To be successful, the program must have been generated with the TIA Portal and saved in "program memory card" format.

The following example searches the `IProfinetDeviceCollection` for a CPU at a specific IP address, and updates the program for that CPU.

```
uint targetIPAddress = 0xC0A80001; // 192.168.0.1
IProfinetDeviceCollection devices = new IProfinetDeviceCollection();

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)
    {

      retVal = devAsCpu.ProgramUpdate(new EncryptedString(""),
                                      @"c:\myFolder\ProgramUpdate");

    }
  }
}
```

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.

Note

The `ProgramUpdate` method is not allowed for an S7 fail-safe CPU. The SIMATIC Automation Tool API will block this operation for a fail-safe CPU. When the software determines that this method was called for a fail-safe CPU, a specific error (`ErrorCode.FailsafeAccessNotAllowed`) is returned.

7.9.3.12 ResetToFactory method

Return type	Method name
`Result`	`ResetToFactory`

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 defaults.

The following example searches the `IProfinetDeviceCollection` for a CPU at a specific IP address, and calls the `ResetToFactory` method.

```
uint targetIPAddress = 0xC0A80001; // 192.168.0.1
IProfinetDeviceCollection devices = new IProfinetDeviceCollection();

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)
    {

      retVal = devAsCpu.ResetToFactory(new EncryptedString(""));

    }
  }
}
```

Note

The `ResetToFactory` method is not allowed for an S7 fail-safe CPU. The SIMATIC Automation Tool API will block this operation for a fail-safe CPU. When the software determines that this method was called for a fail-safe CPU, a specific error (`ErrorCode.FailsafeAccessNotAllowed`) is returned.

7.9.3.13 Restore method (ICPU interface)

Return type	Method name
`Result`	`Restore`

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
`strFile`	`string`	In	A fully qualified path to the folder containing the program card contents.

This method is used to restore the information from a previous backup of the CPU. Not all CPUs support the backup/restore feature. The property `RestoreAllowed` can be checked 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 restore feature, then calls the `Restore` method.

```
uint targetIPAddress = 0xC0A80001; // 192.168.0.1
string bkFile = @"C:\MyCPUBackupFile.s7pbkp";
IProfinetDeviceCollection devices = new IProfinetDeviceCollection();

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 == FeatureSupport.RestoreAllowed)
       )
    {
    retVal = devAsCpu.Restore(new EncryptedString(""), bkFile);
    }
  }
}
```

Note

The `Restore` method is not allowed for an S7 fail-safe CPU. The SIMATIC Automation Tool API will block this operation for a fail-safe CPU. When the software determines that this method was called for a fail-safe CPU, a specific error (`ErrorCode.FailsafeAccessNotAllowed`) is returned.

7.9.3.14 SetOperatingState method

Return type	Method name
`Result`	`SetOperatingState`

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
`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. The property C`hangeModeAllowed` can be checked 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.

```
uint targetIPAddress = 0xC0A80001; // 192.168.0.1
IProfinetDeviceCollection devices = new IProfinetDeviceCollection();


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 == FeatureSupport.ChangeModeAllowed)
       )
   {
      retVal = devAsCpu.SetOperatingState(new EncryptedString(""),
                                          OperatingStateREQ.Run);
    }
  }
}
```

7.9.3.15 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.

```
IProfinetDeviceCollection devices = new IProfinetDeviceCollection();

Result retVal = myNetwork.ScanNetworkDevices(out devices);
if (retVal.Succeeded)
{
  foreach (IProfinetDevice dev in devices)
  {
    ICPU devAsCpu = dev as ICPU;

    if (devAsCpu != null)
    {
      retVal = devAsCpu.SetCurrentDateTime(new EncryptedString(""),
                                           DateTime.UtcNow);
    }
  }
}
```

7.9.3.16 UploadDataLog method

Return type	Method name
`Result`	`UploadDataLog`

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.
`strFileName`	`string`	In	The filename of the data log to upload from a CPU's removable SIMATIC memory card.
`strDestination-Folder`	`string`	In	Fully qualified path where the uploaded file Data log file is stored

This method uploads copies of Data log files from the 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.

```
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 ==
FeatureSupport.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.DataLogsFolder.Files)
                    {
                        //----------------------------------------
                        // Upload a copy of each data log.
                        //----------------------------------------
                        devAsCpu.UploadDataLog(new EncryptedString(""),
                                               datalog.Name,
                                               @"C:\MyDataLogs");
                    }
                }
            }
        }
    }
}
```

7.9.3.17 UploadRecipe method

Return type	Method name
`Result`	`UploadRecipe`

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.
`strFileName`	`string`	In	The filename of the recipe to upload from the CPU memory card.
`strDestination-Folder`	`string`	In	Fully qualified path where the uploaded Recipe file is stored

This method uploads copies of recipe files 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.

```
Result retVal = myNetwork.ScanNetworkDevices(out devices);
if (retVal.Succeeded)
{
    foreach (IProfinetDevice dev in devices)
    {
        ICPU devAsCpu = dev as ICPU;
        if (devAsCpu != null)
        {
            if (devAsCpu.RemoteRecipesAllowed == FeatureSupport.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)
                    {
                        //----------------------------------------
                        // Upload a copy of each recipe.
                        //----------------------------------------
                        devAsCpu.UploadRecipe(new EncryptedString(),
                                              recipe.Name,
                                              @"C:\MyRecipes");
                    }
                }
            }
        }
    }
}
```

7.9.3.18 UploadServiceData method

Return type	Method name
`Result`	`UploadServiceData`

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
`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.

```
uint targetIPAddress = 0xC0A80001; // 192.168.0.1
string strDiagFolder = @"c:\Diagnostics";
IProfinetDeviceCollection devices = new IProfinetDeviceCollection();

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.RefreshStatus(new EncryptedString(""));

      if (devAsCpu.GetOperatingState() == OperatingState.Defective)
      {
        retVal = devAsCpu.UploadServiceData(new EncryptedString(""),
                                            strDiagFolder);
      }
    }
  }
}
```

7.9.4 RemoteInterfaces properties

7.9.4.1 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.

To access information about decentralized IO, it is first necessary to call the `RefreshStatus` method on the CPU. This opens a legitimized connection with the CPU, reads the relevant information, and closes the connection.

The following example shows how to access this information for all the CPUs on a network.

```
retVal = myNetwork.ScanNetworkDevices(out devices);

if (retVal.Succeeded)
{
  foreach (IProfinetDevice dev in devices)
  {
    ICPU devAsCpu = dev as ICPU;

    //------------------------------------------------------------
    // A call to RefreshStatus() is needed to gather information
    // about decentral network(s)
    //------------------------------------------------------------
    retVal = devAsCpu.RefreshStatus(new EncryptedString(""));
    if (!retVal.Succeeded)
      return;

    List<IRemoteInterface> decentalNets = devAsCpu.RemoteInterfaces;

    foreach (IRemoteInterface net in decentalNets)
    {
      //-----------------------------
      // Inspect the remote interface
      //-----------------------------
    }
  }
}
```

7.9.4.2 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 161)
`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 or MLFB number
`BackupAllowed`	`FeatureSupport`	Does the device support the Backup feature?
`ChangeModeAllowed`	`FeatureSupport`	Does the device support changing the mode (RUN/STOP)?
`Comment`	`string`	This allows the user to assign a comment for the device. This is used in the SIMATIC Automation Tool user interface and is not relevant to API operations.
`Configured`	`bool`	Does the DNN for the device show that it has been configured?
`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 fail-safe device?
`Family`	`DeviceFamily`	What is the "family" of the device? For more information, refer to the description of the DeviceFamily enumeration.
`FirmwareUpdateAllowed`	`FeatureSupport`	Does this device support firmware update?

FirmwareVersion	string	The current firmware version of the device
HardwareID	uint	The unique identifier for every device and module in the station. This is used as the unique identifier when executing a FirmwareUpdate.
MemoryResetAllowed	FeatureSupport	Does the device support a memory reset?
Modules	IModuleCollection	A collection of the modules plugged on the station See also IModuleCollection class and module properties (Page 120)
Name	string	The name of the device
NewFirmwareVersion	string	This property is used in the SIMATIC Automation Tool user interface and is not relevant to API operations.
ProgramUpdateAllowed	FeatureSupport	Does the device support a program update?
ProjectData	Uint64	This property is used in the SIMATIC Automation Tool user interface and is not relevant to API operations.
ResetToFactoryAllowed	FeatureSupport	Does the device support reset to factory settings?
RestoreAllowed	FeatureSupport	Does the device support the restore features?
Selected	bool	This property is used in the SIMATIC Automation Tool user interface and is not relevant to API operations.
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 and is not relevant to 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 (Signal Board).
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:

```
retVal = myNetwork.ScanNetworkDevices(out scannedDevices);

if (!retVal.Succeeded)
  return;

foreach (IProfinetDevice dev in scannedDevices)
{
  ICPU devAsCpu = dev as ICPU;

  if (devAsCpu == null)
    continue;
  //------------------------------------------------------------
  // A call to RefreshStatus() is needed to gather information
  // about decentral network(s)
  //------------------------------------------------------------
  retVal = devAsCpu.RefreshStatus(new EncryptedString(""));

  if (!retVal.Succeeded)
    return;

  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 order numbers for all decentralized stations on the industrial network.

Since the `IBaseDevice` also supports the Modules property, it would be simple to extend the example further to look at not only the decentralized stations, but also all the local modules on each station.

7.10 IHMI interface

7.10.1 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.

```
IProfinetDeviceCollection scannedDevices = new IProfinetDeviceCollection();
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 interact with this device.
            //-----------------------------------------------------
        }
    }
}
```

NOTE: The `IHMI` interface inherits from `IProfinetDevice`. Therefore all the properties and methods supported on `IProfinetDevice` are also supported on `IHMI`. The following `IHMI` methods describe only the properties and 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`. You must execute the `ProgramUpdate` method to update firmware, operating system, and runtime software on HMI devices.

7.10.2 Backup method (IHMI interface)

Return type	Method name
`Result`	`Backup`

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.
`strFile`	`string`	In	A fully-qualified path and filename where the backup file is stored

This method is used to backup 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.

```
uint targetIPAddress = 0xC0A80001; // 192.168.0.1
string bkFile = @"C:\MyCPUBackupFile.s7pbkp";
IProfinetDeviceCollection devices = new IProfinetDeviceCollection();

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 == FeatureSupport.BackupAllowed)
            )
        {
            retVal = devAsHMI.Backup(new EncryptedString(""), bkFile);
        }
    }
}
```

7.10.3 ProgramUpdate method (IHMI interface)

Return type	Method name
`Result`	`ProgramUpdate`

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.
`strPath`	`string`	In	A fully-qualified path to the folder containing the program card contents

This method performs a full update on the HMI. The parameter `strPath` assigns a folder containing the program to load. The following example searches the `IProfinetDeviceCollection` for an HMI at a specific IP address, and updates the program for that HMI.

```
uint targetIPAddress = 0xC0A80001; // 192.168.0.1
IProfinetDeviceCollection devices = new IProfinetDeviceCollection();

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)
        {
            retVal = devAsHMI.ProgramUpdate(new EncryptedString(""),
                                            @"c:\myFolder\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

HMI firmware, 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.

7.10.4 Restore method (IHMI interface)

Return type	Method name
`Result`	`Restore`

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.
`strFile`	`string`	In	A fully-qualified path and filename for the backup file to be restored.

Use this method to restore the information from a previous backup of the HMI 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.

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.

```
uint targetIPAddress = 0xC0A80001; // 192.168.0.1
string bkFile = @"C:\MyCPUBackupFile.s7pbkp";
IProfinetDeviceCollection devices = new IProfinetDeviceCollection();

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.RestoreAllowed == FeatureSupport.RestoreAllowed)
            )
        {
            retVal = devAsHMI.Restore(new EncryptedString(""), bkFile);
        }
    }
}
```

7.11 API enumerations

7.11.1 DataChangedType

This enumeration defines the possible argument values for the DataChangedEventHandler (Page 118).

```
Invalid
OperatingState
RackInformation
Folders
ProfinetName
IPAddress
```

7.11.2 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
```

7.11.3 ErrorCode

This enumeration lists all possible return values for a Result object.

```
Abort
AccessDenied
AddonsUnsupported
AttributeNotFound
BackupNotSupported
BufferToSmall
CertificateExpired
CertificateFailure
CertificateInvalid
CertificateNotReady
CertificateRevoked
ConnectionRequired
CPUFailedToEnterRunMode
DatalogNotSupported
DeviceDoesNotSupportFirmwareUpdate
DeviceIsNotAcceptingChanges
DeviceNotOnNetwork
Disconnected
DiskFull
DownloadInvalidRecipe
ErrorCreatingFile
ErrorCreatingFolder
ErrorReadingFromStream
ErrorWritingToFile
ErrorWritingToStream
Failed
FailedToDisconnect
FailedToSetIPAddress
FailedToSetProfinetName
FailedToZipFolderContents
FailsafeAccessNotAllowed
FileReadFailed
FileWriteFailed
FirmwareFileNotCompatible
FirmwareFileNotCompatibleBuildType
FirmwareFileNotCompatibleNotSame
FirmwareFileNotCompatibleSame
FirmwareFileNotCompatibleToNew
FirmwareFileNotCompatibleToOld
FirmwareIDNotFound
FirmwareModuleDeactivated
FirmwareModuleError
FirmwareModuleIOnotAvailable
FirmwareModuleMaintenanceDemanded
FirmwareModuleMaintenanceRequired
FirmwareModuleNotAccepted
FirmwareModuleNotReachable
FirmwareModuleUnknown
FirmwareUpdateModuleNotConfigured
```

```
FirmwareUpdateModuleNotSupported
FirmwareVersionDifferent
FirmwareVersionMatch
FirwareTypeNotInstalled
FirwareTypeNotSupported
FlashNotSupported
GatewayIsNotValid
GetTimeNotAllowed
HardwareSoftwareNotComplete
IncompatibleAddon
InvalidArguements
InvalidFileName
InvalidFirmwarePath
InvalidPath
InvalidPointer
InvalidProjectPath
InvalidProjectVersion
InvalidRuntimePath
InvalidSignature
InvalidTimeoutValue
InvalidVersion
IPAddressIsNotValid
LegitimizationFailsafeLevelNotAllowed
LicenseFailed
LogicalVolumneMissing
LogicalVolumneOutOfSpace
LogoutLoginRequired
MACAddressIsNotValid
MajorImageDowngrade
MajorImageUpgrade
MajorRuntimeDowngrade
MajorRuntimeUpgrade
MemoryResetNotSupported
MultiESConflict
MultiESIncompatibleOtherESVersion
MultiESLimitExceeded
MultiESNotSupported
NoDataToBackup
NoMemoryCardInDevice
NoRuntimeInstalled
NotChangableInRun
NotEnoughMemory
NotEnoughMemoryAvailable
NoValidLicense
ObjectNotFound
OK
OperationCanceledByUser
OperationNotSupportedByThisDevice
OutOfResources
OutOfSpace
PanelOrientationIsLandscape
PanelOrientationIsPortrait
```

```
ParmeterOutOfRange
ProfinetNameIsNotValid
ProgramUpdateNotSupported
ProjectCharacteristicsInvalid
ProjectCharacteristicsMissing
ProjectIPMismatch
ProjectNotCompatibleWithDevice
ReadDiagBufferNotAllowed
RecipesNotSupported
RemoteTransferDisabled
RescueBackupNotPossible
RescueRestoreNotPossible
ResetPerformedDownloadRequired
ResetToFactoryDefaultsNotSupported
RestoreNotSupported
RestoreRebootRequired
RuntimeBroken
RuntimeCorrupt
RuntimeMissing
SecurityLib
ServiceAborted
ServiceActive
ServiceNotConnected
ServiceTimeout
SessionDelegitimated
SetIPErrorDueProjectSettings
SetNameErrorDueProjectSettings
SetTimeNotAllowed
SignatureFailure
SignatureInvalid
SignatureRequired
StoreReadFailed
StoreWriteFailed
SubnetMaskIsNotValid
TooManyRequests
TooManySessions
TraceMeasurementsNotSupported
TypeConversionFailed
UnexpectedOperatingSystemError
UnknownAddon
UnknownApp
UnknownAppAddon
UnknownReferenceApp
UnsupportedDevice
UpdateProgramVersionGreaterPLCVersion
UploadServiceDataNotAllowed
WriteBlockFailed
WriteProtected
WrongDevicetype
WrongRuntime
WrongRuntimeVersion
FirmwareUpdateFileMissing
```

```
RecipeFileMissing
RestoreFileMissing
ProgramFolderMissing
```

7.11.4 Language

The Language enumeration allows you to assign the language for returned string data. It contains the following values:

```
English
German
French
Spanish
Italian
```

7.11.5 OperatingState

This enumeration defines the possible states that can be returned from a call to the GetOperatingState (Page 134) method.

```
NotSupported
StopFwUpdate
StopSelfInitialization
Stop
Startup
Run
RunRedundant
Halt
LinkUp
Update
Defective
ErrorSearch
NoPower
CiR
STOPwithoutODIS
RunODIS
```

7.11.6 OperatingStateREQ

This enumeration defines the possible state transitions that can be requested, on a call to the SetOperatingState (Page 139) method.

```
Stop
Run
```

7.11.7 ProgressAction

This enumeration defines the possible argument values that can be sent to a ProgressChangedEventHandler (Page 119).

```
Invalid
Connecting
Reconnecting
Disconnecting
Initializing
Updating
Processing
Downloading
Uploading
Deleting
Reseting
Rebooting
Verifying
Formatting
Finished
UpdatingFirmware
InstallingRuntime
InstallingAddOns
UninstallingAddOns
UpdatingProgram
```

7.11.8 RemoteInterfaceType

This enumeration defines the possible states that can be returned from a call to the `InterfaceType` property on the IRemoteInterfaces (Page 147) interface.

```
None
Profinet
Profibus
ASi
```

7.11.9 FeatureSupport

The SIMATIC Automation Tool provides this enumeration to indicate what features each device supports.

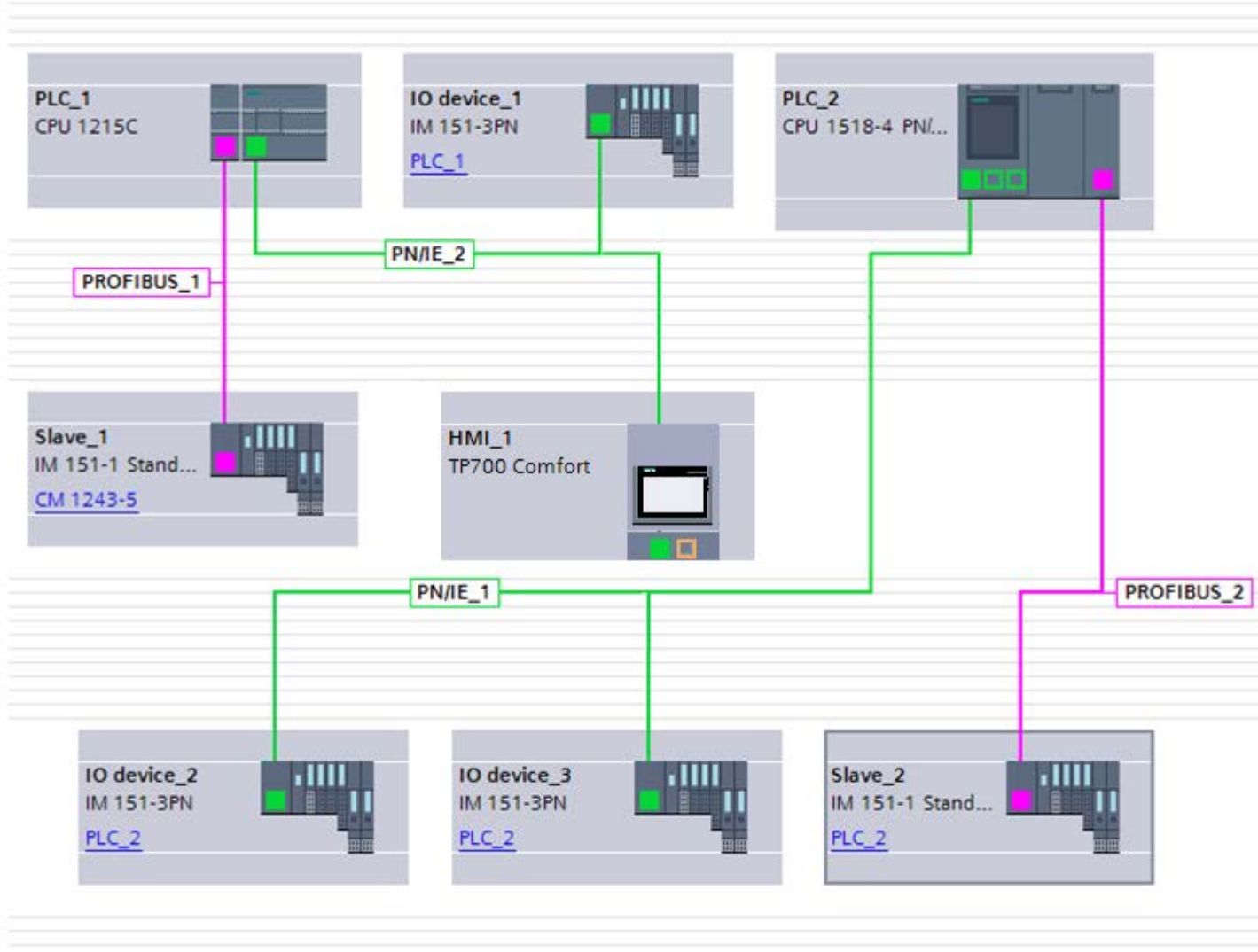
```
Uninitialized
BackupAllowed
ChangeModeAllowed
FirmwareUpdateAllowed
MemoryResetAllowed
PasswordAllowed
ProgramUpdateAllowed
ResetToFactoryAllowed
FormatMCallowed
RestoreAllowed
Failsafe
NotFailsafe
RemoteDataLogsAllowed
RemoteRecipesAllowed
Supported
ServiceDataAllowed
SetTimeAllowed
DiagBufferAllowed
```

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:

```
List<ICPU> cpus = devices.FilterOnlyCpus();
foreach (ICPU cpu in cpus)
{
  if (cpu.MemoryResetAllowed == FeatureSupport.MemoryResetAllowed)
  {
    cpu.MemoryReset(new EncryptedString(string.Empty));
  }
}
```

7.12 Network example

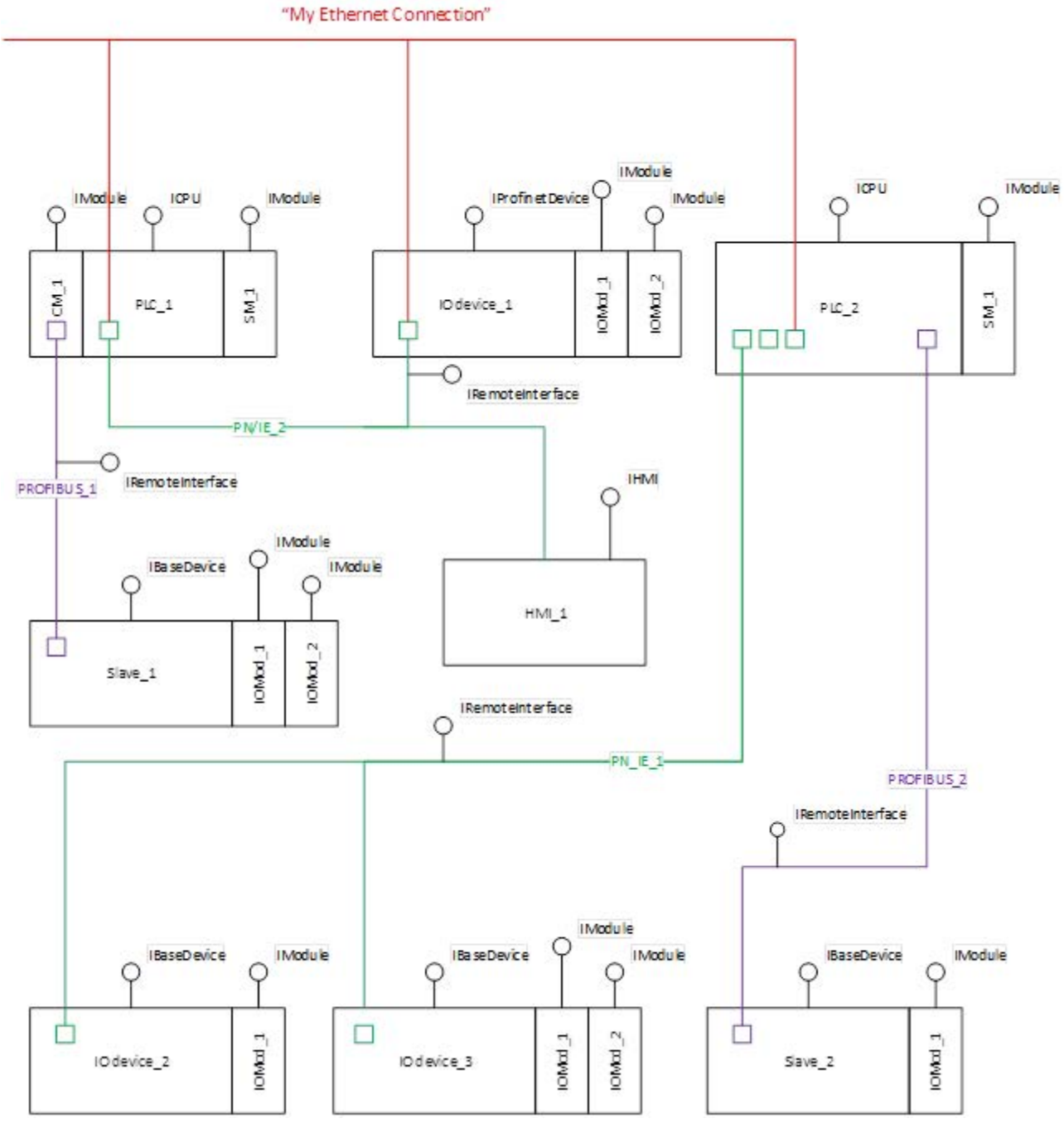
This example shows a TIA Portal network configuration and the API interfaces that represent the networked devices.



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

The device support tables show the correspondence between SIMATIC Automation Tool V3.0, SIMATIC device model, possible firmware versions, and supported tool operations.

If you connect an unrecognized SIMATIC device on your network, then there are two possibilities.

- The SIMATIC Automation Tool V3.0 recognizes the device article number, but the firmware version is newer than the latest supported firmware version. In this case, the device is loaded in the Device table, but the supported tool operations are restricted to those that were possible with the latest supported firmware version.
- The SIMATIC Automation Tool V3.0 does not recognize the article number. In this case, the device is not supported, no entry is made in the Device table, and no tool operations are possible.

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
- Flash device LED
- 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
- Flash device LED
- 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	✓	✓	✓
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
- Flash device LED
- 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
- Flash device LED
- 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
- Flash device LED
- 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	✓	✓	✓
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 S7-1500)

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.

S7-1500	V2.0	V2.1	Fail-Safe	
			V2.0	V2.1
Scan for devices	✓	✓	✓	✓
Flash LED	✓	✓	✓	✓
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	✓	✓		
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
- Flash device LED
- 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 S7-1500)

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.

	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	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Flash LED	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
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	✓	✓	✓	✓	✓					
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
- Flash device LED
- 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, Asi, 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	Motorstarter	✓
3RK1 308-0AC00-0CP0	DS 0.9 - 3A HF 3DI/LC	V1.0	Motorstarter	✓
3RK1 308-0AD00-0CP0	DS 2.8 - 9A HF 3DI/LC	V1.0	Motorstarter	✓
3RK1 308-0BB00-0CP0	RS 0.3- 1A HF 3DI/LC	V1.0	Motorstarter	✓
3RK1 308-0BC00-0CP0	RS 0.9- 3A HF 3DI/LC	V1.0	Motorstarter	✓
3RK1 308-0BD00-0CP0	RS 2.8- 9A 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 132-6BH00-7BA0	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	✓
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	✓

Article number	Module name	Firmware version	Module type	Firmware update
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 136-6PA00-0BC0	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		✓	✓	✓	✓	✓	✓	✓
Flash LED		✓	✓	✓	✓	✓	✓	✓
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		✓	✓	✓	✓	✓		
Firmware update				✓	✓	✓	✓	✓

8.3.2 S7-1200 I/O and CM 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
3RK7243-2AA30-0XB0	CM 1243-2	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	✓
6AG1 231-5PF32-2XB0	AI 8xRTD SIPLUS	V2.0	SM	✓

Article number	Module name	Firmware version	Module type	Firmware update
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	Communica-tionBoard	
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	
6ES7 223-1BL32-0XB0	DI 16/DQ 16x24VDC	V2.0	SM	✓

Article number	Module name	Firmware version	Module type	Firmware update
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	CM	✓
6ES7 241-1CH32-0XB0	CM 1241 (RS422/485)	V2.1	CM	✓

Article number	Module name	Firmware version	Module type	Firmware update
6ES7 241-1CH32-0XB0	CM 1241 (RS422/485)	V2.2	CM	✓
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	CP	✓
6GK7 243-7KX30-0XE0	CP 1243-7 LTE	V3.0	CP	✓
6GK7 243-7SX30-0XE0	CP 1243-7 LTE	V2.1	CP	✓
6GK7 243-7SX30-0XE0	CP 1243-7 LTE	V3.0	CP	✓
6GK7 243-8RX30-0XE0	CP 1243-8 IRC	V2.1	CP	✓
6GK7 243-8RX30-0XE0	CP 1243-8 IRC	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	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Flash LED	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
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	✓	✓	✓	✓	✓	✓	✓	✓						
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

8.4.2 S7-1500 I/O and module support

D I/O, A I/O, CP, and TM

A check mark (✓) means that the operation is supported.

Device firmware version and firmware update operation support

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
Basic	KTP 400 Basic
	KTP 700 Basic
	KTP 900 Basic
	KTP 1200 Basic

HMI panel firmware version and supported operations

Operation	HMI firmware version Greater than or equal to V13.0.0.0
Program update	✓ SIMATIC Automation Tool Program update for HMI devices can update HMI firmware, operating system, and run-time project data.
Backup	✓
Restore	✓

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	KP 400, KTP 400 Comfort
	KP 700, TP 700 Comfort
	KP 900, TP 900 Comfort
	KP 1200, TP 1200 Comfort
	KP 1500, TP 1500 Comfort
	TP 1900 Comfort
	TP 2200 Comfort

A check mark (✓) means that the operation is supported.

Operation	HMI firmware version Greater than or equal to V13.0.0.0
Program update	✓ SIMATIC Automation Tool Program update for HMI devices can update HMI firmware, operating system, and run-time project data.
Backup	✓
Restore	✓

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	KTP 700 Mobile
	KTP 900 Mobile

HMI panel firmware version and supported operations

Operation	HMI firmware version Greater than or equal to V13.0.0.0
Program update	✓ SIMATIC Automation Tool Program update for HMI devices can update HMI firmware, operating system, and run-time project data.
Backup	✓
Restore	✓

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	IM		✓
6EP3 436-8SB00-2AY0	PSU8600	V1.1	IM		✓
6EP3 437-8MB00-2CY0	PSU8600	V1.0, V1.1	IM		✓
6EP3 437-8SB00-2AY0	PSU8600	V1.1	IM		✓
6EP4 134-3AB00-2AY0	UPS1600 10A PN	V1.14, V1.22, V2.0, V2.1	SITOP	✓	✓
6EP4 136-3AB00-2AY0	UPS1600 20A PN	V1.14	SITOP	✓	✓
6EP4 136-3AB00-2AY0	UPS1600 20A PN	V1.22	SITOP	✓	✓
6EP4 136-3AB00-2AY0	UPS1600 20A PN	V2.0	SITOP	✓	✓
6EP4 136-3AB00-2AY0	UPS1600 20A PN	V2.1	SITOP	✓	✓
6EP4 137-3AB00-2AY0	UPS1600 40A PN	V2.0	SITOP	✓	✓
6EP4 137-3AB00-2AY0	UPS1600 40A PN	V2.1	SITOP	✓	✓
6EP4 293-8HB00-0XY0	BUF8600	V1.1	SITOP		✓
6EP4 295-8HB00-0XY0	BUF8600	V1.1	SITOP		✓
6EP4 297-8HB00-0XY0	BUF8600	V1.0	SITOP		✓
6EP4 297-8HB00-0XY0	BUF8600	V1.1	SITOP		✓
6EP4 297-8HB10-0XY0	BUF8600	V1.0	SITOP		✓
6EP4 297-8HB10-0XY0	BUF8600	V1.1	SITOP		✓
6EP4 436-8XB00-0CY0	CNX8600	V1.0	SITOP		✓
6EP4 436-8XB00-0CY0	CNX8600	V1.1	SITOP		✓
6EP4 437-8XB00-0CY0	CNX8600	V1.0	SITOP		✓
6EP4 437-8XB00-0CY0	CNX8600	V1.1	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

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W