SIEMENS

SIMATIC

SIMATIC Automation Tool SDK V4.0 SP3 User Guide

Programming Manual

Preface 1

Tutorial: Creating a custom application with the SIMATIC Automation Tool SDK 2

Updating a custom application made with a previous version of the SDK 3

SIMATIC Automation Tool API for .NET framework 4

V4.0 SP3, 09/2021

A5E45253030-AG

Legal information

Warning notice system

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

⚠ DANGER

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

⚠ WARNING

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

⚠ CAUTION

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

NOTICE

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

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

Qualified Personnel

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

Proper use of Siemens products

Note the following:

⚠ WARNING

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

Trademarks

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

Disclaimer of Liability

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

Siemens AG
Digital Industries
Postfach 48 48
90026 NÜRNBERG
GERMANY

A5E45253030-AG
Ⓟ 07/2021 Subject to change

Copyright © Siemens AG 2021.
All rights reserved

Table of contents

1 Preface 9

1.1 Security information 9

1.2 Overview of the SIMATIC Automation Tool SDK 9

1.3 Device catalog for the SDK 10

2 Tutorial: Creating a custom application with the SIMATIC Automation Tool SDK 11

2.1 Introduction to the example application 11

2.2 Example 1: Selecting a CPU from a network interface and changing the CPU operating mode 13
2.2.1 Creating a project that uses the API 13
2.2.2 Accepting a selected interface from the available network interfaces 15
2.2.3 Connecting to a CPU on the network by IP address 16
2.2.4 Displaying and trusting TLS certificates 19
2.2.5 Displaying the CPU information 21
2.2.6 Changing the operating mode of a CPU 22
2.2.7 Testing Example 1 23

2.3 Example 2: Reading and changing a CPU's IP address, subnet and gateway 24
2.3.1 Testing Example 2 26

2.4 Example 3: Reading and writing a CPU's PROFINET name 27
2.4.1 Testing Example 3 28

2.5 Example 4: Resetting a CPU to factory defaults 29
2.5.1 Testing Example 4 30

2.6 Example 5: Updating a CPU's firmware 31
2.6.1 Testing Example 5 34

2.7 Example 6: Managing configuration data protection 35
2.7.1 Testing Example 6 37

2.8 Example 7: Backing up a CPU and restoring a CPU from a backup file 38
2.8.1 Testing Example 7 41

2.9 Distributing an application to your end user 42

3 Updating a custom application made with a previous version of the SDK 44

4 SIMATIC Automation Tool API for .NET framework 45

4.1 Getting started with the API 45
4.1.1 Architectural overview 52
4.1.2 Referencing the API in a user interface application 54

4.2 API software and version compatibility 55

4.3 Requirement for S7 communications 56

4.4 Designing a user interface application for fail-safe devices and safety-relevant operations 56

4.4.1 API support for safety-relevant operations and fail-safe devices ... 56
4.4.2 User interface programming guidelines for safety-relevant operations ... 58
4.4.3 Color coding safety fields in your user interface ... 60
4.4.3.1 Coloring a CPU device icon ... 61
4.4.3.2 Coloring device data ... 62
4.4.3.3 Coloring a CPU password ... 63
4.4.3.4 Coloring a program folder ... 64
4.4.3.5 Coloring a program password ... 65
4.4.4 Hamming codes ... 66

4.5 Common support classes ... 66
4.5.1 EncryptedString class ... 66
4.5.2 Result class ... 67
4.5.3 DiagnosticsItem class ... 69
4.5.4 DataChangedEventArgs class ... 70
4.5.5 ProgressChangedEventArgs class ... 70
4.5.6 ExportProgressEventArgs class ... 71
4.5.7 FileProgressChangedEventArgs class ... 71
4.5.8 HealthCheckProgressEventArgs class ... 72

4.6 Common support interfaces ... 73
4.6.1 IRemoteFile interface ... 73
4.6.2 IRemoteFolder interface ... 73
4.6.3 IRemoteInterface interface ... 74
4.6.4 IHardware interface ... 74
4.6.5 IBaseDevice interface ... 75
4.6.6 IHardwareCollection interface ... 76
4.6.7 IModuleCollection interface ... 76
4.6.8 IScanErrorCollection class ... 76
4.6.9 IScanErrorEvent class ... 79
4.6.10 IHealthCheckErrorEvent interface ... 80
4.6.11 IDiagnosticBuffer interface ... 80
4.6.12 IResult interface ... 80
4.6.13 ICertificateStore interface ... 81
4.6.14 ICertificate interface ... 82

4.7 Network class ... 82
4.7.1 Network constructor ... 82
4.7.2 QueryNetworkInterfaceCards method ... 83
4.7.3 SetCurrentNetworkInterface method ... 83
4.7.4 CurrentNetworkInterface property ... 84
4.7.5 ScanNetworkDevices method ... 84
4.7.6 SetCommunicationsTimeout method ... 85
4.7.7 GetCommunicationsTimeout method ... 86
4.7.8 GetEmptyCollection method ... 87
4.7.9 ValidateNetworkInterface method ... 87

4.8 The HealthCheck class ... 87
4.8.1 HealthCheck constructor ... 87
4.8.2 ExportPCData method ... 88

4.9 IProfinetDeviceCollection class ... 90
4.9.1 Iterating items in the collection ... 90
4.9.1.1 Iterating items in the collection ... 90
4.9.1.2 FindDevicesBySerialNumber method ... 91

4.9.1.3 GetEnumerator method ... 91
4.9.1.4 SortByIP method ... 91
4.9.1.5 Count property ... 91
4.9.1.6 [ ] property ... 92
4.9.2 Filtering items in the collection ... 92
4.9.2.1 Collection items ... 92
4.9.2.2 FilterByDeviceFamily method ... 92
4.9.2.3 FilterOnlyCPUs method ... 93
4.9.3 Finding a specific device in the collection ... 94
4.9.3.1 FindDeviceByIP method ... 94
4.9.3.2 FindDeviceByMAC method ... 95
4.9.4 Serialization of the device collection ... 96
4.9.4.1 WriteToStream method ... 96
4.9.4.2 ReadFromStream method ... 96
4.9.4.3 ExportDeviceInformation method ... 97
4.9.4.4 ExportDeviceDiagnostics method ... 98
4.9.5 Manually adding items to the collection ... 100
4.9.5.1 InsertDeviceByIP method ... 100
4.9.5.2 InsertDeviceByMAC method ... 101
4.9.6 Copying data from a collection ... 102
4.9.6.1 CopyUserData method ... 102
4.9.7 Removing devices from the collection ... 103
4.9.7.1 Clear method ... 103
4.9.7.2 Remove method ... 103

4.10 IProfinetDevice interface ... 103
4.10.1 IProfinetDevice properties ... 103
4.10.2 IProfinetDevice methods ... 108
4.10.2.1 FirmwareUpdate method ... 108
4.10.2.2 FirmwareActivate method ... 112
4.10.2.3 Identify method ... 115
4.10.2.4 IsFirmwareUpdateAllowed method ... 116
4.10.2.5 IsFirmwareUpdate2StepAllowed method ... 116
4.10.2.6 IsIPAddressOnNetwork method ... 116
4.10.2.7 IsPROFINETNameOnNetwork method ... 117
4.10.2.8 IsUploadServiceDataAllowed method ... 118
4.10.2.9 RefreshStatus method ... 119
4.10.2.10 ResetCommunicationParameters method ... 119
4.10.2.11 SetIP method ... 120
4.10.2.12 SetProfinetName method ... 122
4.10.2.13 UploadServiceData method ... 123
4.10.2.14 ValidateIPAddressSubnet method ... 124
4.10.2.15 ValidatePROFINETName method ... 125
4.10.3 IProfinetDevice events ... 126
4.10.3.1 DataChanged event ... 126
4.10.3.2 ProgressChanged event ... 127

4.11 IModuleCollection class and module properties ... 128
4.11.1 Modules property and IModuleCollection class ... 128
4.11.2 IModule interface ... 129

4.12 ICPU interface ... 130
4.12.1 Identifying CPU devices in an IProfinetDeviceCollection ... 130
4.12.2 ICPU properties ... 131

4.12.3 ICPU flags ... 132
4.12.3.1 Program Update flags ... 132
4.12.3.2 Restore flags ... 133
4.12.3.3 ICPU supported and allowed flags ... 134
4.12.4 ICPU methods ... 135
4.12.4.1 Backup method (ICPU interface) ... 135
4.12.4.2 CreateConfigurationDataProtectionCard method ... 136
4.12.4.3 CreateFirmwareUpdateCard method ... 137
4.12.4.4 CreateProgramUpdateCard method ... 138
4.12.4.5 DeleteConfigurationDataProtectionPassword method ... 139
4.12.4.6 DeleteDataLog method ... 140
4.12.4.7 DeleteRecipe method ... 142
4.12.4.8 DetermineConfirmationMessage ... 143
4.12.4.9 DownloadRecipe method ... 147
4.12.4.10 FormatMemoryCard method ... 148
4.12.4.11 GetCurrentDateTime method ... 150
4.12.4.12 GetDiagnosticsBuffer method ... 151
4.12.4.13 MemoryReset method ... 152
4.12.4.14 ProgramUpdate method ... 153
4.12.4.15 ResetToFactoryDefaults method ... 156
4.12.4.16 Restore method (ICPU interface) ... 158
4.12.4.17 SetBackupFile method ... 161
4.12.4.18 SetBackupFilePassword method ... 161
4.12.4.19 SetConfigurationDataProtectionPassword method ... 162
4.12.4.20 SetCurrentDateTime method ... 163
4.12.4.21 SetOperatingState method ... 164
4.12.4.22 SetPassword method ... 164
4.12.4.23 SetProgramFolder method ... 166
4.12.4.24 SetProgramPassword method ... 167
4.12.4.25 SetTrustCertificate method ... 167
4.12.4.26 UploadDataLog method ... 168
4.12.4.27 UploadRecipe method ... 170
4.12.4.28 ValidateProgramFolder method ... 171
4.12.5 RemoteInterfaces property ... 172
4.12.5.1 Decentralized I/O modules ... 172
4.12.5.2 IRemoteInterface properties ... 173

4.13 The ICPUClassic interface ... 176
4.13.1 Identifying classic CPU devices in an IProfinetDeviceCollection ... 176
4.13.2 GetDiagnosticsBuffer method ... 177

4.14 IHMI interface ... 178
4.14.1 IHMI interface ... 178
4.14.2 IHMI properties and flags ... 179
4.14.2.1 IHMI properties ... 179
4.14.2.2 Program update flags ... 179
4.14.2.3 Restore flags ... 179
4.14.2.4 Feature flags ... 180
4.14.3 IHMI methods ... 180
4.14.3.1 Backup method (IHMI interface) ... 180
4.14.3.2 FirmwareUpdate method ... 181
4.14.3.3 ProgramUpdate method (IHMI interface) ... 182
4.14.3.4 Restore method (IHMI interface) ... 183

4.14.3.5 SetBackupFile method 184
4.14.3.6 SetProgramFolder method 185
4.14.3.7 SetTransferChannel method 185

4.15 IScalance interface 186
4.15.1 IScalance interface 186
4.15.2 IScalance properties 187
4.15.3 IScalance methods 187
4.15.3.1 SetProfile method 187
4.15.3.2 FirmwareUpdate method 188
4.15.3.3 FirmwareActivate method 189

4.16 ISNMPProfile interface 189
4.16.1 ISNMPProfile properties 189
4.16.2 ISNMPProfile methods 190
4.16.2.1 Validate method 190
4.16.3 SNMPProfile class 190
4.16.3.1 SNMPProfile class properties 190
4.16.3.2 SNMPProfile class methods 191

4.17 IScalance and ISNMP firmware update code examples 196
4.17.1 Example: SNMP Version 1 configuration 196
4.17.2 Example: SNMP Version 2 configuration 198
4.17.3 Example: SNMP Version 3 configuration 199

4.18 Exceptions 202
4.18.1 CriticalInternalErrorException 202

4.19 API enumerations 203
4.19.1 BackupType 203
4.19.2 CardSortDirection 203
4.19.3 CardSortItem 204
4.19.4 CertificateStatusError 204
4.19.5 ConfigurationDataProtection 204
4.19.6 ConfirmationType 204
4.19.7 DataChangedType 205
4.19.8 DeviceFamily 205
4.19.9 ErrorCode 206
4.19.10 FailsafeOperation 212
4.19.11 FirmwareUpdateType 212
4.19.12 HMITransferChannel 212
4.19.13 Language 212
4.19.14 OperatingState 213
4.19.15 OperatingStateREQ 213
4.19.16 ProgressAction 213
4.19.17 ProtectionLevel 214
4.19.18 RedundancyRole 214
4.19.19 RemoteFolderType 214
4.19.20 RemoteInterfaceType 214
4.19.21 ResetOptions 215
4.19.22 ScanErrorType 215
4.19.23 SDCardType 215
4.19.24 SNMPAuthAlgorithm 215
4.19.25 SNMPPrivAlgorithm 216
4.19.26 SNMPSecurityLevel 216

4.19.27 SNMPVersion 216
4.19.28 TimeFormat 216
4.20 CPU password access levels 217
4.21 Network example 218
4.22 Reference information 220
4.22.1 About devices behind routers 220
4.22.2 About devices connected to CMs or CPs 221
Index 223

Preface

1

1.1 Security information

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

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

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

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

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

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

1.2 Overview of the SIMATIC Automation Tool SDK

The SIMATIC Automation Tool Software Development Kit (SDK) enables you to create an application using the SIMATIC Automation Tool Application Programming Interface (API). You can distribute your application, including the API software, to third parties. Your custom application can enable your users to perform a variety of device automation tasks. Your users do not need a license from Siemens to use your custom application. Siemens provides a Windows installer package that you can use to create a setup for your custom application. The installer includes all necessary components for distributing your software and for communication with S7 devices.

This user guide provides the following:

- Tutorial on creating a custom application using an example .NET solution (Page 11)

  The SIMATIC Automation Tool SDK provides a complete example program in C# that performs a variety of device operations. The example program solution includes a simple user interface with API method calls to operate the user controls. The tutorial section of the user guide shows you how to set up an application in the .NET framework that uses the API. It then shows you step by step how to program the API method calls to perform

the device operations in the example program. The complete example program solution is included in your installation folder.

- Complete API reference (Page 45)

  While the example program illustrates many of the available API methods, it uses only a subset of the full set of capabilities that the API provides. This reference section includes all of the classes, interfaces, methods, properties, and enumerated types that you have available to use. In many cases, the API reference includes code examples for specific programming tasks.

1.3 Device catalog for the SDK

The SIMATIC Automation Tool SDK provides a device catalog. The device catalog is a Microsoft Excel file that lists the devices that an SDK application can access. For each device, you can see the supported firmware versions and supported operations.

The SDK File Extractor installs the device catalog in the SAT SDK V4.0 SP3\Bin\Catalog folder in your Documents folder. You run the SDK File Extractor from the Siemens Automation group in the Windows Start menu.

Windows 10

From File Explorer, the Documents folder is under Quick Access.

Windows 7

From Windows Explorer, the Documents folder is under the Libraries folder.

Tutorial: Creating a custom application with the SIMATIC Automation Tool SDK

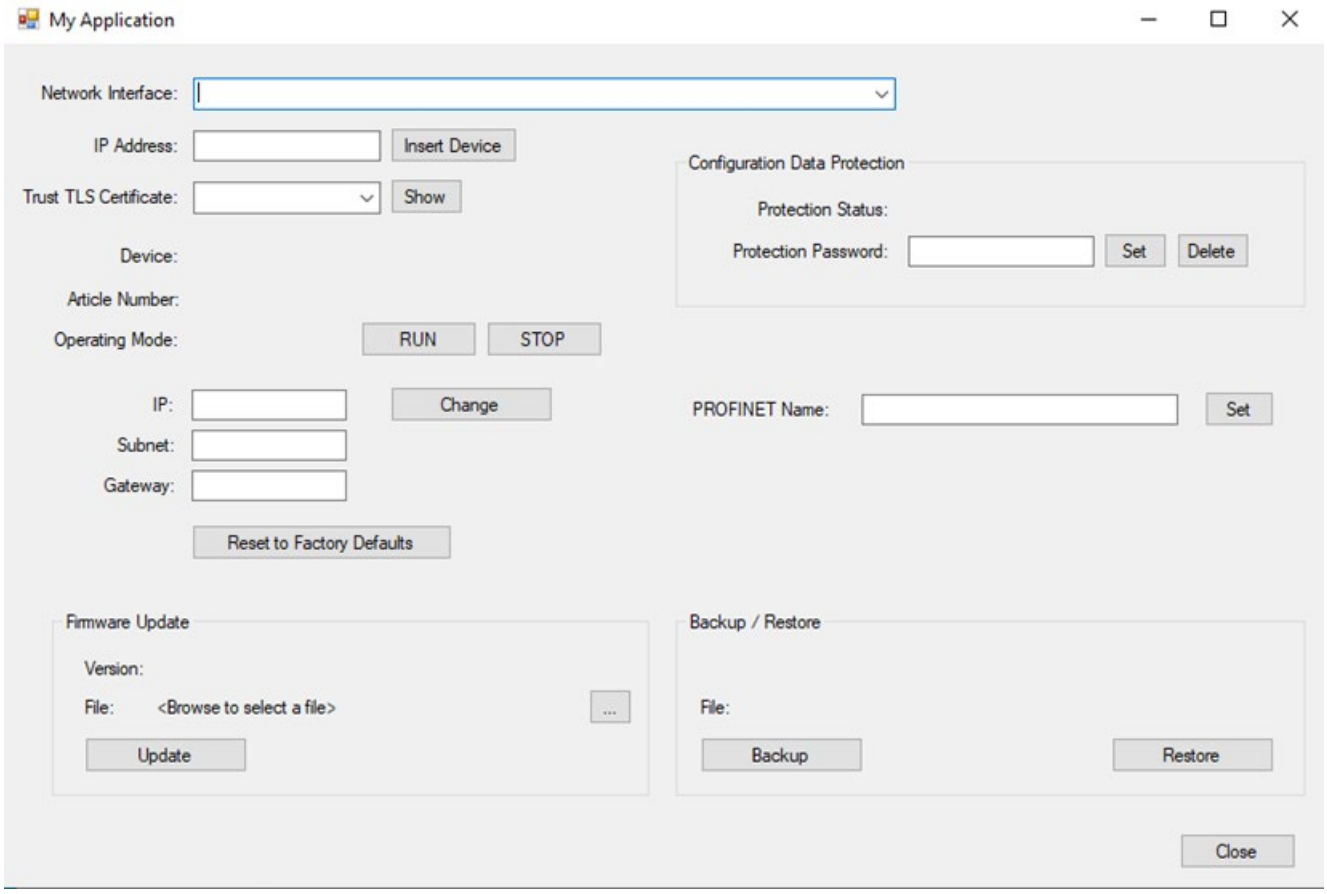
2

2.1 Introduction to the example application

Your goal in this tutorial is to create a Windows application that uses some of the interfaces and methods available in the SIMATIC Automation Tool API. As you proceed through the tutorial, you will progressively add controls and logic to your solution. Your solution will utilize several typical API interfaces and their associated methods. The application enables your user to work with a single CPU on your device network. You will develop the software that enables a user to perform the following tasks:

- Select a network interface for CPU communication
- Insert a CPU by its IP address
- Display identifying information for the CPU
- Change the operating mode of the CPU
- Change the IP address, subnet, and gateway for the CPU
- Set the PROFINET name of the CPU
- Reset the CPU to factory defaults
- Update the firmware of the CPU
- Back up the CPU to a file
- Restore a backup file to the CPU

The user interface for your Windows application will be similar to the following example:



Siemens recommends that you perform the programming tasks without looking at the solution that the SDK provides in your installation folder. Instead, learn how to use the API under the guidance of this tutorial. After you develop a working application, you can then look at the provided solution for comparison and "check your work".

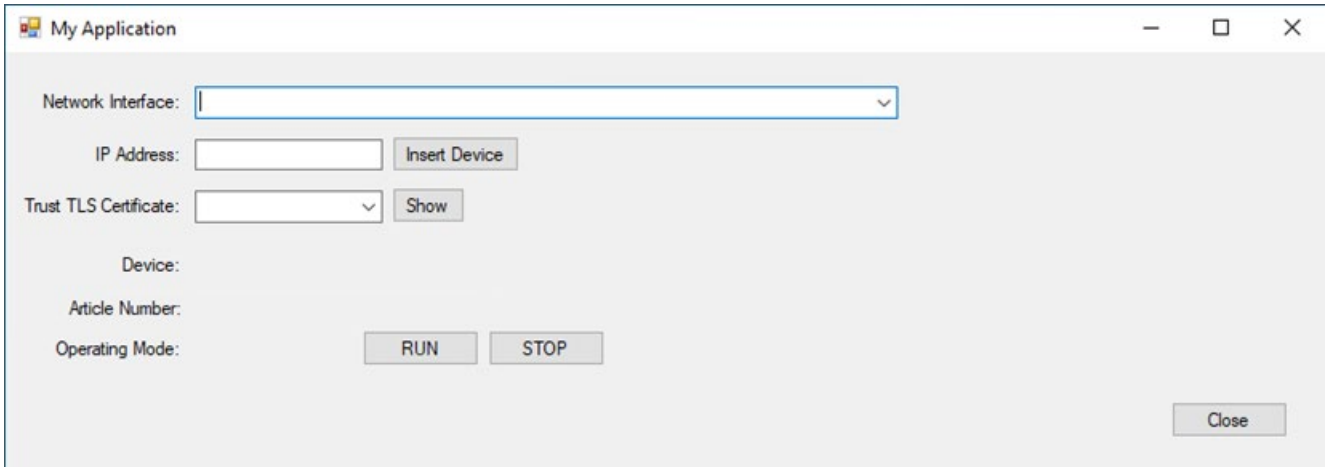
To minimize the amount of code to learn, the provided solution implements minimal result checking of each API operation. In actual practice, use good defensive programming techniques and check the results of operations frequently. Program the application to proceed when operations are successful and to handle errors appropriately. The example programs and API method calls use the Result object to return status of operations. Check the value of the Result object after API method calls in your program logic. Also, add visible feedback for your user such as a wait cursor or progress indicator for long operations.

Note that the user interface of the sample application is in English only. Also note that the sample application exercises only a subset of the full set of the API functionality (Page 45). After you become familiar with programming using the API, you can use any of the API interfaces, classes, and methods that you need to create other applications.

For simplicity, the example program performs no password handling for password-protected CPUs. In actual practice, use the SetPassword method (Page 164) as needed.

2.2 Example 1: Selecting a CPU from a network interface and changing the CPU operating mode

This chapter guides you through programming the controls for a sample application as shown below:



As you work through this chapter, you will use the API to provide the following features:

- A selector for the user to choose a network interface (Page 15)
- A means of inserting a CPU by its IP address (Page 16)
- A means of displaying and trusting the TLS Certificate (Page 19)
- Device information for the entered CPU (Page 21)
- Buttons for changing the operating mode of the CPU (Page 22)

Prerequisites

To develop your application, you must meet the following prerequisites. A user who runs your application must also meet these prerequisites:

- You have a CPU connected to your programming device through a network interface.
- You know the IP address of the CPU.
- You can ping the CPU on the network and verify that it responds.

2.2.1 Creating a project that uses the API

Use Microsoft Visual Studio to develop applications using the SIMATIC Automation Tool API (Page 45). Your application and the SIMATIC Automation Tool API .dll files must reside in the same folder.

Creating a project that uses the API

Siemens developed the API with Microsoft Visual Studio 2017 using the .NET Framework 4.8. Use these versions to create 64-bit (x64) applications with the API. All code examples in this document are in the C# programming language.

To create a project that can work with the SIMATIC Automation Tool API, follow these steps:

1. In Visual Studio, create a new project that is a Visual C# Windows Forms Application. Name it "My Application".
2. From the Solutions Platforms drop-down list select "Configuration Manager".
3. In the Configuration Manager, click the Platform drop-down and create a new platform.
4. From the New Project Platform dialog, select "x64" for the new platform. Click OK and close the Configuration Manager.
5. In the project properties, set the target framework to .NET Framework 4.8.

Note

The SIMATIC Automation Tool API uses the .NET Framework 4.8. The setup installs this framework if it is not present on your programming device.

Setting up the project to use the API

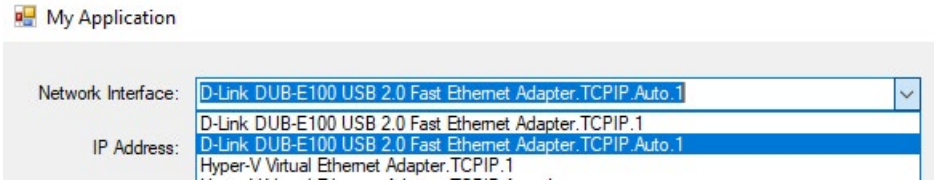
To set up your project to use the API, follow these steps:

1. Run the SAT SDK File Extractor to copy the API files and folders to the default location. You access the SAT SDK File Extractor from the Siemens Automation group from the Start menu.
2. Open the location where you extracted the files and copy the contents of the "Bin" folder to your \bin\x64\Debug folder.
3. Add a reference in your project to `bin\x64\Debug\SIMATICAutomationToolAPI.dll`.
4. In the file `Form1.cs`, add the following statement:
   ```
   using Siemens.Automation.AutomationTool.API;
   ```
   This statement enables the application to use the API dll. You will create objects in your program from the API classes and interfaces.
5. Add a Close button to your form so your user can close the application.
6. Build your solution and verify that it builds successfully.

Your application is now ready to develop using the API.

2.2.2 Accepting a selected interface from the available network interfaces

Your application must determine the network interfaces that are available from your programming device. Using the available interfaces, allow the user to choose one for communication.



Providing a means for the user to select a network interface

To read and display the network interfaces, as well as enable the user to select an interface, follow these steps:

1. Add a ComboBox to your form with a text label such as "Network Interface".
2. Name the ComboBox "SelectNetworkInterface".
3. Enter a global variable for the network to the `public partial class Form1` class in `Form1.cs`:

   ```
   namespace MyApplication
   {
      public partial class Form1 : Form
      {
         public Network Net = new Network();
   ```

   Note that "Network (Page 82)" is a class that the API provides.
4. Set up a region for global helper methods within the `Form1` class:

   ```
   #region helper methods

   #endregion
   ```
5. Add a global empty method in the `#region` for helper methods:

   ```
   private void ClearCPUStatus()
   {
   }
   ```

   You will add code to this method later. Whenever the user selects a network interface, the application will clear any existing CPU status information.

6. In the constructor for `Form1`, add the following code to read the available network interfaces and load them in the ComboBox:

```
List<string> NetworkInterfaces;

Result res = Net.QueryNetworkInterfaceCards(out NetworkInterfaces);

foreach (string nic in NetworkInterfaces)
   SelectNetworkInterface.Items.Add(nic);
ClearCPUStatus();
```

7. Double-click the ComboBox control on the form to add a method that enables the user to select a network interface. Insert the code as shown below in the method:

```
private void SelectNetworkInterface_SelectedIndexChanged(object
sender, EventArgs e)
{
   Result res =
Net.SetCurrentNetworkInterface((string)SelectNetworkInterface.Selec
tedItem);
}
```

At this point, compile your project and verify that it compiles successfully. Test your drop-down box to be sure that it displays the available network interfaces. Verify that you can select each network interface. Add additional error checking as necessary.

When you are satisfied with your work, proceed to Connecting to a CPU on the network by IP address (Page 16).

2.2.3 Connecting to a CPU on the network by IP address

If the network interface connects to a network with S7-1200 or S7-1500 CPUs, the next step is to enable the user to connect to a CPU on the network. This CPU is then the connected device for device commands. Your application, if you implement it according to the provided sample program, must be able to perform the following tasks:

1. Accept and validate the entry of an IP address.
2. Insert a device for the IP address into the device collection. The device collection will be a collection of a single device in this example.
3. Read data from the network device that corresponds to the entered address.
4. Verify that the device at the IP address is an S7-1200 or S7-1500 CPU.
5. Display the device information for the CPU.



This chapter guides you through the form design and programming for each task.

Note

This task sequence assumes that each preceding task is successful. In your application, perform verification and error checking. Only proceed from task to task upon success.

Providing a text box to enter an IP address

The user enters the IP address of the CPU in a text box. To provide this text box for your user, follow these steps:

1. Place a TextBox control on your form with a corresponding label, for example "IP Address:".
2. Name the TextBox "InsertDeviceIPAddress".
3. Ensure that the text box is wide enough for an IPv4 address.
4. Add the following global declarations to the `public partial class Form1` class in `Form1.cs`. Add these right after the public Network declaration that you added in Accepting a selected interface from the available network interfaces (Page 15):

   ```
   public IProfinetDeviceCollection Devices;
   public ICPU CurrentCPU;
   ```

   Note that "IProfinetDeviceCollection (Page 90)" is a class that the API provides and that "ICPU (Page 130)" is an interface. This tutorial application works with the `Devices` collection and the `CurrentCPU` object throughout the examples.

5. In the helper method region, add an IP address parser method named `ParseIP`. `ParseIP` converts the text-based IP address to an unsigned integer:

   ```
   private uint ParseIP(string StrNetParm)
   {
      try
      {
         System.Net.IPAddress ip =
   System.Net.IPAddress.Parse(StrNetParm);
         byte[] bytes = ip.GetAddressBytes();
         Array.Reverse(bytes);
         return BitConverter.ToUInt32(bytes, 0);
      }
      catch (Exception e)
      {
         return 0xffffffff;
      }
   }
   ```

Implementing a button to process the entered IP address

You need a button for the user to click after entering an IP address. The button attempts to connect to a CPU on the network by the IP address. To implement this button, follow these steps:

1. Drop a button on your form and name it "`InsertDevice`".
2. Program a click event for the button:

   ```
   public void InsertDevice_Click(object sender, EventArgs e)
   ```

3. In the `#region` for the helper methods (Page 15), add a global empty method:

   ```
   private void UpdateCPUStatus()
   {
   }
   ```

   You will add code to this method later.

4. Within the `InsertDevice_Click` method, add code to clear any existing device data and to clear the CPU status:

   ```
   CurrentCPU = null;
   ClearCPUStatus();
   ```

5. Call your `ParseIP` method to convert the entered IP address from text to a 32-bit big-endian unsigned integer:

   ```
   uint ipAddress = ParseIP(InsertDeviceIPAddress.Text);
   ```

Throughout your application, add error checking as necessary. For simplicity, this tutorial omits typical error-handling code.

Using the IP address to connect to a CPU

Now that you have a labeled text box and corresponding button on your form, program the code to find the CPU on the network. After finding the CPU, your application can then read the device information from the CPU.

The API provides a device collection class, `IProfinetDeviceCollection` (Page 90). You will insert your CPU into a collection (`Devices`) so that you can perform subsequent device operations on the CPU. The `ScanNetworkDevices` (Page 84) method of the `Network` (Page 82) class is how you enter the CPU into the collection.

To program your `InsertDevice_Click` method to insert the CPU into your device collection, follow these steps:

1. Create an empty device collection:

   ```
   Devices = Network.GetEmptyCollection();

   Devices.Clear();
   ```

2. Insert your device into the collection using the `InsertDeviceByIP` method:

   ```
   IProfinetDevice InsertedDevice;

   IScanErrorCollection InsertErrorCollection =
   Devices.InsertDeviceByIP(ipAddress, out

   InsertedDevice);
   ```

3. Verify that the device is an S7-1200 or an S7-1500 CPU:

```
Result res = Devices[0].RefreshStatus();
if (Devices[0].Family != DeviceFamily.CPU1200 && Devices[0].Family
!= DeviceFamily.CPU1500)
   {
      MessageBox.Show("ERROR: Inserted device is not a CPU",
"Insert Device", MessageBoxButtons.OK);
      return;
   }
```

4. Cast the device as a CPU (`ICPU` (Page 130) class) and save a reference. The application performs all device operations on the CPU through `CurrentCPU`:

```
CurrentCPU = Devices[0] as ICPU;
```

5. Update the status of the device:

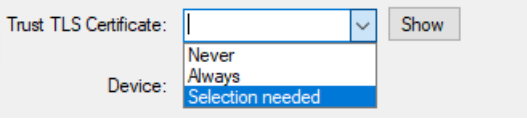
```
UpdateCPUStatus();
```

6. As before, add appropriate error checking of returned results of operations.
7. Build your solution and verify that your code compiles.

After you successfully insert the S7-1200 or S7-1500 CPU in the device collection, your next task is Displaying the CPU information (Page 21).

2.2.4 Displaying and trusting TLS certificates

If your application will support the CPUs that enable secure PG/PC communication using certificates over TLS V1.3, you must create a mechanism so that the user can display and set the trust level for the TLS certificate. The ICPU object provides a method to display the certificate: `ICPU.CertificateStore.ShowDialog()` and a method to set the certificate trust: `ICPU.SetTrustCertificateStore()`.



Selecting certificate trust

Provide a box from which the user can select the certificate trust level. This box allows the customer to select "Never", "Always", or "Selection needed". The latter is the default. This will be done using a Combo box. To implement, follow these steps:

1. Drop a Combo box on your form and name it "TrustTLSCertificate".
2. Add three items to the Items collection: "Never", "Always", and "Selection needed".
3. Double-click the Combo box and add the following code to the SelectedIndexChanged method:

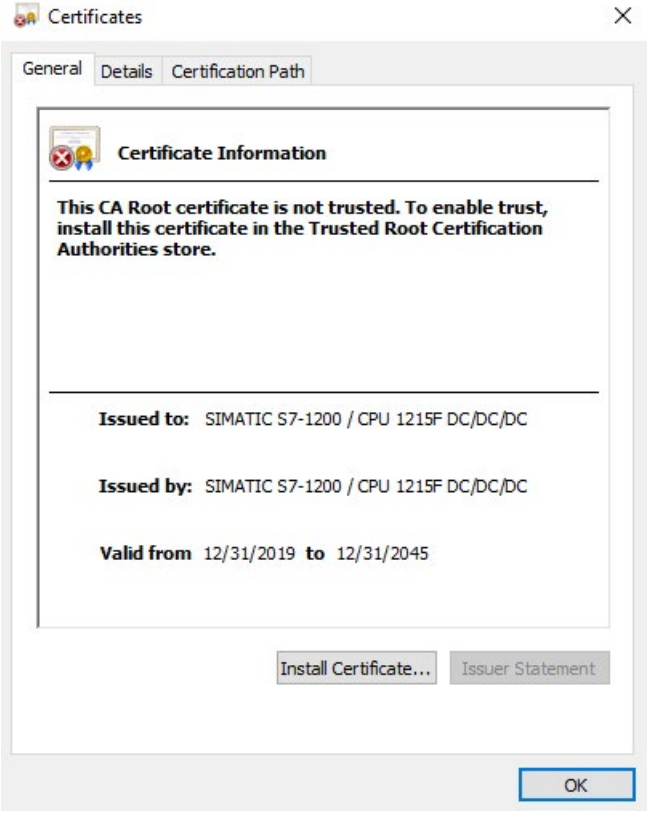
```
private void TrustTLSCertificate_SelectedIndexChanged(object sender,
EventArgs e)

{
```

```
if ((string)TrustTLSCertificate.SelectedItem == "Always")
   CurrentCPU.SetTrustCertificateStore(TrustCertificateType.Always);
else if ((string)TrustTLSCertificate.SelectedItem == "Never")
   CurrentCPU.SetTrustCertificateStore(TrustCertificateType.Never);
else
   CurrentCPU.SetTrustCertificateStore(TrustCertificateType.SelectionNeeded);
}
```

Displaying the certificate

The certificate below will be displayed in a separate window. You need a button in your program interface to launch this operation.



To implement this functionality, take the following steps:

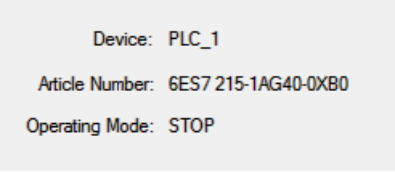
1. Drop a button on your form and give it a descriptive label such as "Show".
2. Name the new button "ShowCertificate".
3. Double-click on the button and add the following code to the Click method:

```
private void ShowCertificate Click(object sender, EventArgs e)
{
```

```
 CurrentCPU.CertificateStore.ShowDialog();
}
```

2.2.5 Displaying the CPU information

At this point in your application, you have implemented controls for connecting to a CPU on a network. Now that your application has successfully established this connection, add controls and logic to display information about the device:



Adding the labels for the CPU device information

Add label controls to your form with descriptions for each label:

Text on form	Label control name
Device:	DeviceName
Article number:	ArticleNumber
Operating mode:	OperatingMode

Set AutoSize for each label control to FALSE and set the width large enough to accommodate maximum expected text for each field.

Programming the CPU status methods

Previously, you created two empty help methods: `ClearCPUStatus` (Page 15) and `UpdateCPUStatus` (Page 16). Your next task is to add code to these methods for the CPU device information:

1. Insert the following code into `ClearCPUStatus()`:

   ```
   DeviceName.Text = string.Empty;
   ArticleNumber.Text = string.Empty;
   OperatingMode.Text = string.Empty;
   ```

   Note that each field initially displays "..." when the user has not yet connected to a CPU.

2. In `UpdateCPUStatus()`, insert the following code:

   ```
   DeviceName.Text = CurrentCPU.Name;
   ArticleNumber.Text = CurrentCPU.ArticleNumber;
   OperatingMode.Text = CurrentCPU.OperatingMode ==
   OperatingState.Run ? "RUN" : "STOP";
   ```

Recall that `CurrentCPU` contains the device information that you read from the device in Connecting to a CPU on the network by IP address (Page 16). The `InsertDevice_Click` method calls the `ClearCPUStatus` method and the `UpdateCPUStatus` method when the user clicks the Insert Device button.

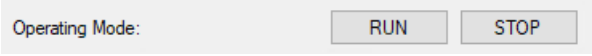
3. Compile and test your program before proceeding to the next step: Changing the operating mode of a CPU (Page 22)

2.2.6 Changing the operating mode of a CPU

By this point, you have completed these tasks:

- Accepting a selected interface from the available network interfaces (Page 15)
- Connecting to a CPU on the network by IP address (Page 16)
- Displaying the CPU information (Page 21)

The next task in this example program is to add button controls and logic to change the operating mode:



Note

The operations in this example require the following:

- The CPU is not password-protected.
- The CPU has a user program in it. (It is not reset to factory defaults.)

Programming the RUN button

To program the RUN button, follow these steps:

1. Add a button to the form with the name "`RunButton`".
2. Add a button click method and populate it with the following code:

```
CurrentCPU.Selected = true;
Result res = CurrentCPU.SetOperatingState(OperatingStateREQ.Run);
UpdateCPUStatus();
```

Programming the STOP button

To program the STOP button, follow these steps:

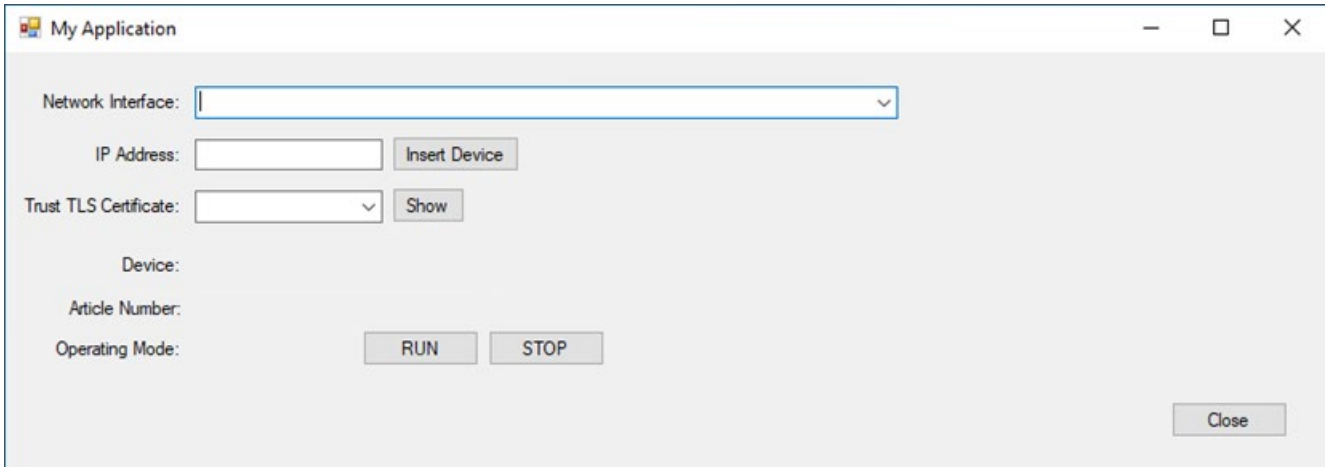
1. Add another button to the form with the name "`StopButton`".
2. Add a button click method and populate it with the following code:

```
CurrentCPU.Selected = true;
Result res = CurrentCPU.SetOperatingState(OperatingStateREQ.Stop);
UpdateCPUStatus();
```

After programming both button methods, check the results of the operations.

2.2.7 Testing Example 1

Compile and test your application. If you followed the steps in this example, you have created an application like the following:



Verify that the following user actions work correctly:

- Clicking the drop-down button for the network interface displays the available network interfaces for your programming device.
- Selecting a network interface fills in the Network Interface field.
- Inserting an IP address and clicking "Insert Device" fills in the device information fields. In this example, the IP address must correspond to an S7-1200 or S7-1500 CPU on the selected network interface.
- Clicking the "Show" button displays the PLC certificate.
- Clicking the drop-down button for Trust TLS Certificate displays the three possible trust levels.
- Selecting a trust level fills in the Trust TLS Certificate field.
- Clicking the RUN button causes the CPU to go to or remain in RUN mode.
- Clicking the STOP button causes the CPU to go to or remain in STOP mode.

Also test various error cases, for example:

- Attempting to insert a device before selecting a network interface
- Attempting to insert a device that is not an S7-1200 or S7-1500 CPU
- Using invalid text in the IP address field
- Choosing to never trust the TLS certificate results in an error when you click the RUN or STOP button.

You can perform other testing as you choose. Refine your program as necessary to provide robust error handling. When you are satisfied with your results on Example 1, you can

proceed to Example 2: Reading and changing a CPU's IP address, subnet and gateway (Page 24).

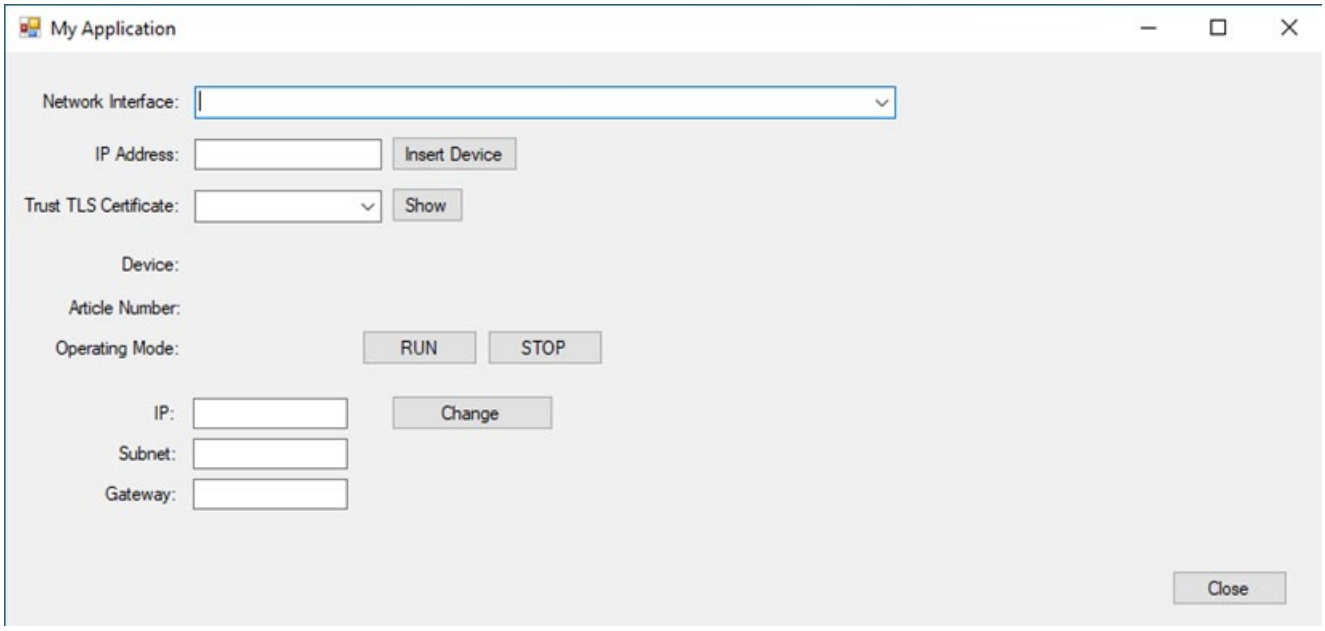
2.3 Example 2: Reading and changing a CPU's IP address, subnet and gateway

Example 2 extends the application that you developed in Example 1 (Page 13). Example 2 adds controls and logic to display the following IP protocol information for the connected CPU:

- IP address (IP)
- Subnet
- Gateway

For each of these fields, Example 2 also allows your user to enter new values and to change the values in the CPU.

Three text boxes on your form display the IP protocol values from the connected CPU. After your user has successfully connected to a CPU device (Page 16), the three text boxes show the IP protocol information for the CPU. These three text boxes are also editable fields. Your user can enter new values for any of the IP protocol values listed above.



Adding the text boxes for IP protocol values

To add text boxes and program the logic for the IP protocol fields, follow these steps:

1. Add three TextBoxes to your form with the following names and label descriptions:

Text on form	TextBox name
IP:	SuiteIp
Subnet:	SuiteSn
Gateway:	SuiteGw

2. Add code to the existing `ClearCPUStatus` (Page 15) helper method to clear the IP protocol text fields:

```
SuiteIp.Text = string.Empty;
SuiteSn.Text = string.Empty;
SuiteGw.Text = string.Empty;
```

3. Add code to the existing `UpdateCPUStatus` (Page 16) helper method to load the IP protocol text fields with the values from the currently connected CPU:

```
if (CurrentCPU.IPString.StartsWith("X"))
   SuiteIp.Text = CurrentCPU.IPString.Substring(4);
else
   SuiteIp.Text = CurrentCPU.IPString;

SuiteSn.Text = CurrentCPU.SubnetMaskString;
SuiteGw.Text = CurrentCPU.DefaultGatewayString;
```

Adding a button to change the IP protocol values

To add and program the button for changing the IP protocol fields, follow these steps:

1. Add a button to your form named "`ChangeIpSuite`".
2. Add the text "Change" for the button text.
3. Add a Button Click event to the Change button and insert the following code:

```
uint newIp = ParseIP(SuiteIp.Text);
uint newSn = ParseIP(SuiteSn.Text);
uint newGw = ParseIP(SuiteGw.Text);

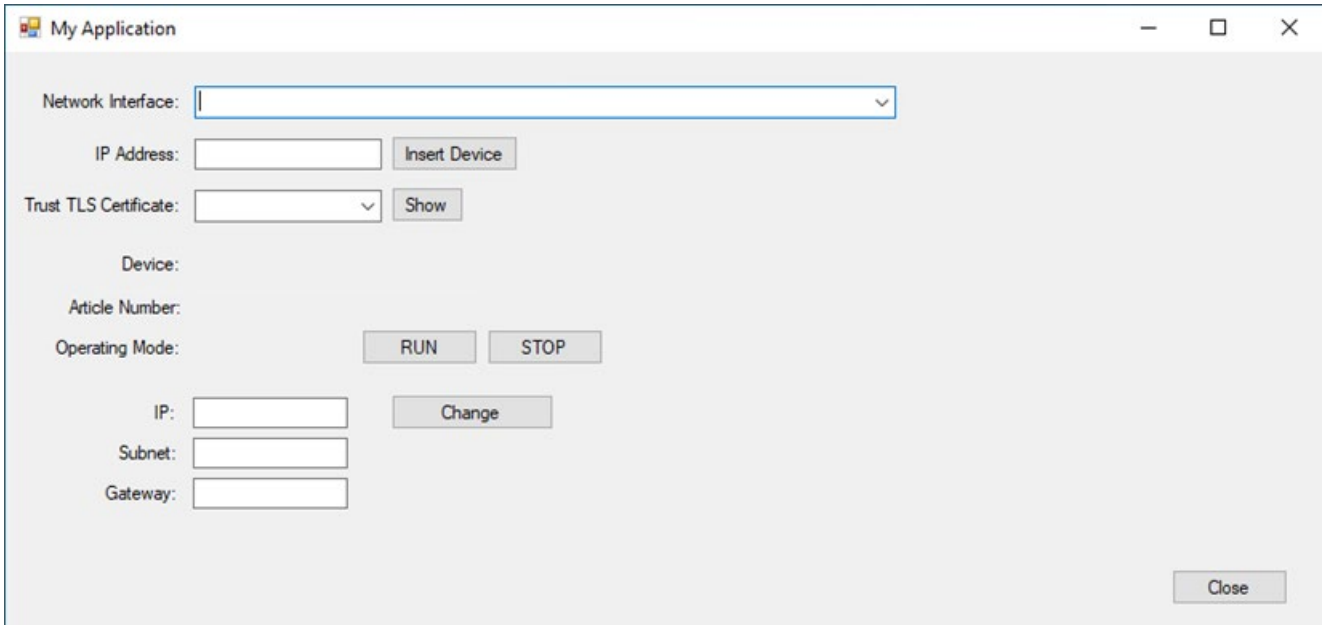
if (newIp == 0xffffffff || newSn == 0xffffffff || newGw ==
0xffffffff)
{
   MessageBox.Show("ERROR: Invalid IP, Subnet, or Gateway address
specified", "Change IP Suite", MessageBoxButtons.OK);
   return;
}

Result res = CurrentCPU.SetIP(newIp, newSn, newGw);
```

The example program does not provide logic to check that the IP address, Subnet, and Gateway are compatible with each other. You can add such logic to make your solution more robust.

2.3.1 Testing Example 2

Compile and test your application. If you followed the steps in this example, you have created an application like the following:



Verify that the following user actions work correctly:

- Entering an IP address of a CPU on the network and clicking "Insert Device" fills in the IP protocol information fields. In this example, the IP address must correspond to an S7-1200 or S7-1500 CPU on the selected network interface.
- Entering a valid IP address and clicking the "Change" button causes the CPU IP address to change. Test by closing the application and reopening it. Insert the CPU by the new IP address and verify that the application finds the CPU and displays the device information and IP protocol fields.
- Entering and changing a Subnet behaves as expected.
- Entering and changing the Gateway behaves as expected.

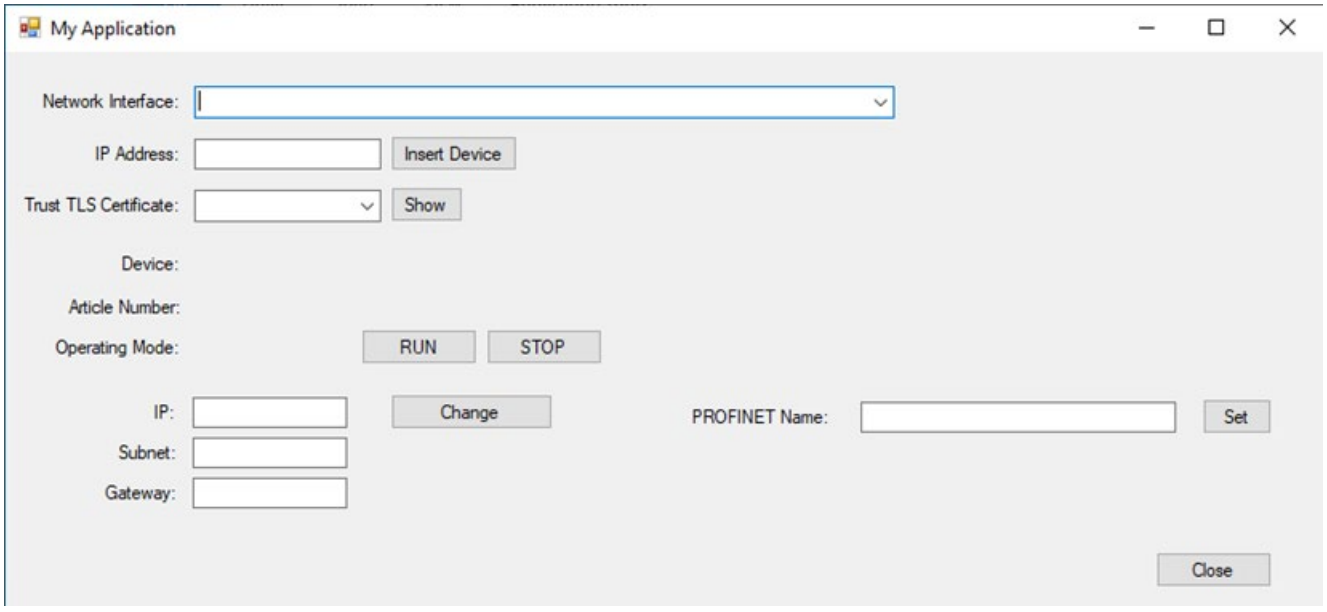
Also test various error or invalid use cases, for example:

- Attempting to change an IP address that is not in a valid format.
- Clicking the Change button with no connected CPU.

You can perform other testing as you choose. Refine your program as necessary to provide robust error handling. When you are satisfied with your results, you can proceed to Example 3: Reading and writing a CPU's PROFINET name (Page 27).

2.4 Example 3: Reading and writing a CPU's PROFINET name

Example 3 extends the application that you developed in Example 2 (Page 24). Example 3 adds a text box for the PROFINET name. The program uses this text box to display the PROFINET name and to accept user input for setting the PROFINET name. The "Set" button sets the PROFINET name of the connected CPU to the name in the PROFINET name text box.



Note

The operations in this example require the following:

- The CPU is not password-protected
- The CPU has a user program in it. (It is not reset to factory defaults.)
- The Device Configuration for the IP protocol of the PROFINET interface of the CPU specifies "IP address is set directly at the device" and "PROFINET device name is set directly at the device" in the TIA project. Depending on which TIA Portal version you are using this option might have another name:
  - Set IP address on the device
  - Set IP address using a different method

Adding logic to display and set the PROFINET name

To add and program the PROFINET name text box and Set button, follow these steps:

1. Add a TextBox to your form and name it "ProfinetName".
2. Add a button next to the TextBox and name it "SetProfinetName".
3. Add a button click event to the SetProfinetName button and insert the following code:

```
Result res = CurrentCPU.SetProfinetName(ProfinetName.Text);
```

4. Add the following code to the existing `ClearCPUStatus` (Page 15) helper method:

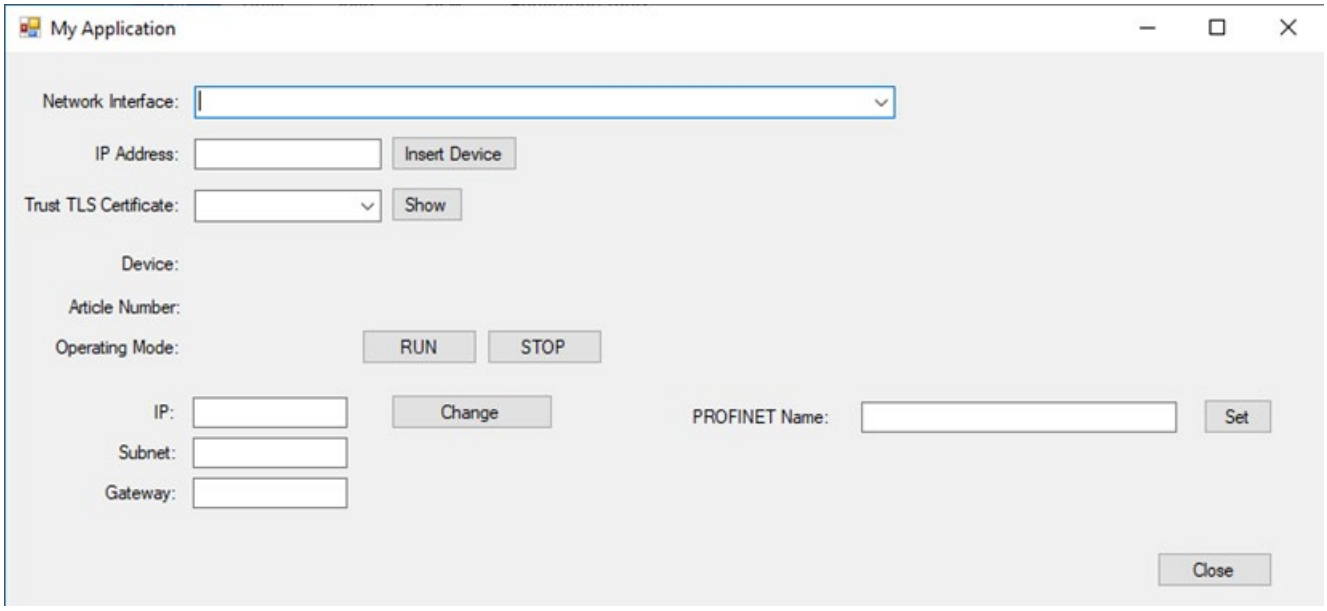
   ```
   ProfinetName.Text = string.Empty;
   ```

5. Add the following code to the existing `UpdateCPUStatus` (Page 16) helper method:

   ```
   ProfinetName.Text = CurrentCPU.ProfinetConvertedName;
   ```

2.4.1 Testing Example 3

Compile and test your application. If you followed the steps in this example, you have created an application like the following:



Verify that the following actions work correctly:

- The PROFINET name displays correctly in the PROFINET name field for the connected CPU.
- You can edit and set a new PROFINET name for the CPU.

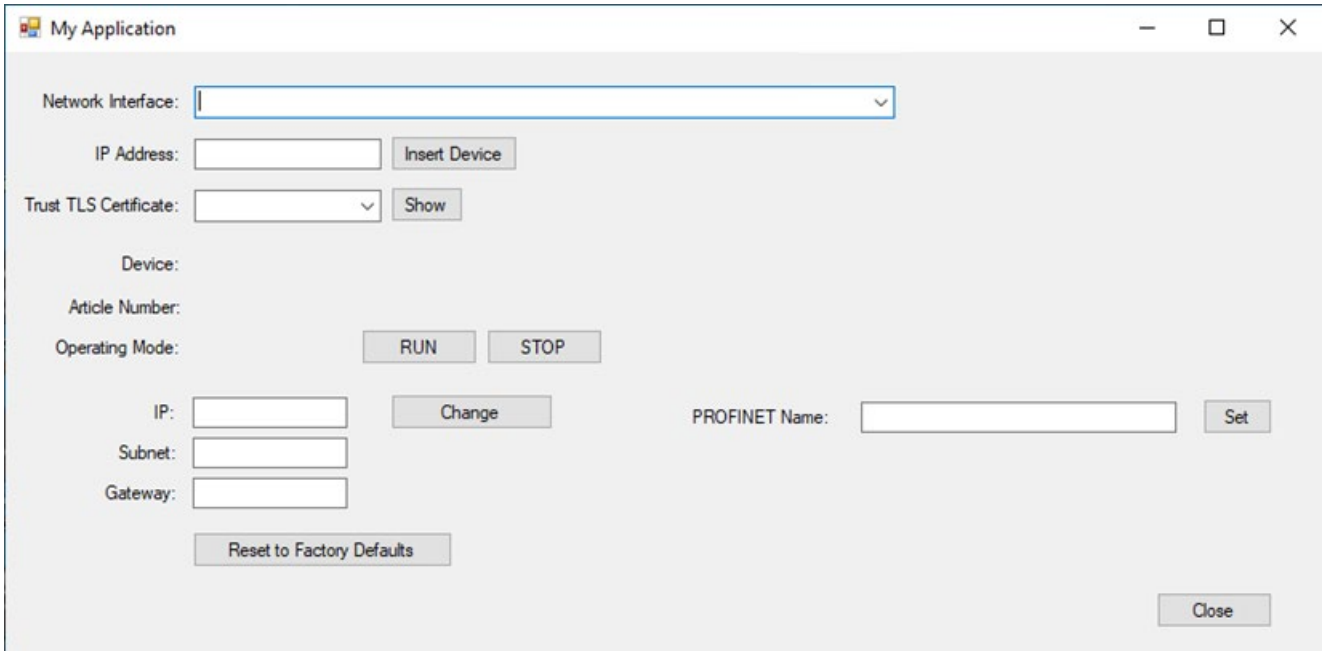
Also test various error cases, for example:

- Attempting to set a PROFINET name that is blank results in a message.
- Attempting to set a PROFINET name that doesn't follow PROFINET naming rules results in a message.

You can perform other testing as you choose. Refine your program as necessary to provide robust error handling. When you are satisfied with your results, you can proceed to Example 4: Resetting a CPU to factory defaults (Page 29).

2.5 Example 4: Resetting a CPU to factory defaults

Example 4 continues to build up the application that you developed in Example 3 (Page 27). Example 4 adds a button for the user to reset the connected CPU to factory defaults. Because resetting a CPU to factory defaults removes the user program and any associated data from the CPU, the application informs the user and asks for confirmation before continuing.



Note

In this example, note the following:

- To be able to reset the CPU to factory defaults, the CPU must not be password-protected.
- A "Reset to Factory Defaults" operation retains the IP address of the CPU.

Adding a Reset to Factory Defaults button

To add a Reset to Factory Defaults button and program the corresponding logic, follow these steps:

1. Add a button to your form and name it "`ResetToFactoryDefaults`".
2. Add the text "Reset to Factory Defaults" for the button text.
3. Add a button click event to the button and add the following code:

```
CurrentCPU.Selected = true;
CurrentCPU.SelectedConfirmed = false;

if (MessageBox.Show("Reset to Factory Defaults is a safety
relevant operation, do you wish to continue?","Reset to Factory
Defaults", MessageBoxButtons.YesNo) == DialogResult.Yes)
{
```

```
        CurrentCPU.SelectedConfirmed = true;
    }
    else
    {
        return;
    }

    Result res = CurrentCPU.ResetToFactoryDefaults();
    res = CurrentCPU.RefreshStatus();
    UpdateCPUStatus();
```

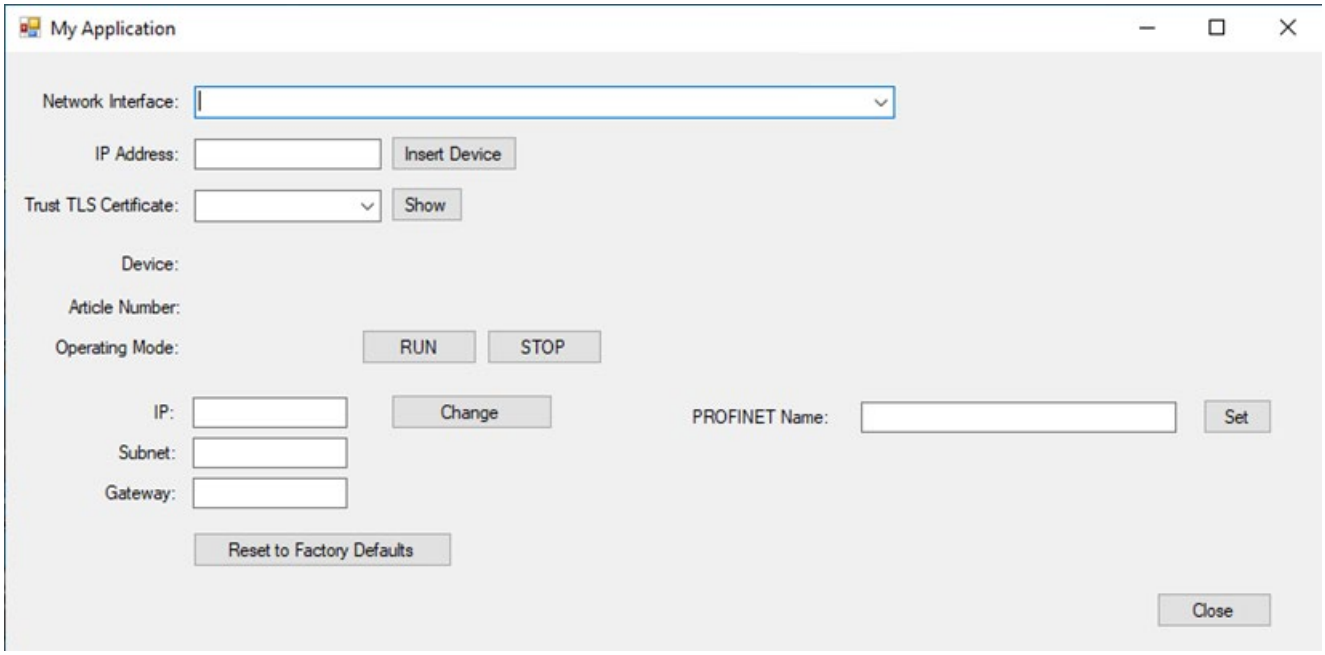
4. Add result checking as necessary.

Note

Resetting a Fail-Safe CPU to factory defaults is a safety-relevant operation. See the guidelines in User interface programming guidelines for safety-relevant operations (Page 58).

2.5.1 Testing Example 4

Compile and test your application. If you followed the steps in this example, you have created an application like the following:



Verify that clicking the "Reset to Factory Defaults" button has the following effects:

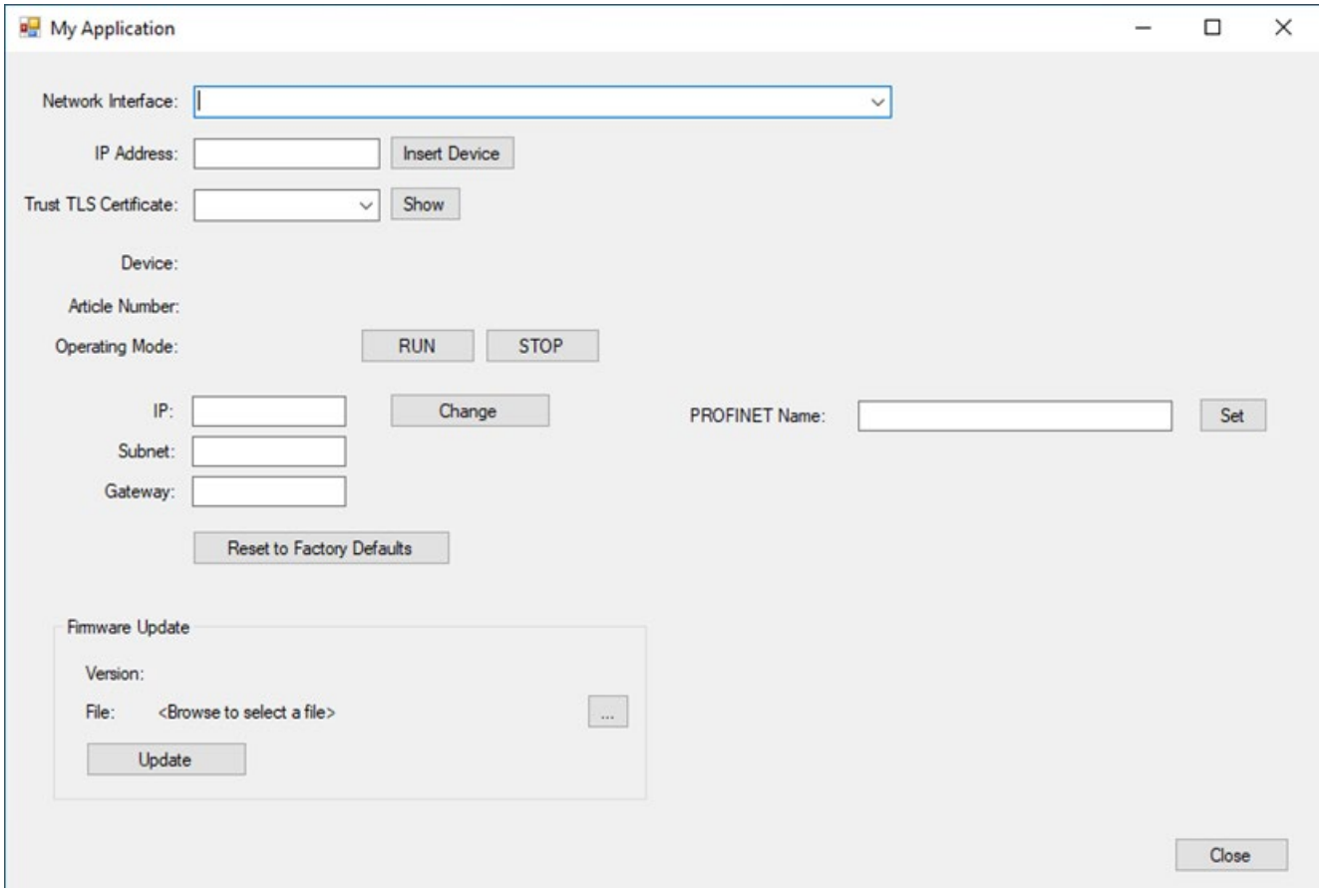
- The operation deletes the user program and data in the CPU. You can use STEP 7, for example, to display the user program.
- The IP address of the CPU remains the same.

You can perform other testing as you choose. Refine your program as necessary to provide robust error handling. When you are satisfied with your results on Example 4, you can proceed to Example 5: Updating a CPU's firmware (Page 31)

2.6 Example 5: Updating a CPU's firmware

Example 5 continues from Example 4 (Page 29) and adds the capability to perform a firmware update. To accomplish this task, your application requires the following:

- A label to display the current firmware version
- A label to display the selected firmware update file
- A browse button for selecting a firmware update file
- A button to perform the firmware update



Note

The operations in this example require the following:

- The CPU is not password-protected
- The CPU has a user program in it. (It is not reset to factory defaults.)

Adding a field to display the current firmware version

To add a field for displaying the firmware version of the connected CPU, follow these steps:

1. Add a label named "`FirmwareVersion`" for displaying the current firmware version.
2. Add the identifying text "Version:" before the label.
3. Add the following code to the existing `ClearCPUStatus` (Page 15) helper method:

   ```
   FirmwareVersion.Text = string.Empty;
   ```

4. Add the following code to the existing `UpdateCPUStatus` (Page 16) helper method:

   ```
   FirmwareVersion.Text = CurrentCPU.FirmwareVersion;
   ```

Adding a field to display the selected file

Your application requires a means to select a firmware update file and a field to display the currently-selected file. Initially, the firmware update file display is empty.

To add the file name field and file browser control, follow these steps:

1. Add a label named "`UpdateFileName`" for displaying the selected filename.
2. Add the identifying text "File:" before the label.
3. Add the following code to the existing `ClearCPUStatus` (Page 15) helper method:

   ```
   UpdateFileName.Text = "";
   ```

Adding a button to open a file browser

You will provide an open file browser control for the user to select a firmware update file. In this example, the file browser initially opens the "My Documents" folder for the user and displays files with the .upd extension. If the user browses to a different folder, the file browser remembers the last location.

To add the button, browser control, and corresponding program logic, follow these steps:

1. Add a button named "`BrowseUpdateFile`" next to the `UpdateFileName` label.
2. Create a global string to contain the path and file name returned from the browser:

   ```
   public String UpdateFilePath = String.Empty;
   ```

3. Add a button click method for the BrowseUpdateFile button and add the following code:

```
using (OpenFileDialog openFileDialog1 = new OpenFileDialog())
{
   openFileDialog1.InitialDirectory = @"%userprofile%\documents";
   openFileDialog1.Filter = "Upd Files (*.upd)|*.upd|All files
(*.*)|*.*";
   openFileDialog1.FilterIndex = 2;
   openFileDialog1.RestoreDirectory = true;


   if (openFileDialog1.ShowDialog() == DialogResult.OK)
   {
      UpdateFilePath = openFileDialog1.FileName;
      UpdateFileName.Text = openFileDialog1.SafeFileName;
   }
}
```

Adding a button to update the CPU firmware with the selected file

Next, you must add a means to update the connected CPU with the firmware that the user selected.

To add a button for performing the firmware update, follow these steps:

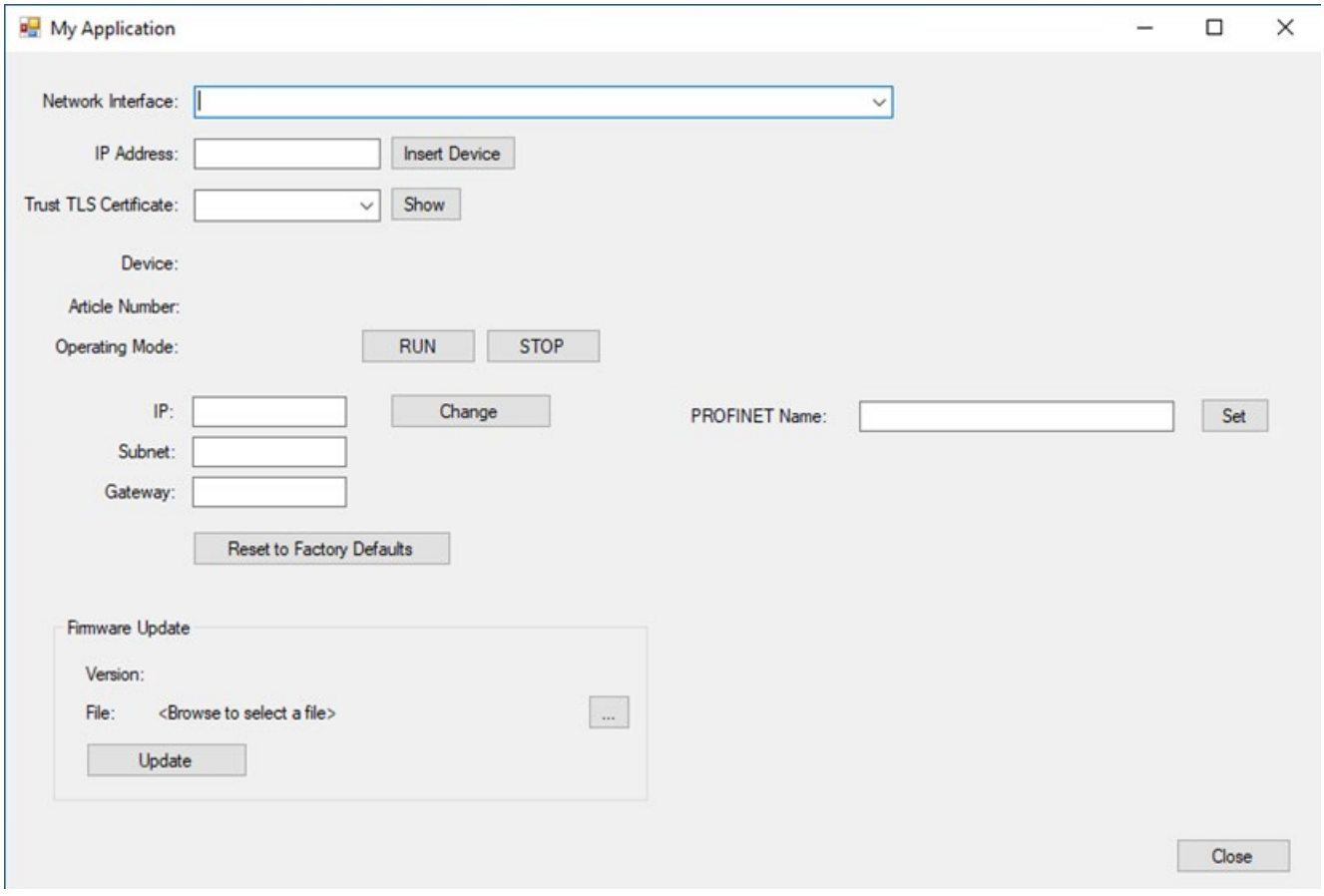
1. Add a button named "`FirmwareUpdate`" with "Update" as the button text.
2. Add a button click event to the button and add the following code:

```
CurrentCPU.Selected = true;
Result res = CurrentCPU.SetFirmwareFile(UpdateFilePath);
res = CurrentCPU.FirmwareUpdate(CurrentCPU.ID, true);
res = CurrentCPU.RefreshStatus();
UpdateCPUStatus();
```

In your actual application, add checking to verify that the selected file is valid for the connected CPU. Provide feedback to the user in case of error or invalid file selection.

2.6.1 Testing Example 5

Compile and test your application. If you followed the steps in this example, you have created an application like the following:



Verify that the following user actions work correctly:

- When you connect to a CPU with the Insert Device button, the Firmware Update section displays the correct version.
- You can use the File browse button to browse to any folder on your programming device.
- Selecting a file from the browser causes the file name to display in the File field.
- Clicking Update when you have selected a valid firmware update file causes the CPU to update the firmware to the version you selected.

Also test various error cases, for example:

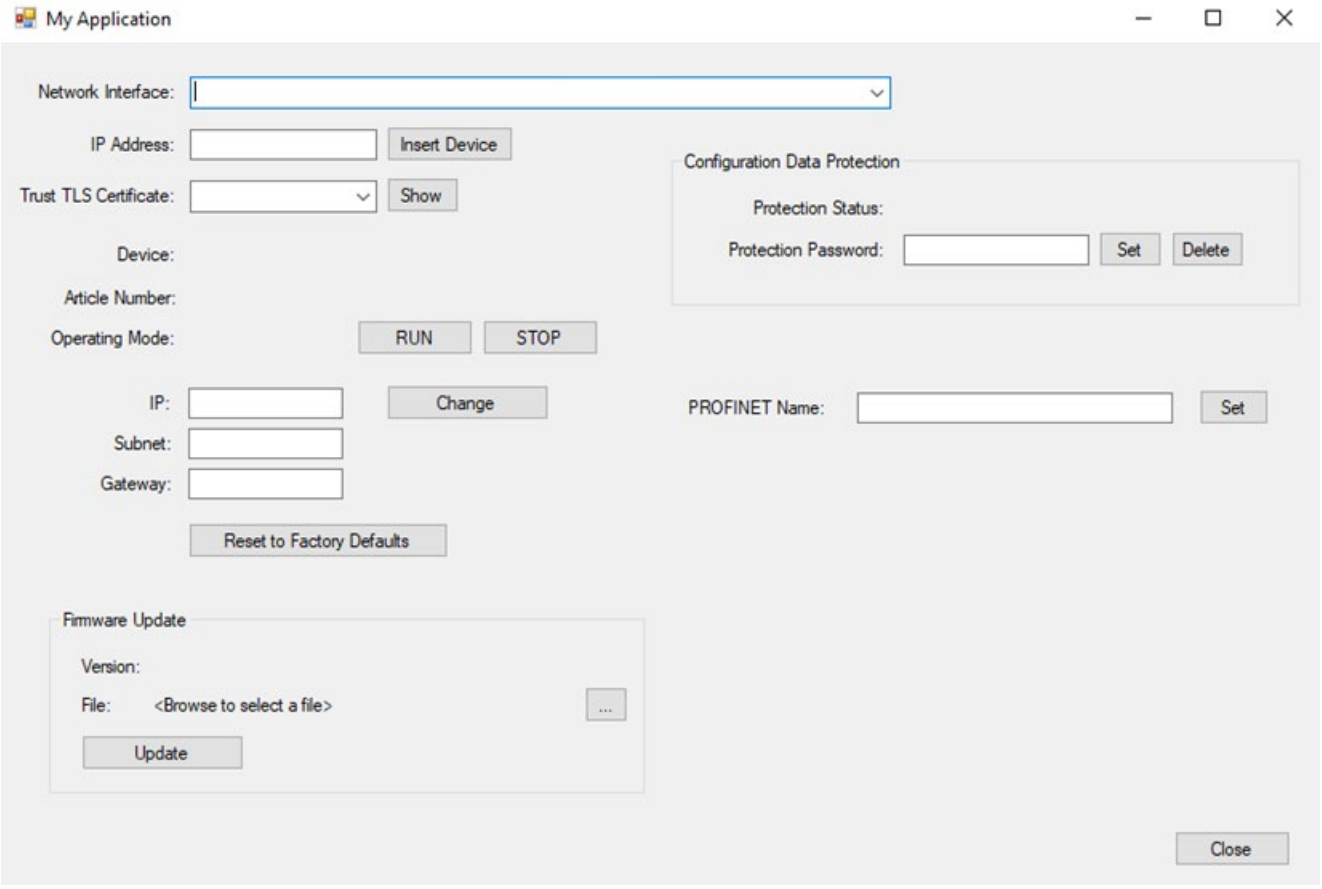
- Clicking Update with no CPU connected causes an error message to display.
- Clicking Update when you have selected an invalid firmware update file causes an error message to display.

You can perform other testing as you choose. Refine your program as necessary to provide robust error handling. When you are satisfied with your results on Example 5, you can proceed to Example 7: Backing up a CPU and restoring a CPU from a backup file (Page 38)

2.7 Example 6: Managing configuration data protection

Example 6: Managing Configuration Data Protection

Example 6 continues from Example 5 and adds the capability to manage configuration data protection.



Note: This example assumes that you are connected to a CPU that supports Configuration Data Protection, which is called Protection of Confidential PLC Configuration Data in the TIA Portal. CPU firmware versions and STEP 7 device configurations that support Configuration Data Protection are the following:

- S7-1500 V2.9 or higher
- S7-1200 V4.5 or higher

Furthermore, your program must contain the following:

- A field to display the current data protection status
- A text box for entering the data protection password
- A button to set the protection password
- A button to delete the password

Adding a field for displaying the current Configuration Data Protection version

To add a field for displaying the configuration data protection:

1. Add a label named "CdpStatus" for displaying the current protection status.
2. Add the identifying text "Protection Status:" before the label.
3. Add the following code to the existing ClearCPUStatus (Page 15) helper method:
   `CdpStatus.Text = string.Empty;`
4. Add the following code to the existing UpdateCPUStatus (Page 16) helper method:
   `CdpStatus.Text = CurrentCPU.ConfigurationProtectionData.ToString() ;`

Adding fields and buttons to set or clear the password

Your application requires a means by which the user can specify the data protection password. To add a textbox and buttons, follow these steps:

1. Add a TextBox to your form and name it "CdpPW".
2. Add a button next to the TextBox and name it "CdpSet".
3. Add a button click event to the the CdpSet button button and insert the following code:

```
private void CdpSet Click(object sender, EventArgs e)
{
   CurrentCPU.SetConfigurationDataProtectionPassword(new
EncryptedString(CdpPW.Text));
   UpdateCPUStatus();
}
```

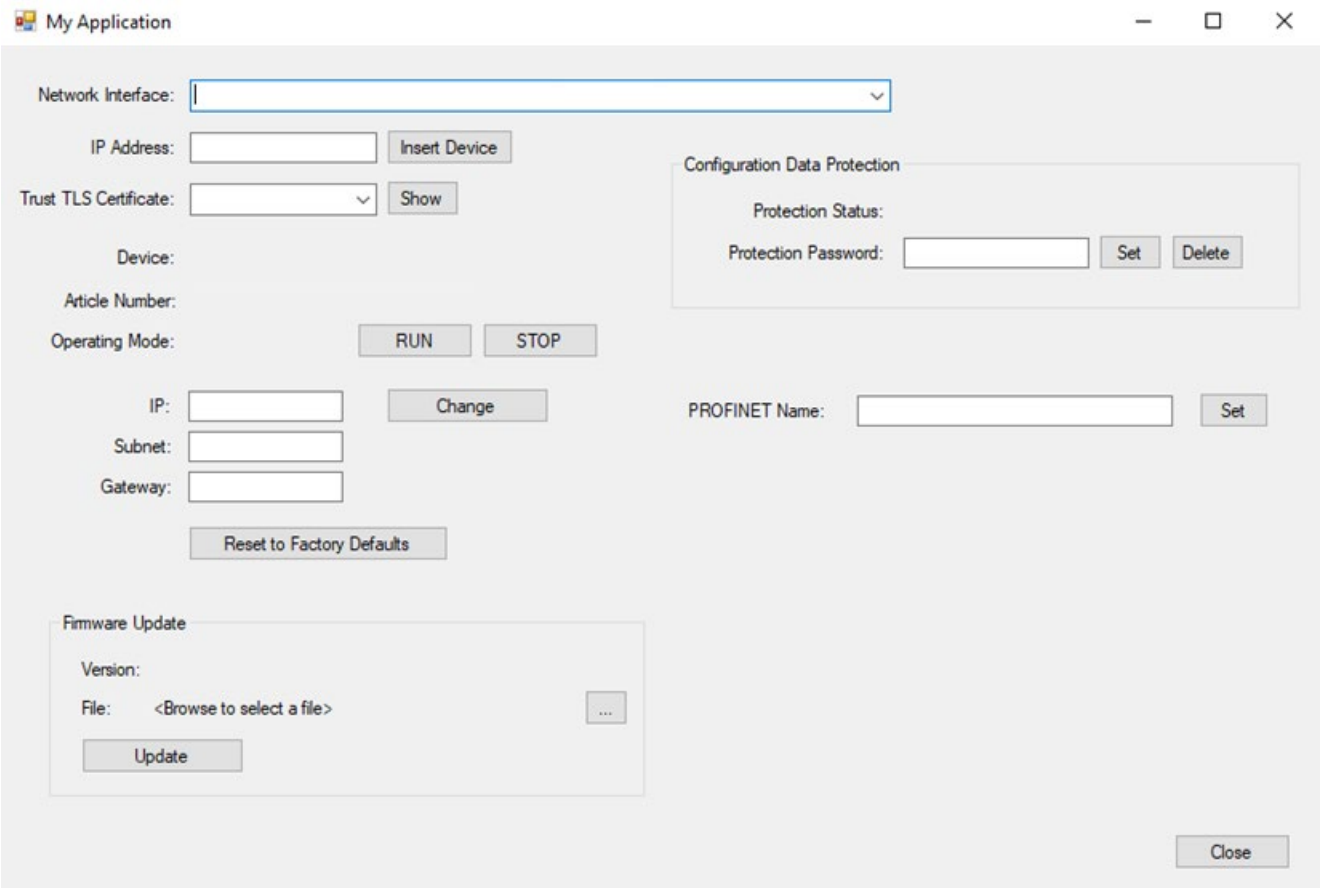
4. Add a button next to the CdpSet button and name it "CdpDelete".
5. Add a button click event to the CdpSet button and insert the following code:

```
private void CdpDelete Click(object sender, EventArgs e)
{
   CurrentCPU.DeleteConfigurationDataProtectionPassword();
   UpdateCPUStatus();
}
```

2.7.1 Testing Example 6

Testing Example 6

Compile and test your application. If you followed the steps in this example, you have created an application like the following:



Verify that the following user actions work correctly:

- The Protection Status field correctly displays the data protection status.
- You can enter a password in the Password Protection field.
- Clicking the Set button sets the protection password and results in a change in the Protection Status field.

Also test various error cases, such as:

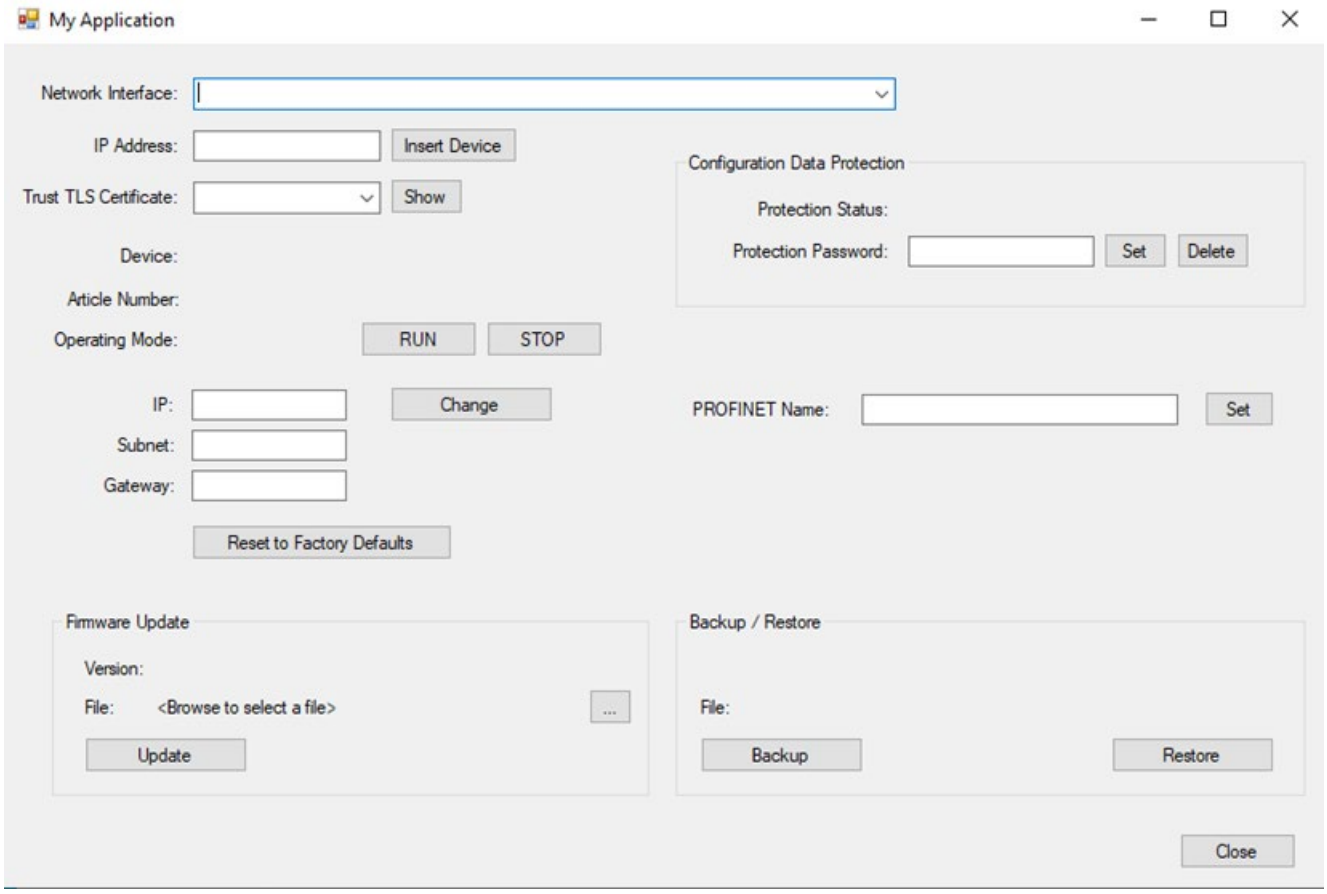
- Clicking Set or Delete with no CPU connected causes an error message to be displayed.
- Clicking Set or Delete with Trust set to "Never" causes an error message to be displayed.

You can perform other testing as you choose. Refine your program as necessary to provide robust error handling. When you are satisfied with your results, you can proceed to Example 7: Backing up a CPU and restoring a CPU from a backup file (Page 32).

2.8 Example 7: Backing up a CPU and restoring a CPU from a backup file

Example 7 continues from Example 6 (Page 31) and adds the capability to back up the current CPU program and to restore the CPU program. To accomplish this task, your application requires the following:

- A label to display the name of the backup / restore file
- A button to launch the backup operation
- A button to launch the restore operation



Note

The operations in this example require the following:

- The CPU is not password-protected.
- The CPU has a user program in it.

Adding the backup / restore file label

To add the label field for the backup / restore file name, follow these steps:

1. Add a label to the form and name it `BrFile`.
2. Add a global string to the form and name it `BrFilespec`:

```
public String BrFilespec = String.Empty;
```

3. In the method `ClearCPUStatus` (Page 15), add the following line of code:

```
BrFile.Text = string.Empty;
```

Programming the button to back up the CPU to a file

To add the backup button for this example and program the logic, follow these steps:

1. Insert a button control and name it "`Backup`".
2. To be able to perform file operations, add the following `using` statement to the existing using statements:

```
using System.IO;
```

3. Add a click event to the `Backup` button and populate it with the following code:

```
BrFilespec =
Environment.GetFolderPath(Environment.SpecialFolder.UserProfile) +
"\\" + CurrentCPU.Name + " " + CurrentCPU.MACString.Replace(':',
'-') + ".s7pbkp";

if (File.Exists(BrFilespec))
    File.Delete(BrFilespec);

CurrentCPU.Selected = true;
Result res = CurrentCPU.Backup(BrFilespec);
BrFile.Text = BrFilespec;
```

Programming the button to restore the CPU from the backup file

To add the restore button for this example, and program the logic, follow these steps:

1. Insert a button control and name it "`Restore`".
2. Add a click event to the `Restore` button and populate it with the following code:

```
CurrentCPU.Selected = true;
CurrentCPU.SelectedConfirmed = false;
if (MessageBox.Show("Restore is a safety relevant operation, do
you wish to continue?",
        "Backup / Restore Defaults", MessageBoxButtons.YesNo) ==
DialogResult.Yes)
{
    CurrentCPU.SelectedConfirmed = true;
}
```

```
else
{
    return;
}

Result res = CurrentCPU.SetBackupFile(BrFilespec);
res = CurrentCPU.Restore();
```

For both button operations, check the results of operations and handle as you prefer.

Note

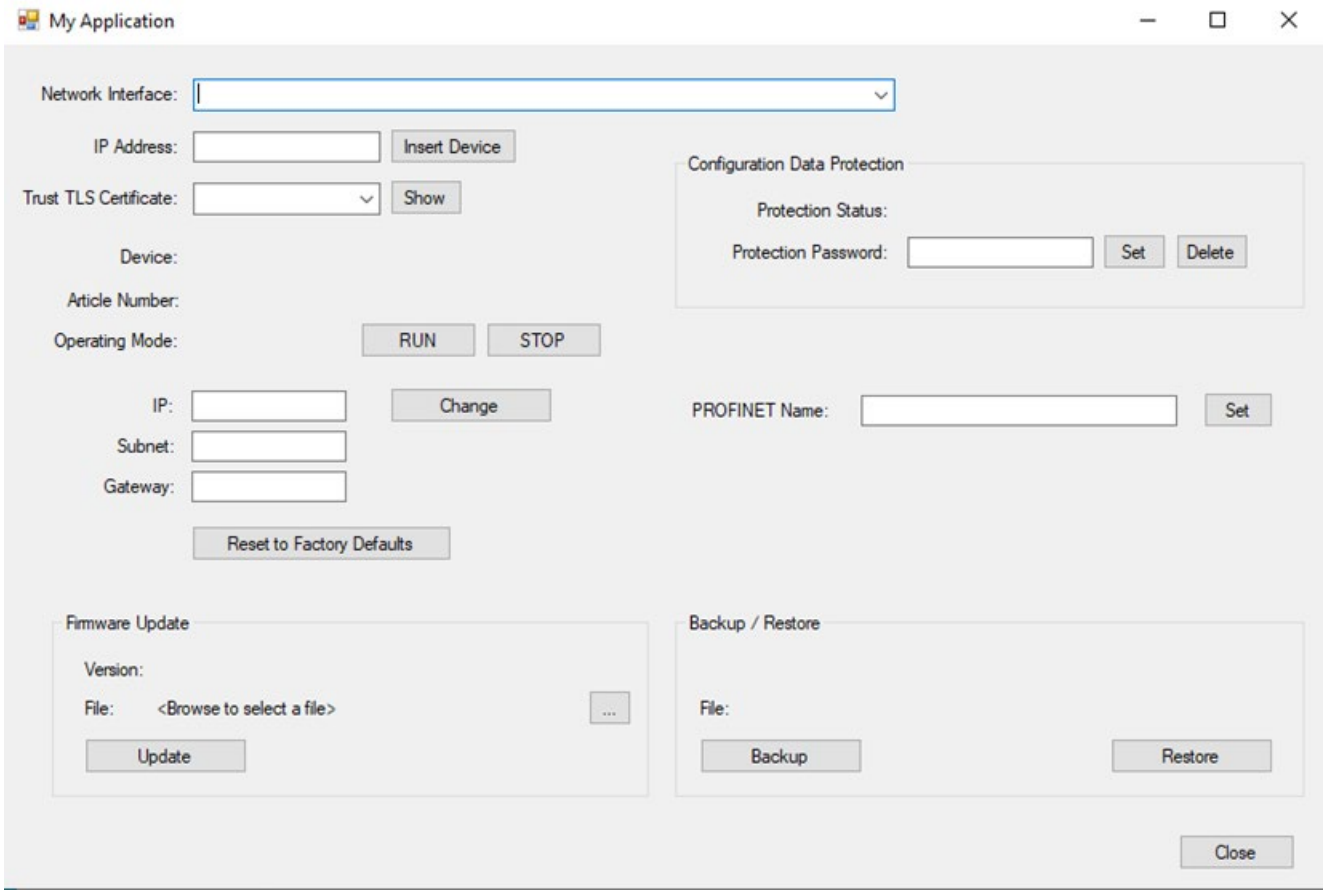
Restoring a program to a Fail-Safe CPU is a safety-relevant operation. See the guidelines in User interface programming guidelines for safety-relevant operations (Page 58).

Possible extensions

This example backs up a CPU program and configuration and restores it to the same CPU. You might want to add additional controls and extend the logic in this example to back up one CPU and restore it to a different CPU. Once you are knowledgeable about the capabilities of the API (Page 45), you can perform a wide variety of device operations on many types of devices on your network.

2.8.1 Testing Example 7

Compile and test your application. If you followed the steps in this tutorial, you have created an application like the following:



Verify that the following user actions work correctly:

- If you are connected to a CPU that has a user program and is not password-protected, clicking the Backup button displays a backup file name in the File field.
- If you click Reset to Factory Defaults and then click Restore, the application reloads the backup file into the reset CPU.
- If you connect to a different CPU with the Insert Device button, the backup / restore file is empty.

Also test various error cases, for example:

- Trying to back up a CPU with no program does not generate a backup file.
- Trying to back up a CPU that is password-protected does not generate a backup file.
- Trying to back up a CPU with the TLS Trust level set to "Never" displays an error message.
- Trying to restore a backup file to a CPU after you add password protection to the CPU fails.

You can perform other testing as you choose. Refine your program as necessary to provide robust error handling.

Comparing to the installed Siemens example program

When you ran the SAT SDK File Extractor (Page 13), you extracted an Examples folder. After you complete the tasks in this tutorial, you can compare your `Form1.cs` file to the `Form1.cs` file in the extracted Examples folder. You can compile and run the Siemens example and compare it against your application.

2.9 Distributing an application to your end user

The end user of your application needs the following:

- The Bin folder that the SAT SDK File Extractor (Page 13) created
- The Prerequisite folder that the SAT SDK File Extractor created, or at a minimum the "SIMATIC Automation Tool SDK PreReq Installer.exe" file from the Prerequisite folder
- Your custom application .exe file and any other custom files belonging to your application.

Place your custom files in the Bin folder along with the extracted API files.

Third-party software license conditions and copyrights

The SIMATIC Automation Tool SDK installs a document listing third-party software information, including license conditions and copyrights, and open source software information. The setup installs this file for each language in the Documents folder of the installation folder as follows:

Language folder	File name
Chinese	SAT_SDK.ReadMe.OSS.V4.0.SP3.zh-CHS.rtf
Deutsch	SAT_SDK.ReadMe.OSS.V4.0.SP3.de-DE.rtf
English	SAT_SDK.ReadMe.OSS.V4.0.SP3.en-US.rtf
Espanol	SAT_SDK.ReadMe.OSS.V4.0.SP3.es-ES.rtf
Francais	SAT_SDK.ReadMe.OSS.V4.0.SP3.fr-FR.rtf
Italiano	SAT_SDK.ReadMe.OSS.V4.0.SP3.it-IT.rtf

Siemens recommends that you include the contents of these documents in the files you provide to your users as well as any other third-party software information.

Creating a setup for your user

Using the tool of your choice, create a setup for your user that installs the necessary files and folders. Implement the setup to call the following command:

```
"SIMATIC Automation Tool SDK PreReq Installer.exe" -silent -
norestart
```

You can find an example of this command line in the "Install Prerequisites Example.bat" file in the Prerequisite folder.

Silent option

The SDK PreReq Installer might not install silently. In addition, the included installation of .Net version 4.8 might fail if you already have .Net version 4.8 on your programming device. You can ignore the error.

Updating a custom application made with a previous version of the SDK

3

If you have created and distributed a custom application using a previous version of the SIMATIC Automation Tool SDK, you can update it to the latest version of the SDK and redistribute it to your end users. Updated versions of the SIMATIC Automation Tool SDK can provide improvements and enhancements, which might change the Application Programming Interface (API) (Page 45). Additionally, support components can differ from one release to the next.

Siemens recommends that you update your custom application to the latest released version:

- Recompile your project (Page 13).
- Correct any compilation errors that might result from API changes.
- Test your application to ensure functional compatibility with the API.
- Distribute your updated project (Page 42) to your end user as described previously.

SIMATIC Automation Tool API for .NET framework 4

4.1 Getting started with the API

The SIMATIC Automation Tool Application Programming Interface (API) for PROFINET network and device maintenance allows you to create custom applications using an extensive set of .NET interfaces, classes, and methods (Page 52). You will design your custom application for your specific purposes.

Setting a network interface

The first task your application must perform is to set or select a network interface for communicating to your PROFINET network. Your computer or programming device is likely to have multiple network interfaces.

If you do not know the name of the network interface you wish to use, you can use the `QueryNetworkInterfaceCards` (Page 83) method of the Network class to read a list of all network interfaces. You can present this list to the user of your application to select a specific network interface. Recommend that the users of your application select the network interface with "Auto" in the name. The "Auto" selection allows the API to find all devices.

You then use the `SetCurrentNetworkInterface` (Page 83) method to set the network interface that your application will use for device communication.

Communicating to devices

As the application designer, you can choose whether to operate on devices found by a network scan or on devices that are added by the Insert Device methods (Page 100). If you only have a few devices on your network and you know their IP or MAC addresses, then use the Insert Device methods for application performance. If you do not know the IP or MAC addresses, then use the `ScanNetworkDevices` (Page 84) method to find all devices connected to your PROFINET network.

Scanning a network

Your application might need to connect to a PROFINET network and scan for all attached devices. In this case, your application can scan the network to fill a collection of all devices on your network. The API provides a `ScanNetworkDevices` (Page 84) method for this purpose. This method sends a broadcast DCP command to the network and fills an `IProfinetDeviceCollection` (Page 90) of the accessible devices.

The method also allows specifying a filter that limits which devices are included in the scan. The filter defines which IP addresses on the network are candidates for inclusion in the device table. Any devices that do match the IP address filter are omitted from the scan results. This can substantially reduce the scan and device initialization time.

Inserting devices

If you or your application user knows the IP or MAC addresses of your devices, you might want to design your application so that you or your user can add specific devices. The API provides methods for inserting a device by either IP address or MAC address. If you anticipate your user performing operations on specific devices after adding them, you might choose to develop an application based on the `InsertDeviceByIP` (Page 100) method or `InsertDeviceByMAC` (Page 101) method.

Note

Devices behind routers

If you are communicating to devices that are behind a router, you must use the `InsertDeviceByIP` (Page 100) method. A network scan does not discover devices behind routers.

Working with protected CPUs

If a CPU is protected, the `Protected` (Page 131) property is `true` for the `ICPU` interface. You must first check whether a CPU is protected before calling the `SetPassword` (Page 164) method. If the CPU is NOT protected, do NOT call the `SetPassword` (Page 164) method. If you call the `SetPassword` (Page 164) method for a CPU that is not protected, the API throws a critical error exception. The critical error exception lets you know that you are using the API incorrectly.

For CPUs that are protected, your application must call the `SetPassword` (Page 164) method with a password that provides a sufficient access level (Page 217) for the operation. After you set a valid password, you can refresh the status of the device.

The best practice is to call `SetPassword` (Page 164) on all devices for which you wish to communicate immediately after scanning the network. Set the passwords on standard CPUs to the write access password (Page 217) and for F-CPUs to the safety password (Page 217) to avoid API errors.

CPUs might have firmware that supports Transport Layer Security to provide secure communications. Transport Layer Security (TLS) is a cryptographic protocol designed to provide communications security over a communications network. When performing device operations on CPUs that support secure communications, the SAT API client must fulfill the additional TLS requirements such as trusting the CPU's digital certificate.

Refer to the certificate-handling API interfaces for additional information. Knowledge about TLS concepts and certificate handling will be helpful.

Using RefreshStatus

Objects in the `IProfinetDeviceCollection` (Page 90) that represent devices have only partial data from each device after a network scan. To obtain all the data about a device and to use the API properly, you must do the following:

1. Call the `SetPassword` (Page 164) method for each protected CPU. The CPU password must provide sufficient privileges to read all the device data.
2. Call the `RefreshStatus` (Page 119) method for each device in the `IProfinetDeviceCollection` (Page 90).

The `RefreshStatus` (Page 119) method updates all the data that represents the status of the device.

Critical error exceptions

If the API throws a critical error exception, then you are not using the API correctly for the status of the device. Remember to call the `RefreshStatus` (Page 119) method to keep the data about a device up to date before using additional API functions.

Additional requirements

Many of the API methods have specific prerequisites for using the methods. The method and interface descriptions list these prerequisites as applicable. For device-specific operations, the application must set the Selected property to true to perform the device operation.

Safety-relevant operations on F-CPUs (Page 56) require additional safeguards in your application design.

Example: Initializing devices by scanning the network

```
using Siemens.Automation.AutomationTool.API;using
System.Security.Cryptography.X509Certificates;

#region Getting started with a network scan
//-----------------------------
// Setting a Network Interface
//-----------------------------
Network myNetwork = new Network();
List<String> interfaces = new List<String>();
Result retVal = myNetwork.QueryNetworkInterfaceCards(out interfaces)
;
if (retVal.Succeeded)
{
   for (Int32 index = 0; index < interfaces.Count; index++)
   {
      String strInterfaceName = interfaces[index];
      // ------------------------------------------
      // Find the NIC that is connected to your
      // PROFINET network or display a dialog and let
      // the user select a NIC from the list
      // ------------------------------------------
      if (strInterfaceName == "myNIC.Auto.1")
      {
         retVal = myNetwork.SetCurrentNetworkInterface(strInterfaceN
ame);
      }
   }
}
//--------------------
// Scanning a Network
//--------------------
IProfinetDeviceCollection scannedDevices;
IScanErrorCollection scanResults = myNetwork.ScanNetworkDevices(out
scannedDevices);

if (scanResults.Succeeded)
{
   foreach (IProfinetDevice device in scannedDevices)
   {
      ICPU cpu = device as ICPU;
      if (cpu != null)
      {
         if (cpu.Protected)
         {
            //-----------------------------------------
            // Set the password for all CPUs on your
            // PROFINET network with at least read-write
            // privileges to use the API
            //-----------------------------------------

            retVal = cpu.SetPassword(new EncryptedString("CPUPasswor
d"));
```

```
            }
        }
        //-----------------------------------------------------
        // Update all device data that was not available on scan
        //-----------------------------------------------------
        retVal = device.RefreshStatus();
        if (retVal.Succeeded)
        {
            // Do we have any certificates?
            if (cpu.CertificateStore.Count > 0)
            {
                // Optional view of certificates in Windows dialog -
Remove if no UI
                cpu.CertificateStore.ShowDialog();
                bool bTrust = true;
                if (bTrust)
                    retVal = cpu.SetTrustCertificateStore(TrustCertifica
teType.Always);
                else
                    retVal = cpu.SetTrustCertificateStore(TrustCertifica
teType.Never);
             }
        }
    }
}
//-------------------------------------------------------------
// You are now ready to use the API and call device operations
//-------------------------------------------------------------
#endregion
```

Example: Initializing devices by inserting devices

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

#region Getting started with inserting devices
//-----------------------------
// Setting a Network Interface
//-----------------------------
Network myNetwork = new Network();
List<String> interfaces = new List<String>();
Result retVal = myNetwork.QueryNetworkInterfaceCards(out interfaces)
;
if (retVal.Succeeded)
{
   for (Int32 index = 0; index < interfaces.Count; index++)
   {
      String strInterfaceName = interfaces[index];
      // ------------------------------------------
      // Find the NIC that is connected to your
      // PROFINET network or display a dialog and let
      // the user select a NIC from the list
      // ------------------------------------------
      if (strInterfaceName == "myNIC.Auto.1")
      {
         retVal = myNetwork.SetCurrentNetworkInterface(strInterfaceN
ame);
         if (retVal.Succeeded)
         {
            break;
         }
      }
   }
}
```

```
//--------------------
// Inserting Devices
//--------------------
IProfinetDeviceCollection insertedDevices = Network.GetEmptyCollecti
on();
IProfinetDevice insertedDevice = null;
IScanErrorCollection
insertErrorCollection = insertedDevices.InsertDeviceByIP(0xC0A80001,
out insertedDevice); // 192.168.0.1

if (retVal.Succeeded)
{
   foreach (IProfinetDevice device in insertedDevices)
   {
      ICPU cpu = device as ICPU;
      if (cpu != null)
      {
         if (cpu.Protected)
         {
            //-----------------------------------------
            // Set the password for all CPUs on your
            // PROFINET network with at least read-write
            // privileges to use the API
            //-----------------------------------------

            retVal = cpu.SetPassword(new EncryptedString("CPUPasswor
d"));
         }
      }
      //------------------------------------------------------------
      // Update all the device data that was not available on scan
      //------------------------------------------------------------
      retVal = device.RefreshStatus();

      // Do we have any certificates?
      if (cpu.CertificateStore.Count > 0)
      {
          // Optional view of certificates in Windows dialog -
Remove if no UI
          cpu.CertificateStore.ShowDialog();
          bool bTrust = true;
          if (bTrust)
             retVal = cpu.SetTrustCertificateStore(TrustCertificateT
ype.Always);
          else
             retVal = cpu.SetTrustCertificateStore(TrustCertificateT
ype.Never);
       }
   }
}
//------------------------------------------------------------
// You are now ready to use the API and call device operations
//------------------------------------------------------------
```

```
/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

See also

API software and version compatibility (Page 55)

4.1.1 Architectural overview

The API provides classes, interfaces, and methods to support communication with a PROFINET network of SIMATIC devices.

Note

Use only documented API classes, interfaces, and methods in your application

The API includes public methods to support the SIMATIC Automation Tool user interface in addition to public methods for your application programming. In your application development, use only the classes, interfaces, and methods that this User Guide describes. Any other public methods that exist in the API that the User Guide does not describe are reserved for exclusive use by the SIMATIC Automation Tool user interface.

Networks

The `.NET` class `Network` (Page 82) represents the PROFINET network as a whole. This class performs functions using a network interface card (NIC) installed on a programming device. The `Network` class is used to search for available network interface cards and to select the network interface that is attached to your PROFINET network.

The Network class includes the constructor and the following property and methods:

- Network constructor (Page 82)
- QueryNetworkInterfaceCards method (Page 83)
- SetCurrentNetworkInterface method (Page 83)
- CurrentNetworkInterface property (Page 84)
- ScanNetworkDevices method (Page 84)
- SetCommunicationsTimeout method (Page 85)
- GetCommunicationsTimeout method (Page 86)
- GetEmptyCollection method (Page 87)

Devices

The individual devices on the PROFINET 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` (Page 103) – All PROFINET devices directly accessible on the PROFINET network can be represented by this interface, because all devices are derived from this class.

`ICPU` (Page 130) – This represents S7-1X00 CPUs that are directly connected to the network. Specific functionality is supported for CPUs.

`ICPUClassic` (Page 176) – This represents classic type S7-300 and S7-400 CPUs that are directly connected to the network.

`IHMI` (Page 178) – This represents SIMATIC HMIs that are directly connected to the network. Specific functionality is supported for HMIs.

`IBaseDevice` (Page 75) – This interface is used to represent devices not directly connected to the PROFINET 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` (Page 129) – This interface is used to represent individual I/O modules that are plugged into a CPU, PROFINET device, or PROFIBUS station.

`IHardware` (Page 74) – 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.

`IScalance` (Page 186) – This is the interface that represents SCALANCE devices.

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

`IProfinetDeviceCollection` (Page 90) – A collection of all devices directly accessible on the network.

`IModuleCollection` (Page 76) – A collection that may represent modules plugged into a CPU or IM.

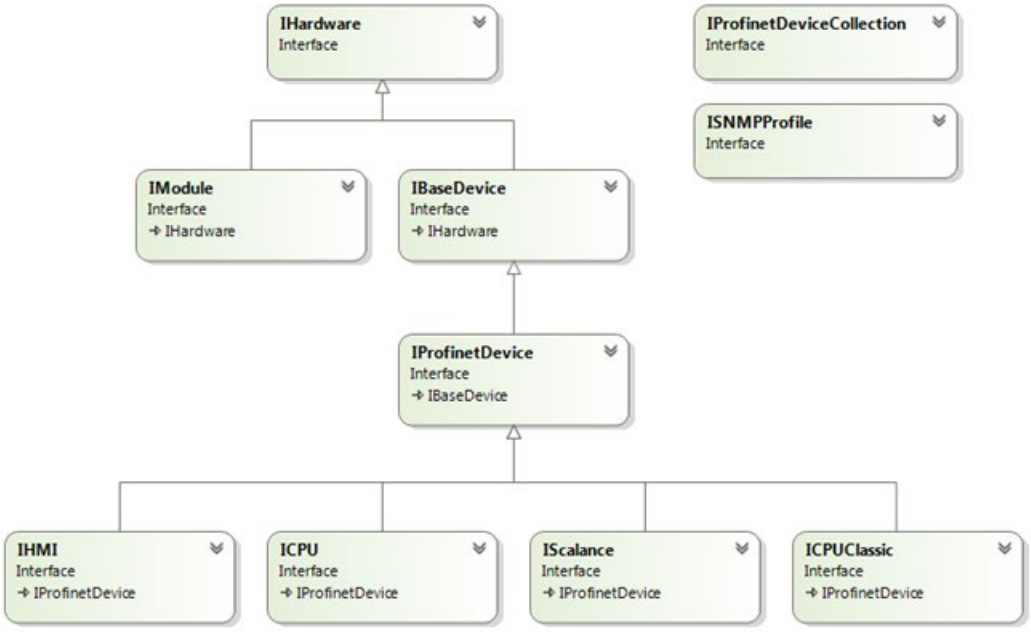
`IHardwareCollection` (Page 76) – This collection represents a CPU and all its modules.

`IScanErrorCollection` (Page 76) – This collection represents the set of errors returned from all devices on a network scan operation.

Device classes, interfaces, and methods:

- IProfinetDeviceCollection class (Page 90)
- IProfinetDevice interface (Page 103)
- ICPU interface (Page 130)
- IHMI interface (Page 178)

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



Note

See the example PROFINET network and the SIMATIC Automation Tool API classes that are used to represent each network component.

4.1.2 Referencing the API in a user interface application

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

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

When using the API, these files and the `AsModels` folder must be in the same folder where you develop your custom application.

Programming environment

Siemens developed the API with Microsoft Visual Studio 2017 using the .NET Framework 4.8.

To create a project that can work with the SIMATIC Automation Tool API, follow these steps:

1. In Visual Studio, create a new project that is a Visual C# Windows Forms Application.
2. From the Solutions Platforms drop-down list, select "Configuration Manager".
3. In the Configuration Manager, click the Platform drop-down and create a new platform.
4. From the New Project Platform dialog, select "x64" for the new platform. Click OK and close the Configuration Manager.
5. In the project properties, set the target framework to .NET Framework 4.8.

References in your solution

To include the API in your application, add `SIMATICAutomationToolAPI.dll` as a reference in your Visual Studio solution.

To be able to export PC data from your applications, add `SimaticAutomationToolHealthCheck.dll` as a reference.

Required "using" statements

In any application that references the API classes, you must add the following statement referencing the API namespace:

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

If your application exports PC Data, you must add the following statement:

```
using Siemens.Automation.AutomationTool.HealthCheck;
```

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

4.2 API software and version compatibility

You must purchase the SIMATIC Automation Tool SDK and agree to the license terms and conditions to use the SDK API (Application Programming Interface). The setup provides license terms and conditions for you to accept. If you have purchased the SDK, you do not have to also purchase and license the SIMATIC Automation Tool to use the API. Refer to the SIMATIC Automation Tool SDK Installation notes for instructions on purchasing and installation.

You can use the SDK API for programming custom applications after you perform these tasks:

- Purchase the SIMATIC Automation Tool SDK
- Install the SIMATIC Automation Tool SDK
- Agree to all the license terms when installing the SDK

Compatibility with previous versions

If you have .NET framework code in .cs (C#) files from the SIMATIC Automation Tool V3.1 or V3.1 SP1 API, you can use this code in developing your application. You will need to update your Visual Studio reference to the API.

You might need to change some of your method calls to use the updated versions of some methods. Compiling your project shows you where you need to make changes. Refer to the method header definitions in this user guide to make the necessary changes.

Take note of the CPU password access levels (Page 217) for using methods for CPU operations, especially for operations on fail-safe CPUs (Page 56).

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

4.3 Requirement for S7 communications

S7 communications as you develop your application

To run applications that you develop with the API, you must have S7 communications components on your programming device. The setup program for the SIMATIC Automation Tool SDK installs the required S7 communications components. Run the SAT SDK File Extractor (Page 13) to save the API, HMI, S7 communication components, and other required components to a folder. Place your executable for your custom application in the same folder. Run the application from this folder and perform your testing.

S7 communications for end users of your application

You will typically create a setup for the users of your custom application. You can create this setup, for example, with the Windows Installer that the SIMATIC Automation Tool SDK provides. When your customer runs your setup, this setup installs the API, HMI, S7 communication components, other required components and your custom application in a folder. Your user can then run your custom application from that folder.

4.4 Designing a user interface application for fail-safe devices and safety-relevant operations

4.4.1 API support for safety-relevant operations and fail-safe devices

The SIMATIC Automation Tool API supports the following safety-relevant operations:

- Program update
- Restore device from backup file

- Reset to factory defaults
- Format memory card

Note

The *SIMATIC Safety - Configuring and Programming* manual contains a warning identified as "S078". This warning states, "When using tools for the automation or operation (of TIA Portal or Web server) which allow access protection for the F-CPU to be bypassed (e.g. saving or automatic entry of a CPU password for the protection level "Full access incl. fail-safe (no protection)" or Web server password), the safety relevant project data may not be protected against unintentional changes anymore."

This S078 warning does not apply to the SIMATIC Automation Tool API. The API supports communicating with F-CPUs and storing CPU passwords for F-CPUs.

Safety features that the API provides

TÜV SÜD has certified the API for the SIMATIC Automation Tool and SIMATIC Automation Tool SDK.

If your custom application performs a safety-relevant operation to an F-CPU, you should certify your application with TÜV SÜD or an equivalent company. Just because the Siemens API is certified doesn't mean that your application is automatically certified.

The API provides a set of protections for your application. Your application should also follow the recommended set of user interface guidelines for working with F-CPUs (Page 58).

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

- Independent connection and legitimization process for each safety-relevant operation
- Identity checks for safety-relevant operations to fail-safe devices
- Identification of safety programs
- Required use of safety password (Page 217) for any safety-relevant operation to a protected F-CPU
- Use of 32-bit CRC checksums to compare fail-safe device online and offline representations
- Use of hamming codes (Page 66) to indicate TRUE and FALSE states
- Comparison of F-signatures after the program update and restore from backup operations to verify that the operation completed successfully

4.4.2 User interface programming guidelines for safety-relevant operations

WARNING

Protect safety-relevant operations as much as possible

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

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

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

Identifying and protecting safety-relevant operations

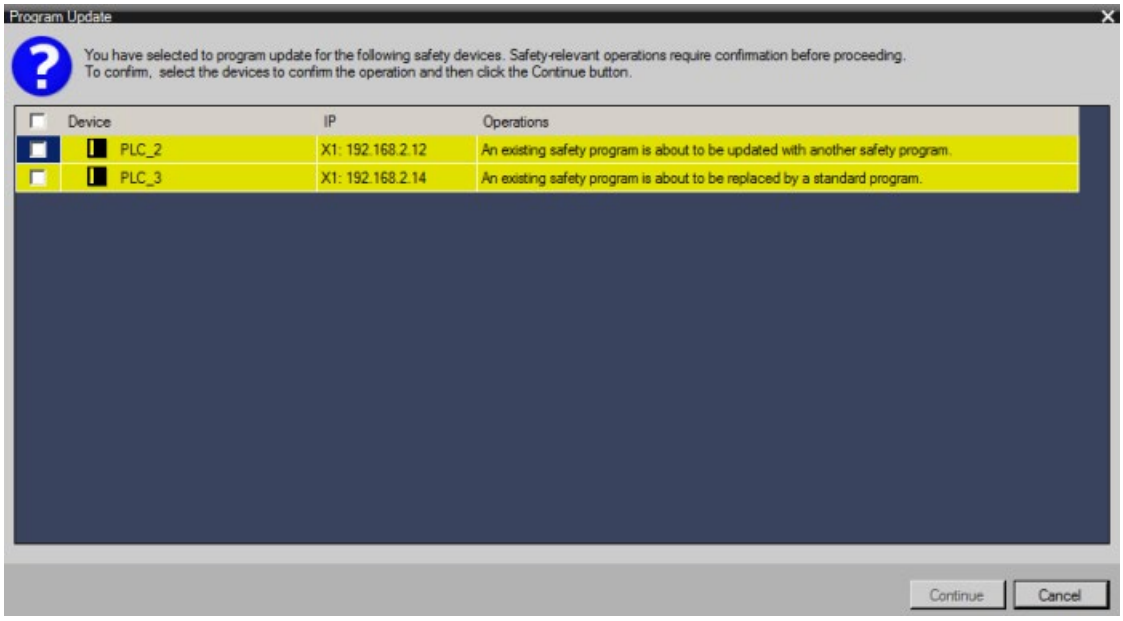
Siemens recommends that you provide a two-step process before performing a safety-relevant operation to an F-CPU.

1. The user should take an explicit action to select an F-CPU and select the operation to be performed before continuing to step 2.
2. Your software should present a dialog or equivalent to the user with one of the Siemens-defined confirmation messages (Page 143) for safety-relevant operations. Your application should prompt the user to confirm that they wish to continue before your code calls the safety-relevant operation to an F-CPU.

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

- Program update (Page 153)
- Restore device from backup file (Page 158)
- Reset to factory defaults (Page 156)
- Format memory card (Page 148)

For operations that are safety-relevant, provide a confirmation dialog for your users. Use the DetermineConfirmationMessage API method (Page 143) to determine the type of confirmation dialog to display. Providing an additional confirmation dialog protects users from accidentally performing an unintended safety-relevant operation. The following dialog is an example of a confirmation dialog for a program update operation:



Recommended programming practices

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

- Perform all safety-relevant operations on a single thread.
- Require entry of the safety password (Page 217)for safety-relevant operations. Verify the entered password against the CPU password. Use asterisks to hide passwords from the display when the user enters passwords.
- As described above, provide a confirmation dialog for the user to confirm safety-relevant operations on F-CPUs.
- Check the return codes of all methods. Ensure that your program logic only proceeds upon successful method returns.
- Include appropriate exception handling in your implementation. The API throws exceptions for critical internal faults that it detects when you are not using the API correctly. Be sure that your software handles any exceptions in an appropriate manner.
- For all safety-relevant operations, evaluate whether the operation succeeded. Display a message to the user for a successful operation. Display an error message upon an unsuccessful operation.
- Use hamming codes (Page 66) in your application to implement Boolean states.
- Use yellow coloring (Page 60) in the application to indicate fail-safe devices, safety programs, safety passwords, and other user-entered data.

- Prompt for confirmation for all operating mode changes (RUN/STOP).
- Refresh the user interface after each operation so that the application displays the correct device data.

Program update requirements

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

- An operation to a standard program is about to be initiated using the safety password
- An existing safety program is about to be deleted
- An existing safety program is about to be updated with another safety program
- An existing safety program is about to be replaced by a standard program
- A safety program is about to be loaded for the first time

Following a safety program update, the API automatically verifies that the F-signature of the program updated in the CPU. Check all function return values.

Restore from backup requirements

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

Certification

Note

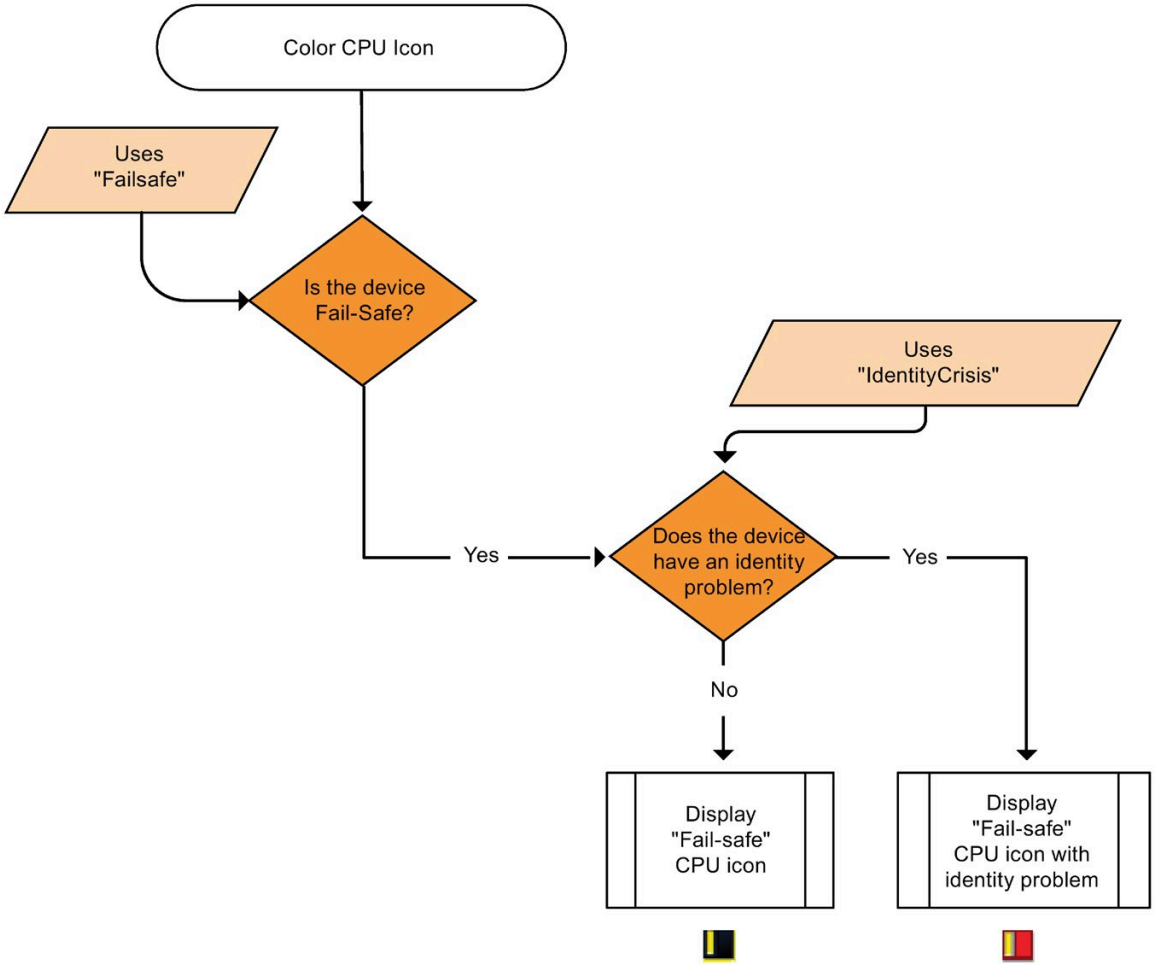
Acquire certification for your user interface application

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

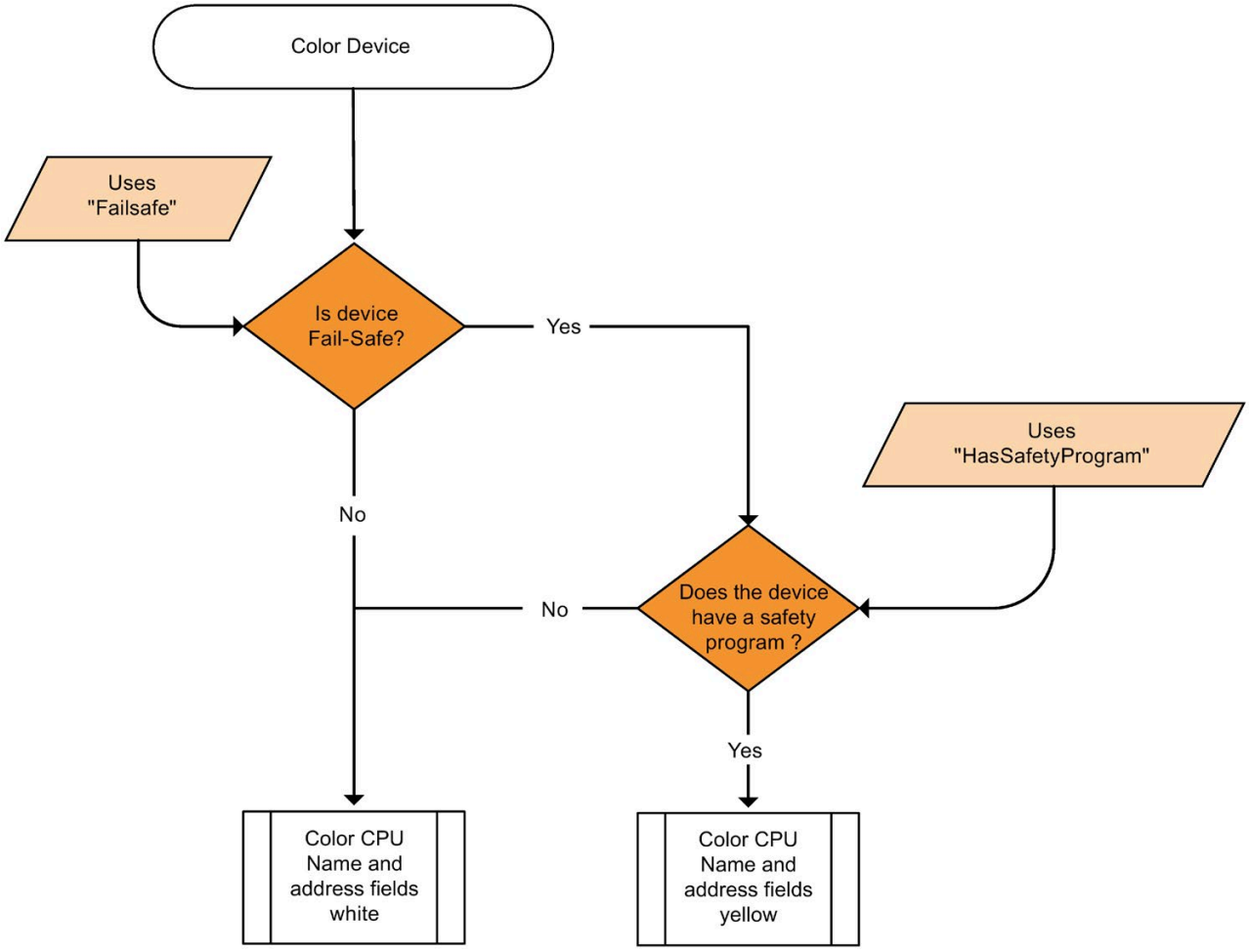
4.4.3 Color coding safety fields in your user interface

If you develop a user interface using the API, Siemens strongly recommends that you use color coding to give the user a visual indication of anything related to fail-safe CPUs and safety programs. The decision trees indicate the logic Siemens recommends in color coding typical safety-relevant fields. Consider adopting an identical or similar approach as you design your application.

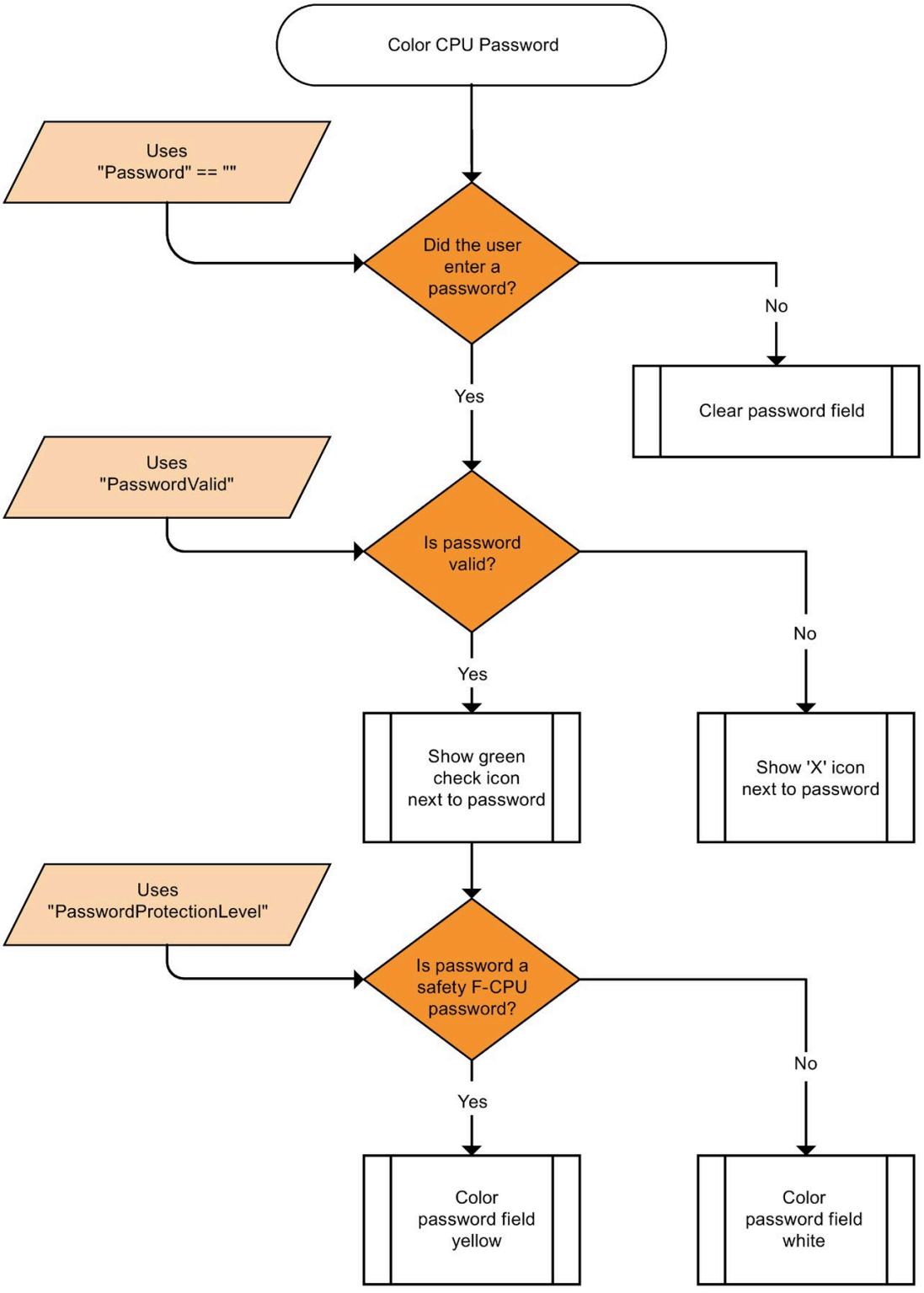
4.4.3.1 Coloring a CPU device icon



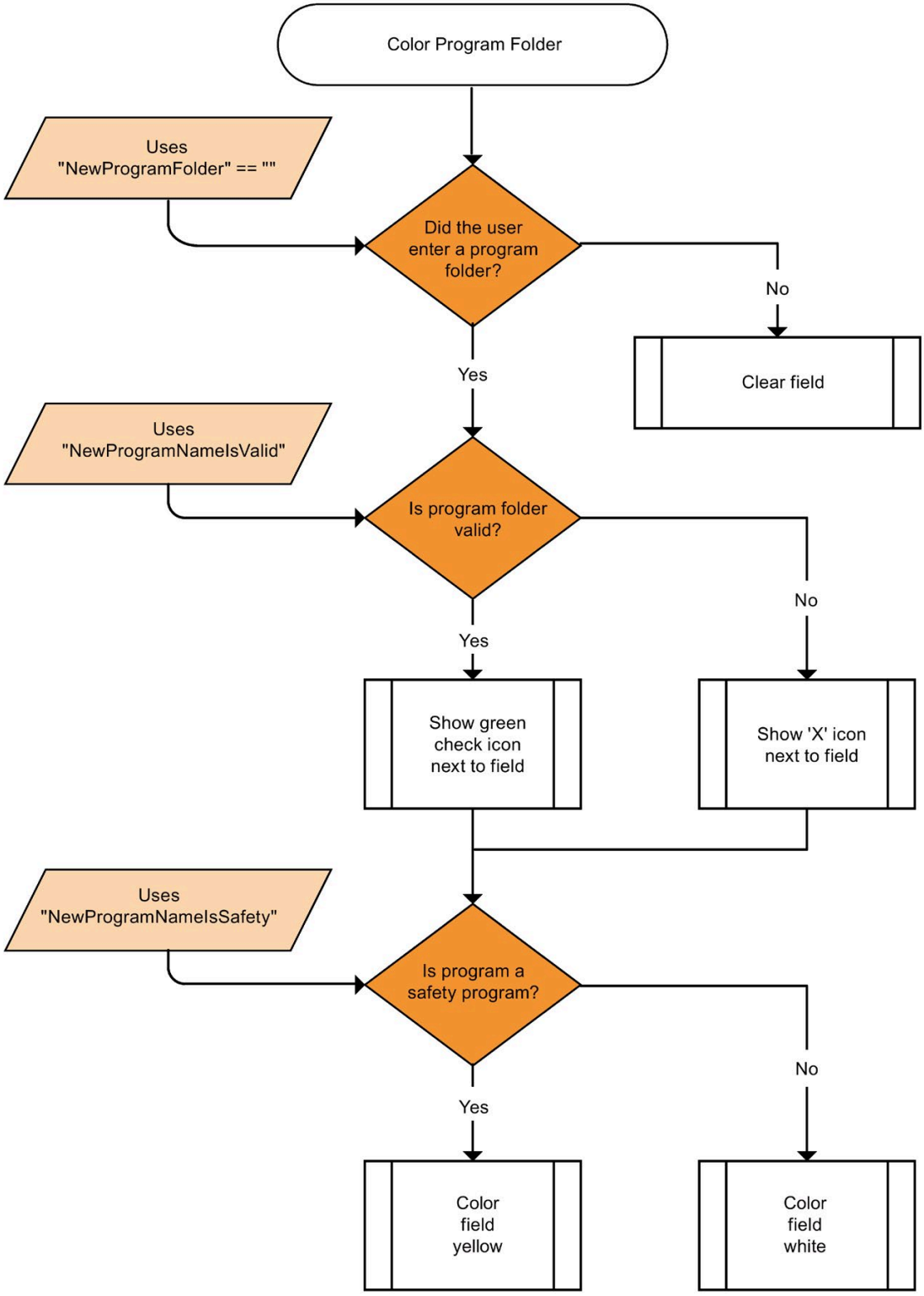
4.4.3.2 Coloring device data



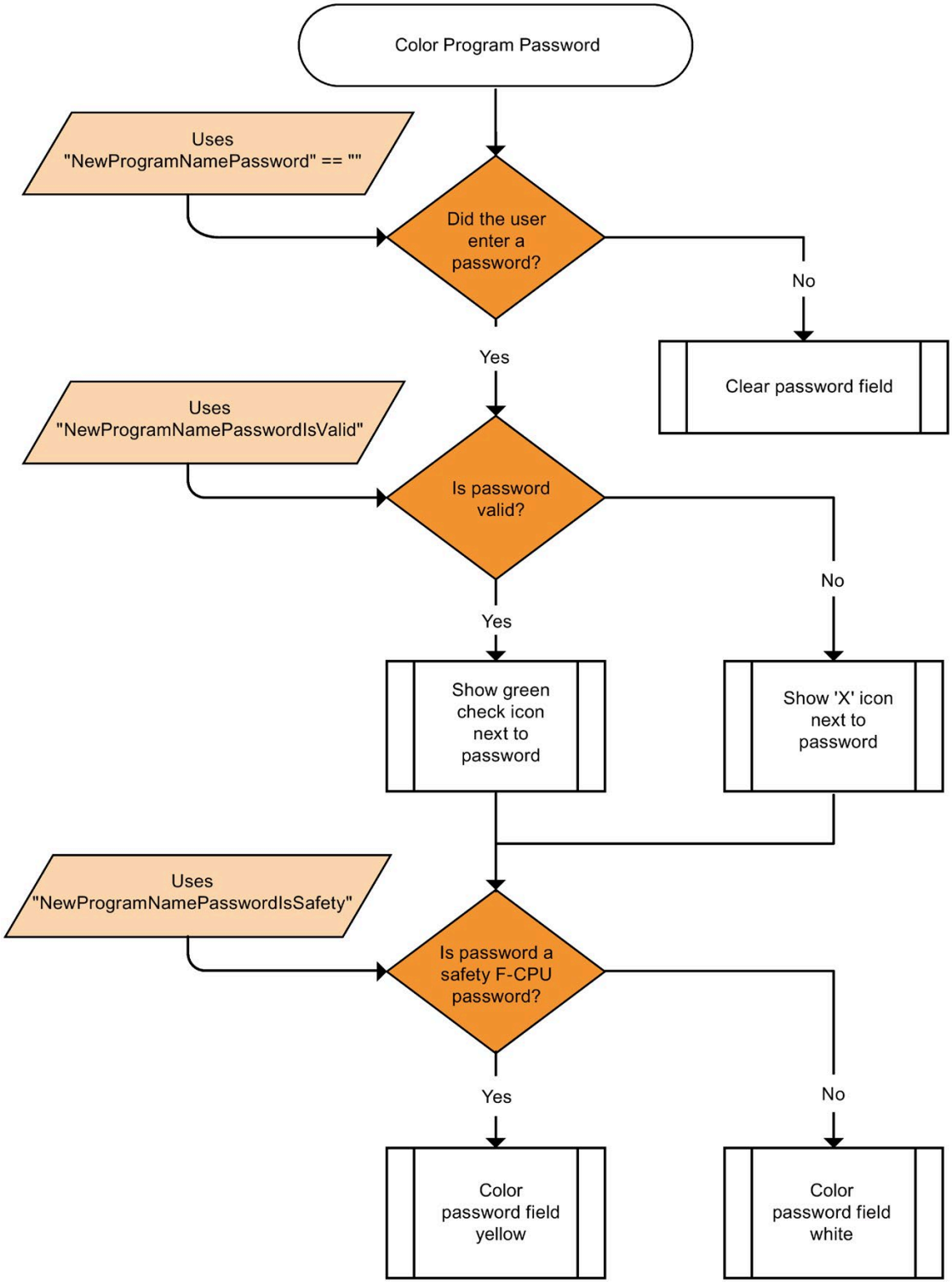
4.4.3.3 Coloring a CPU password



4.4.3.4 Coloring a program folder



4.4.3.5 Coloring a program password



4.4.4 Hamming codes

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

4.5 Common support classes

4.5.1 EncryptedString class

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

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

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

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

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

To encrypt a password to use multiple times in your code, instantiate the `EncryptedString`. You can then pass it as a parameter to one or more calls, as follows:

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

```
myCPU.SetPassword(pwd);
```

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

For example, `new EncryptedString("myPassword")` compiles the plain text "`myPassword`" into the user application. This password can be visible to others using .NET reflection.

4.5.2 Result class

The `Result` class provides information about whether a given API action succeeded. Inspect the `Result` object that API operations return to determine whether the operation succeeded or failed.

Constructor	Description
`Result()`	Creates successful result with no warnings
`Result(ErrorCode nCode)`	Creates specific error with no warnings
`Result(string strDefinedError)`	Device-defined error description
`Result(Result result)`	Creates a result with the result parameter contents

Property name	Return type	Description
`Warnings`	`ErrorCode[]{get;}`	Returns all warnings in an array of error codes
`Error`	`ErrorCode {get;}`	Returns error code
`HasWarnings`	`bool {get;}`	True when warnings exist
`Failed`	`bool {get;}`	True when the result failed
`Succeeded`	`bool {get;}`	True when the result is successful
`OMSCode`	`bool {get;}{set;}`	Gets and sets OMS code

Method Name	Return type	Description
`Clear()`	`void`	Clears error and all warnings
`SetError(ErrorCode error)`	`void`	Sets the current error
`AddWarning(ErrorCode warning)`	`void`	Adds a warning
`ChangeErrorToWarning()`	`void`	Changes current error to a warning
`GetErrorDescription(Language language)`	`string`	Gets current error string in specified language
`GetWarningDescription(Language language)`	`string[]`	Gets warning string in specified language

Example: Checking whether an operation succeeded

To check whether a given action was successful, check the `Succeeded` property:

```
ICPU devAsCpu = dev as ICPU;
if (devAsCpu != null)
{
   //-----------------------------------------------------
   // The device is a CPU.
   // You can use the ICPU interface to interact with it.
   // Set the "Selected" property to true to work with it.
   //-----------------------------------------------------


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

Example: Checking specific error codes

In other cases, it can be helpful to have more information about the failure. To inspect the specific error, use the `ErrorCode` property (Page 206), as follows:

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

If the device returns an error that is a device-specific error, the returned `ErrorCode` is `DeviceDefinedError`. The method `GetErrorDescription` provides the device-specific error string.

Example: Getting the error description in a specific language

The `Result` class also provides a language-specific error description. The `GetErrorDescription` method uses a `Language` value (Page 212)as a parameter to provide the error description in the specified language.

For example, the following code returns the error description in German:

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

Example: Checking warnings

The API has a warnings feature for information about issues that have occurred. For instance, the Refresh that is performed on the device at the end of a `ProgramUpdate` can create warnings that are not directly related to the main calling function. You can access these warnings through the `Result` class, as follows:

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

4.5.3 DiagnosticsItem class

A diagnostics item contains diagnostic information for a single event. You can read the diagnostic buffer from a CPU with the GetDiagnosticsBuffer method (Page 151).

Constructor	Description
`DiagnosticsItem()`	Creates default diagnostic item

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

4.5.4 DataChangedEventArgs class

A data changed event contains information about data that has changed within the API. See the IProfinet interface (Page 103) chapter for details.

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

Property Name	Return type	Description
`Type`	`DataChangedType`	Type of event

The `DataChangedEventArgs` class is used with the following event handler:

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

4.5.5 ProgressChangedEventArgs class

A progress changed event contains information about data that has changed within the API. See the IProfinetDevice interface (Page 103) chapter for details.

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

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

The `ProgressChangedEventArgs` class is used with the following event handler:

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

4.5.6 ExportProgressEventArgs class

An export progress changed event contains information about data that has changed within the API. Please see the section regarding The IProfinetDeviceCollection class.

Constructor	Description
ExportProgressEventArgs(int WorkItem, int MaxEntries)	Used to create and default an export progress changed event args class
int WorkItem	Index of current item being processed
int MaxEntries	Total items to process

Property Name	Return type	Description
`Cancel`	`bool {get;set;}`	Set to true to terminate current operation
`MaxEntries`	`int {get;}`	Maximum value
`WorkItem`	`int {get;}`	Current value

The `ExportProgressEventArgs` class is used with the following event handler:

```
public delegate void ExportProgressChangedEventHandler(object
sender, ExportProgressEventArgs e);
```

4.5.7 FileProgressChangedEventArgs class

A file progress changed event contains information about file data that has changed within the API. See the IProfinetDevice interface (Page 103) chapter for details.

Constructor	Description
`FileProgressChangedEventArgs(ProgressAction action, int nIndex, int nCount)`	Used to create and default a file progress changed event args class
`FileProgressChangedEventArgs(ProgressAction action)`	Used to create and default a progress changed event args class
`FileProgressChangedEventArgs()`	Used to create and default a progress changed event args class
`ProgressAction action`	Type of progress that has occurred
`int nIndex`	Index of current item being processed
`int nCount`	Total items to process

Property Name	Return type	Description
`Action`	`ProgressAction {get;set;}`	Action type of this event
`Cancel`	`bool {get;set;}`	Set to true to terminate current operation
`DoThisForRemainingOperations`	`bool {get;set;}`	Action done for remaining operations
`FileCount`	`uint {get;set;}`	Number of files
`FileIndex`	`int {get;set;}`	Index of current file
`FilePercentComplete`	`int {get;set;}`	Percent of action complete
`LocalFile`	`string {get;set;}`	Local file name

Property Name	Return type	Description
`RemoteFile`	`string {get;set;}`	Remote file name
`Replace`	`bool {get;set;}`	Replace file

Method Name	Return type	Description
`GetRemoteFileName Only()`	`string`	Get only the remote file name

The `FileProgressChangedEventArgs` class is used with the following event handler:

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

4.5.8 HealthCheckProgressEventArgs class

A health check progress changed event contains information about the health check operation progress. Please see the section regarding The HealthCheck Class.

Constructor	Description
`HealthCheckProgressEventArgs(string name, int workItem, int maxEntries)`	Used to create and default a health check progress event args class
`string name`	Name of the health entry
`int workItem`	Index of current health entry being processed
`int maxEntries`	Total health entries count to process

Property Name	Return Type	Description
`Name`	`string {get;set;}`	Health check item name
`Cancel`	`bool {get;set;}`	Set to true to terminate current operation
`MaxEntries`	`int {get;set}`	Maximum number of health check items
`WorkItem`	`int {get;set;}`	Index of current health check item

The HealthCheckProgressEventArgs class is used with the following event handler:

```
public delegate void HealthCheckProgressUpdateHandler(object sender,
HealthCheckProgressEventArgs e);
```

4.6 Common support interfaces

4.6.1 IRemoteFile interface

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

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

4.6.2 IRemoteFolder interface

`IRemoteFolder` represents folders used in data logs and recipes.

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

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

4.6.3 IRemoteInterface interface

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

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

4.6.4 IHardware interface

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

Method Name		Return type	Description
`SetFirmwareFile()`		`Result`	Sets the firmware file to update on this device or module.
Parameters	**Data type**	**Parameter type**	**Description**
`strFile`	`string`	`In`	Full file name of the firmware file

Method name	Return type	Description
`ValidateUserData`	`Result`	Confirms that the user data is valid.

Method Name		Return type	Description
`ValidateFirmwareFile()`		`Result`	Confirms that the user data is valid.
Parameters	**Data type**	**Parameter type**	**Description**
`strFile`	`string`	In	Full file name of the firmware file
`bValid`	`string`	Out	Return true if firmware file is valid for this device.
`strVersion`	`string`	Out	Returns the firmware version of this file.

Property name	Return type	Description
`ArticleNumber`	`string {get;}`	Article number of device or module
`Comment`	`string {get;set;}`	Comment for each device and module
`Configured`	`bool {get;}`	True if this device or module is configured
`ConfiguredVersion`	`string {get;}`	The configured version number for this device
`Description`	`string {get;}`	Description of article number of device or module
`Failsafe`	`bool {get;}`	True if device or module is failsafe

Property name	Return type	Description
FirmwareUpdateSupported	bool {get;}	True when this device or module supports firmware update
FirmwareVersion	string {get;}	Firmware version of device or module
HardwareNumber	short {get;}	Hardware revision number of device or module
ID	uint {get;}	ID of device or module
Name	string {get;}	Name of device or module
NewFirmwareFile	string {get;}	Full file path of the firmware file
NewFirmwareNameErrorCode	Result {get;}	Last error from SetFirmwareFile method
NewFirmwareNameIsValid	bool {get;}	True when the firmware file is valid for this device or module
NewFirmwareVersion	string {get;}	Version of the firmware file which is displayed in the dropdown
Selected	bool {get;set;}	Used to for external storage of the selected state
SerialNumber	string {get;}	Serial number of device or module
Slot	uint {get;}	Slot number of device or module
SlotName	string {get;}	Name of the slot of device or module
StationNumber	uint {get;}	Station number of device or module
SubSlot	uint {get;}	Sub slot number of device or module
Supported	bool {get;}	True if this device or module is supported
FWUpdateActivationFlag	bool {get;}	Firmware has been downloaded to the device but has not been activated.

4.6.5 IBaseDevice interface

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

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

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

Events	Return type	Description
`ProgressChanged`	`ProgressChangedEventArgs`	Called to monitor progress
`DataChanged`	`DataChangedEventArgs`	Called when data changes in the API
`FileProgressChanged`	`FileProgressChangedEventArgs`	Called when an SAT initiated file transfer has updated progress to report.

4.6.6 IHardwareCollection interface

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

Method Name	Return Type	Description
`GetHardwareFromID(uint hardwareID)`	`IHardware`	Finds a device or module using an ID.

Parameters			
`hardwareID`	`uint`	in	Hardware identifier number

4.6.7 IModuleCollection interface

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

Property name	Type	Description
None		

4.6.8 IScanErrorCollection class

This interface class encapsulates the logic needed to determine whether a network scan for devices succeeded using the ScanNetworkDevices call. Additionally, this class provides error information for any device on the network for which a scan was not successful by returning a ScanErrorEvent for the device. The IScanErrorCollection object returned by a ScanNetworkDevices call should always be inspected for success or failure. This interface is added to allow the user to get more device specific scan error information that is useful in diagnosing network scan issues.

Constructor	Description
`ScanErrorCollection()`	Creates successful network scan result

Property name	Return type	Description
`Count`	`Int`	Number of scan error events
`ScanErrorEvent[int]`	`IScanErrorEvent`	Returns the designated scan event in the collection of scan events Returns null for an invalid element
`Failed`	`bool {get;}`	True when the result failed
`Succeeded`	`bool {get;}`	True when the result is successful `Succeeded` = `Failed`

The following conditions result in a failed scan status:

- No SAT license present or SDK installation
- Invalid network interface
- No devices found on scan
- Operation canceled by user

Method name	Return type	Description
`GetEnumerator()`	`IEnumerator<IScanErrorEvent>`	Gets an enumerator to the scan error event collection

Method name	Return type	Description
`Clear()`	`void`	Clears all errors, warnings, and informational results.

Method name		Return type	Description
`AddDeviceLog()`		`void`	Add results to device log
Parameters			
Name	**Data type**	**Parameter type**	**Description**
`result`	`Result`	In	Result code to add to the collection
`device`	`IProfinetDevice`	In	Device information to add to the collection.

Method name		Return type	Description
`AddDeviceSuccess()`		`void`	Add results to device log as success
Parameters			
Name	**Data type**	**Parameter type**	**Description**
`result`	`Result`	In	Result code to add to the collection
`device`	`IProfinetDevice`	In	Device information to add to the collection.

Method name	Return type	Description
`AddDeviceLogInformation ()`	`void`	Add results to device log as informational.

Parameters

Name	Data type	Parameter type	Description
`result`	`Result`	In	Result code to add to the collection
`device`	`IProfinetDevice`	In	Device information to add to the collection.

Method name	Return type	Description
`AddDeviceLogWarning()`	`void`	Add results to device log as warning.

Parameters

Name	Data type	Parameter type	Description
`result`	`Result`	In	Result code to add to the collection
`device`	`IProfinetDevice`	In	Device information to add to the collection.

Method name	Return type	Description
`AddGeneralLog()`	`void`	Add results to device log as general

Parameters

Name	Data type	Parameter type	Description
`result`	`Result`	In	Result code to add to the collection
`device`	`IProfinetDevice`	In	Device information to add to the collection.

Method name	Return type	Description
`AddGeneralLogWarning()`	`void`	Add results to device log as general warning

Parameters

Name	Data type	Parameter type	Description
`result`	`Result`	In	Result code to add to the collection
`device`	`IProfinetDevice`	In	Device information to add to the collection.

Method name	Return type	Description
`AddGeneralLogInformation()`	`void`	Add results to device log as general informational.

Parameters

Name	Data type	Parameter type	Description

Method name		Return type	Description
result	Result	In	Result code to add to the collection
device	IProfinetDevice	In	Device information to add to the collection.

Method name			Return type	Description
AddCollection()			void	Add a collection to another collection
Parameters				
Name	**Data type**	**Parameter type**		**Description**
collection	IScanErrorCollection	In		Result code to add to the collection

Operator name			Return type	Description
operator +=			void	Add a collection to a collection
Parameters				
Name	**Data type**	**Parameter type**		**Description**
collection	IScanErrorCollection	In		Collection to add to the collection

Operator name			Return type	Description
operator +=			void	Add a collection to a collection
Parameters				
Name	**Data type**	**Parameter type**		**Description**
collection	Result	In		Result code to add to the collection

4.6.9 IScanErrorEvent class

This class provides information about a failed scan attempt of a device as a result of a call to the ScanNetworkDevices (Page 84) method. The content of the ScanErrorEvent provides available information about the device and specifies the error as information, warning, error, or invalid.

Constructor	Description
ScanErrorEvent()	Creates a ScanErrorEvent

Property Name	Return type	Description
Code	ErrorCode {get;}	The scan error
IP	string {get;}	The IP address of the device
MAC	string {get;}	The MAC address of the device
Name	string {get;}	The name of the device

Property Name	Return type	Description
TimeStamp	DateTime {get;}	The time stamp associated with the error
Type	ScanErrorType {get};	The type of the scan on which the error occurred

4.6.10 IHealthCheckErrorEvent interface

Interface for getting data associated with a Health Check error event.

Property Name	Return Type	Description
TimeStamp	DateTime{get;}	Array of files in this folder listed in ascending order
Type	LogType{get;}	Array of files in this folder
Description	string{get;}	Type of folder (datalog or recipe)

4.6.11 IDiagnosticBuffer interface

The IDignosticBuffer interface is an interface for acquiring diagnostic buffers

Method Name	Return Type	Description
GetDiagnosticsBuffer()	Result	Get a diagnostic buffer

Parameters			
Name	Data type	Parameter type	Description
aDiagnosticItems	List<DiagnosticsItem>	out	Collection of diagnostic items
language	Language	in	Language for the diagnostic items

Property Name	Return Type	Description
DiagBufferAllowed	bool{get;}	Diagnostic buffer acquisition is allowed
ScheduleDeviceDiagnostics	bool{get;set;}	Schedule get diagnostics operation

4.6.12 IResult interface

The `IResult` interface supports the `IResult` class.

Property name	Return type	Description
Warnings	ErrorCode[]{get;}	Returns all warnings in an array of error codes
Error	ErrorCode {get;}	Returns error code
HasWarnings	bool {get;}	True when warnings exist

Property name	Return type	Description
Failed	bool {get;}	True when the result failed
Succeeded	bool {get;}	True when the result is successful Succeeded = not Failed

Method Name	Return type	Description
GetErrorDescription(Language language)	string	Gets current error string in specified language
GetWarningDescription(Language language)	string[]	Gets warning string in specified language

4.6.13 ICertificateStore interface

The ICertificateStore interface supports use of the ICertificate Class by holding the ICertificates used in secure communications..

Method name	Return type	Description
GetEnumerator()	IEnumerator<ICertificate>	Allows enumeration through all certificates. Example: allows foreach to work.

Method name	Return type	Description
ShowDialog()	void	Show the Windows Certificate Chain dialog

Parameters			
Name	**Data type**	**Parameter type**	**Description**
handleParent	IntPtr	In (optional)	Parent Windows handle – for modal dialog

Property name	Return type	Description
Count	Int32 {get;}	Number of certificates
[]	ICertificate {get(Int32 nIndex);}	Get certificate from index
ContainsCPUGeneratedCertificate	bool {get;}	True if one or more of the certificates was generated by the CPU.
ContainsCAVerifiedChain	bool {get;}	True if all the of the certificates in the chain are a CA verified chain.

4.6.14 ICertificate interface

The ICertificate interface support certificates.

Property name	Return type	Description
`Data`	`Byte[] {get;}`	Blob of bytes containing the certificate.
`CPUGeneratedCertificate`	`bool {get;}`	True if the certificate was generated by the CPU.

4.7 Network class

4.7.1 Network constructor

The `.NET` class `Network` performs functions using a network interface card (NIC) installed on the programming device. The `Network` class is used to search for available interface cards and to select the interface card that communicates with the PROFINET network.

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

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

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

4.7.2 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 programming device listed by name

To identify the available network interface cards, use the `QueryNetworkInterfaceCards` method. The method outputs a list of strings. Each item in the list represents an available network interface identified by name.

Example: Querying the network interfaces

See the example and description in Getting started with the API (Page 45) for the procedure of querying the network interfaces on your computer or programming device.

4.7.3 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. It is one of the names returned from the `QueryNetworkInterfaceCards` (Page 83)method.

To use one of the network interfaces on your computer or programming devices to access the PROFINET network, you must "set" this interface. `SetCurrentNetworkInterface` enables your application to communicate with a specific PROFINET network and the devices on that network.

Example: Setting the network interface

See the example and description in Getting started with the API (Page 45) for the procedure of setting the network interface.

4.7.4 CurrentNetworkInterface property

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

This property returns an empty string if no network interface was selected by a previous call to the `SetCurrentNetworkInterface` (Page 83) method.

Example: Getting the current network interface

```
//------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//------------------------------------------------------------

#region Getting the current network interface
string currentInterface = myNetwork.CurrentNetworkInterface;


/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.7.5 ScanNetworkDevices method

After you set a network interface (Page 83), you can search for the devices on the PROFINET network. The `ScanNetworkDevices` method outputs an `IProfinetDeviceCollection` (Page 90). Each item in the collection represents a device connected directly to the PROFINET network. These devices can include CPUs, HMIs, and other devices.

The API provides the ability to filter the device scan. The filter identifies a set or range of IP addresses for the application to find, initialize, and gather the device data. A filtered scan can significantly reduce network scan time and provide more control to the user.

Return type	**Method name**
`IScanErrorCollection`	`ScanNetworkDevices`

Parameters			
Name	Data type	Parameter type	Description
`baseDevices`	`IProfinetDeviceCollection`	Out	A collection containing an `IProfinetDevice` element for each accessible device on the PROFINET network
`strFilter`	`string`	In (optional)	Optional IP address filter string, reduces scanned devices to the set that matches the filter.

Scanning a network with many devices can take several minutes. The Succeeded property in the returned `IScanErrorCollection` (Page 76) tells whether the scan succeeded or failed.

Note

Requirements for use of `ScanNetworkDevices` method.

You must have a valid and unexpired SAT license or an installation of the SDK to successfully use the ScanNetworkDevices method. If the SDK or a valid unexpired SAT license is not present at runtime, the ScanNetworkDevices method returns an empty collection. ScanNetworkDevices returns no device information to the calling application.

Example: Scanning the network for devices

See the example and descriptions in Getting started with the API (Page 45) for the complete procedure and required information for scanning a network.

To use a filtered scan in the Getting started with the API example, replace the existing ScanNetworkDevices call with code such as the following:

```
// Optional filter: If you want to only scan devices between
192.168.0.1 through 192.168.0.20
IScanErrorCollection scanResults = myNetwork.ScanNetworkDevices(out
scannedDevices, "192.168.0.[1-20]");

// Optional filter: If you want to only scan devices between
192.168.0.2, 192.168.0.5 and 192.168.0.10
IScanErrorCollection scanResults = myNetwork.ScanNetworkDevices(out
scannedDevices, "192.168.0.[2,5,10]");
```

4.7.6 SetCommunicationsTimeout method

You can set a time limit on S7-1200 and S7-1500 CPU communication operations that you call using the API. `SetCommunicationsTimeout` allows you to specify a time limit in seconds from 180 to 999 seconds. Any value outside of this range results in a failure of the operation.

Return type	Method name
`Result`	`SetCommunicationsTimeout`

Parameters			
Name	Data type	Parameter type	Description
`nTimeout`	`uint`	In	Specified time for operations to time out

Example: Setting the communications timeout

```
//-------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-------------------------------------------------------------

#region Setting the CPU network communications timeout

// Get and set the CPU network communications timeout
uint timeout = Network.GetCommunicationsTimeout();
if (timeout > 180)   // Set timeout to 3 minutes
{
   retVal = Network.SetCommunicationsTimeout(180);
}

/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.7.7 GetCommunicationsTimeout method

You use the `GetCommunicationsTimeout` method to retrieve the CPU network communications timeout value in seconds.

Return type	Method name
uint	GetCommunicationsTimeout

Example: Getting the communications timeout

```
//-------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-------------------------------------------------------------

#region Getting the CPU network communications timeout

// Get the CPU network communications timeout
uint timeout = Network.GetCommunicationsTimeout();


/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.7.8 GetEmptyCollection method

The `GetEmptyCollection` method returns an empty `IProfinetDeviceCollection`. You can then insert devices into the empty collection. Inserting devices into an empty collection is an alternative to scanning the network for devices.

Return type	Method name
`IProfinetDeviceCollection`	`GetEmptyCollection`

Example: Querying the network interfaces

See the example and description in Getting started with the API (Page 45) for the procedure of getting an empty collection with the `GetEmptyCollection` method and inserting devices.

4.7.9 ValidateNetworkInterface method

Call the `ValidateNetworkInterface` method to determine whether the network interface is valid. If the method cannot find the network interface, it returns the error "InvalidNetworkInterface". This error could result if the interface card has been removed or is in a failed state.

Method Name	Return Type	Description
`ValidateNetworkInterface()`	`Result`	Verify if this network interface is (still) valid on this system.

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`nTimeout`	`string`	in	Network interface to be validated

4.8 The HealthCheck class

4.8.1 HealthCheck constructor

The .NET class `HealthCheck` supports creation of the PC data file and is defined in the `SimaticAutomationToolHealthCheck.dll`.

To interact with the PROFINET network, your program instantiates an object of type `HealthCheck`, as follows:

```
HealthCheck myHealthCheck = new HealthCheck();
```

4.8.2 ExportPCData method

The `ExportPCData` method creates a HealthCheck.zip file containing data about the programming device. The returned `HealthCheckResultType` object indicates the outcome of the operation. The HealthCheck.log file in the HealthCheck.zip file contains details about the exported data or about errors that occurred.

Return type	Method name
`HealthCheckResultType`	`ExportPCData`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`filePath`	`string`	`In`	The complete file path for storing the zip file containing the exported PC data

Example: Exporting PC data

```
using Siemens.Automation.AutomationTool.HealthCheck;

#region Exporting PC data

HealthCheck myHealthCheck = new HealthCheck();
String healthCheckFilePath = @"c:\export\healthcheck.zip";
HealthCheckResultType hcResult =
myHealthCheck.ExportPCData(healthCheckFilePath);
if (hcResult == HealthCheckResultType.Success)
{
  //-------------------------------------------------------
  // Continue with operations.
  // The HealthCheck.log file in the HealthCheck.zip file
  // contains detailed information about the exported data.
  //-------------------------------------------------------
}
else
{
  //-------------------------------------------------------------
  // The action failed, had warnings, or was canceled by user.
  //
  // If the operation was not canceled, the HealthCheck.log file
  // in the HealthCheck.zip file contains information
  // about the export operation.
  //-------------------------------------------------------------
}
#endregion
```

Example: Using the ProgressChanged event to monitor the progress of a PC data export

The API provides the `ProgressChanged event` (Page 127) to monitor the progress of methods that can take a long time. `ExportPCData` is a method that can take a long time.

To use the `ProgressChanged` event to monitor the progress of `ExportPCData`, attach an event handler to the event. A change in the progress of the operation then automatically calls the event handler.

The following example shows how to use the `ProgressChanged` event to monitor the progress of the PC data export. The example code defines an event handler and attaches it to the `ProgressChanged` event. When `ExportPCData` finishes, the example code detaches the event handler from the event.

```
using Siemens.Automation.AutomationTool.HealthCheck;

#region Monitoring the progress of exporting PC data

{
HealthCheck myHealthCheck = new HealthCheck();
String healthCheckFilePath = @"c:\export\healthcheck.zip";
// Add enrollment for progress event
myHealthCheck.ProgressChanged += HealthCheck_ProgressChanged;

HealthCheckResultType hcResult =
myHealthCheck.ExportPCData(healthCheckFilePath);

// Remove enrollment for progress event
myHealthCheck.ProgressChanged -= HealthCheck_ProgressChanged;
}

void HealthCheck_ProgressChanged(object
sender,HealthCheckProgressEventArgs e)
{
   String strProgress = String.Format("Processing {0} of {1}",
e.WorkItem, e.MaxEntries);
   // Set to true to cancel to terminate progress
   e.Cancel = false;
}

/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.9 IProfinetDeviceCollection class

4.9.1 Iterating items in the collection

4.9.1.1 Iterating items in the collection

The `ScanNetworkDevices` (Page 84) method outputs an object of type `IProfinetDeviceCollection` (Page 90). This class provides the ability to iterate the items in the collection using `foreach` syntax or to access each `IProfinetDevice` (Page 103) using array syntax.

Scanning a network with many devices can take several minutes. The `Succeeded` property in the returned `IScanErrorCollection` tells whether the scan succeeded or failed.

Example: Iterating through each device in the collection

```
//-------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-------------------------------------------------------------

#region Iterating through each PROFINET device
foreach (IProfinetDevice dev in scannedDevices)
{
   //-----------------------------------------
   // The variable "dev" represents the
   // next item in the IProfinetDeviceCollection
   //-----------------------------------------
}
#endregion
```

Example: Iterating through the scanned devices as an array

```
//-------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-------------------------------------------------------------

#region Iterating through the scanned devices as an array

for (int deviceIdx = 0; deviceIdx < scannedDevices.Count;
deviceIdx++)
{
   //-------------------------------------------------
   // Each item in the collection is an IProfinetDevice.
   // The variable "dev" is the next indexed device.
   //-------------------------------------------------
   IProfinetDevice dev = scannedDevices[deviceIdx];
```

```
}
#endregion
```

4.9.1.2 FindDevicesBySerialNumber method

Use the FindDevicesBySerialNumber method to find all devices in the collection that have the same serial number.

Method name		**Return type**	**Description**
`FindDevicesBySerialNumber ()`		`IProfinetDevice[]`	Find devices in the collection that have the same serial number
Parameters			
Name	**Data type**	**Parameter type**	**Description**
`strArticleNumber`	`string`	In	Article number of devices you want to find.
`strSerialNumber`	`string`	In	Serial number of devices you want to find.
`device`	`IProfinetDevice`	Out	Returns a reference to the device that was found

4.9.1.3 GetEnumerator method

Use the `GetEnumerator` method to enumerate all `IProfinetDevices` in the `IProfinetDeviceCollection`.

Return type	**Method name**
`IEnumerator<IProfinetDevice>`	`GetEnumerator`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
None			

4.9.1.4 SortByIP method

Method Name	**Return Type**	**Description**
`SortByIP()`	`void`	Sorts the device collection in a numeric sequence based on IP address.

4.9.1.5 Count property

Return type	**Property name**
`int`	`Count`

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

4.9.1.6 [ ] property

Return type	Property name
`IProfinetDevice`	`this[int index]`

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

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

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

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

4.9.2 Filtering items in the collection

4.9.2.1 Collection items

The collection contains an item for each device on the PROFINET network. The collection can contain devices from different multiple product families, for example, S7-1200, S7-1500, and ET 200S).

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

4.9.2.2 FilterByDeviceFamily method

The `FilterByDeviceFamily` method returns a collection that includes only devices of the specified product families.

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

Parameters			
Name	Data type	Parameter type	Description
`familiesToInclude`	`List<DeviceFamily>`	In	Device family types to return in the list

First, contruct a filter of one or more device families. Pass this filter to the `FilterByDeviceFamily` method. The result is an `IProfinetDeviceCollection` that contains only the devices of the specified product families,

Note

Passing an empty `List<DeviceFamily>` returns an empty collection.

Example: Filtering the scanned devices by CPU family

This example filters the scanned devices for only S7-1200 and S7-1500 devices.

```
//------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//------------------------------------------------------------

#region Filtering devices by S7-1200 and S7-1500 devices


List<DeviceFamily> families = new List<DeviceFamily> {
DeviceFamily.CPU1200, DeviceFamily.CPU1500 };
List<IProfinetDevice> onlyPlus =
scannedDevices.FilterByDeviceFamily(families);
#endregion
```

4.9.2.3 FilterOnlyCPUs method

The API supports many operations that only apply to CPUs. The `FilterOnlyCPUs` method provides the ability to filter the collection to include only the CPUs on the scanned network.

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

This method returns an `ICPU` list. The `ICPU` interface (Page 130) provides the properties and methods for CPUs.

Example: Filtering only CPUs from the scanned devices

This example filters the scanned devices for only CPUs.

```
//-------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-------------------------------------------------------------

#region Filtering only CPUs from the scanned devices


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

4.9.3 Finding a specific device in the collection

4.9.3.1 FindDeviceByIP method

Use the `FindDeviceByIP` method to search for a specific device in the collection by its IP address.

Return type	Method name
`IProfinetDevice`	`FindDeviceByIP`

Parameters			
Name	Data type	Parameter type	Description
`ip`	`uint`	In	The IP address to search for

Example: Finding the device at a specific IP address

```
//-------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-------------------------------------------------------------

#region Finding device an an IP address
IProfinetDevice dev = scannedDevices.FindDeviceByIP(0xC0A80001); //
192.168.0.1
if (dev != null)
{
    // Found it
```

```
}
#endregion
```

4.9.3.2 FindDeviceByMAC method

Use the `FindDeviceByMAC` method to search for a specific device in the collection by its MAC address.

Return type	Method name
`IProfinetDevice`	`FindDeviceByMAC`

Parameters			
Name	Data type	Parameter type	Description
`mac`	`ulong`	In	The MAC address to search for

Example: Finding the device with a specific MAC address

```
//------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//------------------------------------------------------------

#region Finding device with a specific MAC address
IProfinetDevice dev =
scannedDevices.FindDeviceByMAC(0x112233445566);
// MAC address: 11:22:33:44:55:66
if (dev != null)
{
    // Found it
}
#endregion
```

4.9.4 Serialization of the device collection

4.9.4.1 WriteToStream method

Use the `WriteToStream` method to store the contents of the collection to an external destination such as a file.

Return type	Method name
`Result`	`WriteToStream`

Parameters			
Name	Data type	Parameter type	Description
`stream`	`Stream`	In	Destination for serialized output of the collection

Example: Writing a collection to a file

```
//-------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-------------------------------------------------------------

#region Writing the device collection to a file
FileStream f = File.Create("myDataFile.SAT");

retVal = scannedDevices.WriteToStream(f);

f.Close();


/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.9.4.2 ReadFromStream method

The `ReadFromStream` method creates a collection from a previously-created serialization file.

Return type	Method name
`Result`	`ReadFromStream`

Parameters			
Name	Data type	Parameter type	Description
`stream`	`Stream`	In	The source for de-serializing the collection

Example: Creating a collection from a file

```
//-------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-------------------------------------------------------------

#region Reading a device collection from a file
IProfinetDeviceCollection devices = Network.GetEmptyCollection();

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

retVal = devices.ReadFromStream(f);

f.Close();

/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.9.4.3 ExportDeviceInformation method

The `ExportDeviceInformation` method creates and exports a .csv file containing the current device collection data:

Return type	Method name
`Result`	`ExportDeviceInformation`

Parameters			
Name	Data type	Parameter type	Description
`ExportFilePath`	`string`	In	Destination file path for the generated export file

Example: Exporting device information

```
//-------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-------------------------------------------------------------

#region Exporting device information to a file


String exportFilePath = @"c:\export\DeviceInformation.csv";
retVal = scannedDevices.ExportDeviceInformation(exportFilePath);

/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */
```

```
#endregion
```

4.9.4.4 ExportDeviceDiagnostics method

The `ExportDeviceDiagnostics` method creates and exports a .csv file containing diagnostic data of each CPU in the current device collection. Column headers in the .csv file are in English.

Return type	Method name
`IScanErrorCollection`	`ExportDeviceDiagnostics`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`strPath`	`string`	In	Destination file path for the generated export file
`language`	`Language`	In	Language for the exported diagnostic buffer entries
`format`	`TimeFormat`	In (optional)	Format for displaying date and time for diagnostic entries

For each CPU in the collection, the returned `IScanErrorCollection` (Page 76) designates whether acquistion of the diagnostic data was successful. If a required password was not provided or if a network error occurred, then the method returns an error for the CPU. An error results in storing a null data entry for the CPU.

Example: Exporting device diagnostics

```
//----------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//----------------------------------------------------------------

#region Exporting device diagnostics to a file


//--------------------------------------
// Select all CPUs in the device list to
// include for diagnostic information
//--------------------------------------
foreach (IProfinetDevice profi in scannedDevices)
{
   if((profi.Family == DeviceFamily.CPU1200)||
      (profi.Family == DeviceFamily.CPU1500)||
      (profi.Family == DeviceFamily.CPU300) ||
      (profi.Family == DeviceFamily.CPU400))

   profi.Selected = true;
}
```

```
// With the API, the client application determines the file path and
name

String exportFilePath = @"c:\export\Device\DeviceDiagnostics.csv";

IScanErrorCollection exportResult =
   scannedDevices.ExportDeviceDiagnostics(exportFilePath,
Language.English,TimeFormat.Local);

/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

Example: Using the ProgressChanged event to monitor the progress of an export of device diagnostics

The API provides the `ProgressChanged` (Page 127) event to monitor the progress of methods that can take a long time. ExportDeviceDiagnostics is a method that can take a long time.

To use the ProgressChanged event to monitor the progress of `ExportDeviceDiagnostics`, attach an event handler to the event. A change in the progress of the operation then automatically calls the event handler.

The following example shows how to use the `ProgressChanged` event to monitor the progress of an export of device diagnostics. The example code defines an event handler and attaches it to the `ProgressChanged` event. The code then calls the `ExportDeviceDiagnostics` method, which can take a long time. When `ExportDeviceDiagnostics` finishes, the example code detaches the event handler from the event.

```
//-------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-------------------------------------------------------------

#region Monitoring the progress of exporting device diagnostics


{
// Enroll for progress event
scannedDevices.ProgressChanged += Export_ProgressChanged;

IScanErrorCollection ExportDeviceDiagnosticsErrors =
scannedDevices.ExportDeviceDiagnostics(@"C:\MyDocuments\DeviceDiagno
stics.csv", Language.English, TimeFormat.UTC);

// Remove enrollment for progress event

scannedDevices.ProgressChanged -= Export_ProgressChanged;

}
void Export_ProgressChanged(object sender,ExportProgressEventArgs e)
```

```
{
   String strProgress = String.Format("Processing {0} of {1}",
e.WorkItem, e.MaxEntries);
   // Set to true to cancel to terminate progress
   e.Cancel = false;
}

/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.9.5 Manually adding items to the collection

The API provides the following methods for inserting a device to the collection:

- InsertDeviceByIP method (Page 100)
- InsertDeviceByMAC method (Page 101)

Depending on the physical topology of the PROFINET network, devices might exist on the network that cannot respond to a DCP command, but that you can add by IP address. Additionally, you might choose to design your application for inserting devices rather than discovering devices by a network scan.

For devices behind an IP address router or NAT router, you can only insert them into the collection. A network scan does not discover devices behind an IP address router or NAT router. See the topic about devices behind routers and open port requirements (Page 220).

4.9.5.1 InsertDeviceByIP method

The `InsertDeviceByIP` method adds a device to the `IProfinetDeviceCollection` (Page 90). The method inserts the device with a specific IP address at the specified index.

Return type	**Method name**
`Result`	`InsertDeviceByIP`

Parameters			
Name	Data type	Parameter type	Description
`ip`	`uint`	In	The IP address of the device to add to the collection
`routerIP`	`uint`	In (optional)	When the device is behind a router, this is the applicable router IP address
`insertedDevice`	`IProfinetDevice`	Out	If this device was inserted, it will be a valid reference. Otherwise this value returned will be null.

Example: Inserting a device by IP address

See the Getting started with the API (Page 45) example for inserting devices into an empty collection.

The method call in the example is as follows:

```
IScanErrorCollection
insertErrorCollection = insertedDevices.InsertDeviceByIP(0xC0A80001,
out insertedDevice); // 192.168.0.1
```

4.9.5.2 InsertDeviceByMAC method

The `InsertDeviceByMAC` method adds a device to the `IProfinetDeviceCollection` (Page 90). The method inserts the device with a specific MAC address at the specified index.

Return type	Method name
`Result`	`InsertDeviceByMAC`

Parameters			
Name	Data type	Parameter type	Description
`mac`	`ulong`	In	The MAC Address of the device to add to the collection
`insertedDevice`	`IProfinetDevice`	Out	If this device was inserted, it will be a valid reference. Otherwise this value returned will be null.

Example: Inserting a device by MAC address

```
//------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//------------------------------------------------------------

#region Inserting a device by MAC address
UInt64 targetMAC = 0x112233445566; // equivalent to string
11:22:33:44:55:66
IScanErrorCollection insertErrorCollection =
insertedDevices.InsertDeviceByMAC(targetMAC, out insertedDevice);
}

/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.9.6 Copying data from a collection

4.9.6.1 CopyUserData method

Use the `CopyUserData` method to copy the user-entered data from one `IProfinetDeviceCollection` to another. You can use this method to avoid requiring your user to re-enter data.

Return type	Method name
`Result`	`CopyUserData`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`oldNetwork`	`IProfinetDeviceCollection`	In	Previous list used in the application

Example: Copying user data from one scan to another

```
//-------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-------------------------------------------------------------

#region Copying user data from a former scan to a new scan

IProfinetDeviceCollection rescannedDevices;
IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out
rescannedDevices);
retVal = rescannedDevices.CopyUserData(scannedDevices);

/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.9.7 Removing devices from the collection

4.9.7.1 Clear method

The `Clear` method clears the contents of the scanned devices.

Return type	Method name
`void`	`Clear`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
None			

4.9.7.2 Remove method

The `Remove` method deletes a specific device from the collection.

Return type	Method name
`void`	`Remove`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`device`	`IProfinetDevice`	In	Device to remove from the collection

4.10 IProfinetDevice interface

4.10.1 IProfinetDevice properties

Each item in the `IProfinetDeviceCollection` (Page 90) collection is represented by the `IProfinetDevice` interface. This interface provides access to the data and provides operations for all devices that are directly connected to the PROFINET network.

The `IProfinetDevice` interface supports the following properties that provide information about the network device. To ensure the properties return the current information, call the `RefreshStatus` (Page 119) method on the device before reading a property.

Property name	Return type	Description
`ArticleNumber`	`string {get;}`	The order number for the module. This is also known as MLFB or "article number".
`Comment`	`string {get;set;}`	This allows the user to specify a comment for the device and is used in the SIMATIC Automation Tool user interface. The comment is not relevant for API operations.
`Configured`	`bool {get;}`	True when the device has a valid configuration
`DefaultGateway`	`uint {get;}`	The default gateway address of the device, represented as an unsigned integer. The encoded gateway address uses one byte to represent each decimal value in the address. For example, the encoded value 0xC0A80001 is equivalent to the more common string representation of 192.168.0.1
`DefaultGatewaystring`	`string {get;}`	The default gateway address of the device, represented as a string in the form "xx.xx.xx.xx" Example: 192.168.0.1
`Description`	`string {get;}`	A description of the hardware item, based on the article number. This is the same description that the user would see in TIA Portal. Example: CPU 1215 DC/DC/DC
`DeviceFound`	`bool {get;}`	Was the device found on the network scan?
`DuplicateIP`	`bool {get;}`	Does the device have an IP address that is a duplicate?
`DuplicateNewIP`	`bool {get;}`	A proposed New IP address is duplicated
`DuplicateProfinetName`	`bool {get;}`	Does the device have a PROFINET Name that is a duplicate?
`DuplicateNewProfinetName`	`bool {get;}`	A proposed new PROFINET name is duplicated
`Failsafe`	`bool {get;}`	Based on its ArticleNumber, Is this a fail-safe device?
`Family`	`DeviceFamily {get;}`	What is the family of the device? For more information refer to the description of the `DeviceFamily` enum (Page 205).
`FamilyName`	`string {get;}`	What is the family name of the device?

Property name	Return type	Description
FirmwareUpdateAllowed	bool {get;}	Are firmware updates allowed for this device?
FirmwareVersion	string {get;}	Current firmware version of the device
ID	uint {get;}	The unique identifier for every device and module in the station. This is used as the unique identifier when executing a FirmwareUpdate.
IdentifyAllowed	bool {get;}	Is Identify currently allowed and permitted on this device?
IdentifySupported	bool {get;}	Does this device support Identify?
Initialized	bool {get;}	True: The device is in the initialized state.
HardwareNumber	short {get;}	Hardware version or "F-Stand" for the device. (Functional State)
IP	uint {get;}	The IP Address of the device, represented as an unsigned integer. The encoded IP Address uses one byte to represent each decimal value in the IP Address. For example, the encoded value 0xC0A80001 is equivalent to the more common string representation of "192.168.0.1" NOTE: The SIMATIC Automation Tool supports only IPv4 addresses. Ipv6 addressing is not supported.
IPString	string {get;}	The IP Address of the device, represented as a string in the form "xx.xx.xx.xx" (i.e. 192.168.0.1)
MAC	ulong {get;}	The unique MAC address 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 {get;}	The unique MAC address assigned to the device, represented as a string in the form 11:22:33:44:55:66
Modules	IModuleCollection {get;}	A collection of the modules (Page 128) plugged on the station.
Name	string {get;}	Name of the device
NewFirmwareFile	string {get;}	Location of the firmware file to be used in firmware update
NewFirmwareNameErrorCode	Result {get;}	ErrorCode attached to new Firmware name

Property name	Return type	Description
NewFirmwareNameIsValid	bool {get;}	Is the set Firmware file valid?
NewFirmwareVersion	string {get;}	This property is used in the SIMATIC Automation Tool user interface. It is not relevant for API operations.
NewDefaultGateway	string {get;}	This property is used in the SIMATIC Automation Tool user interface. It is not relevant for API operations.
NewIP	string {get;set;}	This property is used in the SIMATIC Automation Tool user interface. It is not relevant for API operations.
NewProfinetName	string{get;set;}	This property is used in the SIMATIC Automation Tool user interface. It is not relevant for API operations.
NewProgramName	string {get;set;}	This property is used in the SIMATIC Automation Tool user interface. It is not relevant for API operations.
NewRestoreName	string{get;set;}	This property is used in the SIMATIC Automation Tool user interface. It is not relevant for API operations.
NewSubnetMask	string {get;set;}	This property is used in the SIMATIC Automation Tool user interface. It is not relevant for API operations.
NotAccessiblewithDCP	string{get;}	True: DCP access is currently not possible with this device. It is behind a router
ProfinetConvertedName	string {get;}	Converted PROFINET name for the device
ProfinetName	string {get;}	PROFINET name for the device
ResetCommunicationsParametersAllowed	bool {get;}	Is "reset communications parameters" currently allowed and permitted on this device?
ResetCommunicationsParametersSupported	bool {get;}	Does this device support "reset communications parameters"?
ResetToFactoryAllowed	bool {get;}	Is ResetToFactory allowed on the device?
ResetToFactorySupported	bool {get;}	Does this device support ResetToFactory?

Property name	Return type	Description
`RouterIP`	`uint {get;}`	The IP Address of the router, if used, 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" If no router is used, this property is 0. NOTE: SIMATIC Automation Tool supports only IPv4 addresses.
`RouterIPString`	`string {get;}`	The IP Address of the device, represented as a string in the form "xx.xx.xx.xx" (for example, "192.168.0.1")
`Selected`	`bool {get;set;}`	Marks the device as selected to enable operations to be performed
`SerialNumber`	`string {get;}`	Unique serial number for the device
`SetIPAllowed`	`bool {get;}`	Is "Set IP Address" currently enabled and permitted on this device?
`SetIPSupported`	`bool {get;}`	Does this device support "Set IP address"?
`SetProfinetNameAllowed`	`bool {get;}`	Is "Set PROFINET Name" currently enabled and permitted on this device?
`SetProfinetNameSupported`	`bool {get;}`	Does this device support "Set PROFINET Name"?
`Slot`	`uint {get;}`	Slot number for the hardware item
`SlotName`	`string {get;}`	This property is used in the SIMATIC Automation Tool user interface. It is not relevant for API operations.
`StationNumber`	`uint {get;}`	Station number of the device
`SubSlot`	`uint {get;}`	The subslot of the device. This is relevant for pluggable submodules such as S7-1200 SB modules.
`Supported`	`bool {get;}`	True when the article number exists in the database and the device is supported by current SIMATIC Automation Tool API.

Property name	Return type	Description
SubnetMask	uint {get;}	The subnet mask of the device, represented as an unsigned integer. The encoded subnet mask uses one byte to represent each decimal value in the address. For example, the encoded value 0xFFFFFF00 is equivalent to the more common string representation of 255.255.255.0.
SubnetMaskString	string {get;}	The subnet mask of the device, represented as a string in the form "xx.xx.xx.xx" (i.e. 192.168.0.1)

4.10.2 IProfinetDevice methods

4.10.2.1 FirmwareUpdate method

You use the IHardware method SetFirmwareFile (Page 74)and the IProfinetDevice method FirmwareUpdate to update the firmware of the following hardware through a CPU interface:

- CPU
- Local I/O Modules
- Distributed I/O
- Displays, Signal Boards, Signal Modules, Communication Modules, and others

Return type	Method name
Result	FirmwareUpdate

Parameters			
Name	**Data type**	**Parameter type**	**Description**
hardwareID	uint	In	Hardware identifier of the module
bUpdateSameVersion	bool	In	If true, the method will proceed with the update. The update proceeds if the update file is the same version as the current firmware version of the module.
type	FirmwareUpdateType (Page 212)	In	Optional input parameter that specifies the type of firmware update operation to perform

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.

Restrictions

The API only supports the CPU network interface for a firmware update. You cannot update firmware through a CM or CP interface.

Precautions

A firmware update puts the CPU in STOP mode.

Firmware update of a CPU directly connected to the network interface

To update the firmware of a CPU, the application must perform these steps:

1. Set the `Selected` (Page 103) property of the CPU.
2. Call the `SetFirmwareFile` (Page 74) method with the appropriate UPD file for this CPU.
3. Call the `FirmwareUpdate` method on the CPU with the ID of the CPU.
4. Clear the `Selected` (Page 103) property of the CPU.

Firmware update of a directly connected CPU's local modules, displays, signal boards, signal modules

To update the firmware of a directly connected CPU's local module, the application must perform these steps:

1. Set the `Selected` (Page 103) property of the local module.
2. Call the `SetFirmwareFile` (Page 74) method with the appropriate UPD file for the local module.
3. Call the `FirmwareUpdate` method on the CPU with the ID of the local module.
4. Clear the `Selected` (Page 103) property of the local module.

Note

Updating firmware for PROFINET devices that are directly connected to the network interface

If a PROFINET device that is not a CPU is directly connected to the network interface, you can follow the same procedures and code examples to update the firmware of those devices.

Accessing distributed I/O devices

You can also update the firmware of devices that are configured as distributed I/O on a PROFINET network. You must configure the distributed I/O devices in the TIA Portal's Device Configuration. You must download the configuration to the CPU for the CPU to forward the firmware update to the appropriate device.

Restrictions

The API only supports firmware update over PROFINET. The API does not support the network options PROFIBUS and AS-i distributed I/O.

Firmware update of ET200 IM (distributed I/O) through a CPU interface

To update the firmware, the application must perform these steps:

1. Set the `Selected` (Page 103) property of the PROFINET device
2. Call the `SetFirmwareFile` (Page 74) method with the appropriate UPD file for this PROFINET device.
3. Call the `FirmwareUpdate` method on the CPU with the ID of the PROFINET device.
4. Clear the `Selected` (Page 103) property of the PROFINET device.

Firmware update of ET200 IM local modules (distributed I/O) through a CPU interface

To update the firmware, the application must perform these steps:

1. Set the `Selected` (Page 103) property of the PROFINET device local module.
2. Call the `SetFirmwareFile` (Page 74) method with the appropriate UPD file for this PROFINET device's local module.
3. Call the `FirmwareUpdate` method on the CPU with the ID of the PROFINET device's local module.
4. Clear the `Selected` (Page 103) property of the PROFINET device local module.

Example: Firmware update

```
//-----------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-----------------------------------------------------
#region Firmware Update

IProfinetDevice myDevice =
scannedDevices.FindDeviceByIP(0xC0A80001); // 192.168.0.1
if (myDevice != null)
{
   //---------------------------------------
   // Updating CPU or PROFINET Device Firmware
   //---------------------------------------
   myDevice.Selected = true;
   retVal = myDevice.SetFirmwareFile("C:\\DeviceFirmwareFile.upd");
   retVal = myDevice.FirmwareUpdate(myDevice.ID, false);
   myDevice.Selected = false;

   //---------------------------------------
   // Updating CPU Local Modules or PROFINET
   // Device Local Modules Firmware
   //---------------------------------------
   foreach (IModule module in myDevice.Modules)
   {
      module.Selected = true;
      retVal =
module.SetFirmwareFile("C:\\LocalModuleFirmwareFile.upd");
      retVal = myDevice.FirmwareUpdate(module.ID, false);
      module.Selected = false;
   }

   //-----------------------------------------------
   // Updating Distributed I/O configured in a CPU
   //-----------------------------------------------
   ICPU myCPU = myDevice as ICPU;
   if (myCPU != null)
   {
      foreach (IRemoteInterface interFace in myCPU.RemoteInterfaces)
      {
         foreach (IBaseDevice remoteDevice in interFace.Devices)
         {
            //-------------------------------
            // Updating a Distributed I/O IM
            //-------------------------------
            remoteDevice.Selected = true;
            retVal =
remoteDevice.SetFirmwareFile("C:\\RemoteDeviceFirmwareFile.upd");
            retVal = myCPU.FirmwareUpdate(remoteDevice.ID, false);
            remoteDevice.Selected = false;

            //-------------------------------------------
            // Updating a Distributed I/O IM Local Modules
            //-------------------------------------------
```

```
            foreach (IModule module in remoteDevice.Modules)
            {
               module.Selected = true;
               retVal =
module.SetFirmwareFile("C:\\RemoteModuleFirmwareFile.upd");
               retVal = myCPU.FirmwareUpdate(module.ID, false);
               module.Selected = false;
            }
         }
      }
   }
}

/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.10.2.2 FirmwareActivate method

The `FirmwareActivate` method activates the downloaded firmware update file for the specified hardware item (`hardwareID`) on the device. The `hardwareID` can specify either the device itself or a module on the same rack.

Return type	Method name
`Result`	`FirmwareActivate`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`hardwareID`	`uint32`	In	Hardware identifier of the module

Types of firmware update

The API supports the following types of firmware update:

- Typical firmware update that includes file download and activation
- Two step firmware update without activation. The file download occurs in RUN mode.
- Two step firmware update without activation. The file download occurs in STOP mode.

If a device is not a CPU, it does not have an operating mode (RUN/STOP). The second and third types are the same.

Firmware update with download and activation

The typical firmware update downloads the firmware update file to the CPU or device and immediately activates the new firmware. Before downloading the firmware update file, the `FirmwareUpdate` (Page 108) method puts all CPUs in STOP mode. Refer to the `FirmwareUpdate` (Page 108) topic for a description and example code.

Two step firmware update with download in RUN mode and no activation

You can choose to download the firmware to the device and not immediately activate it. You can choose to activate later at a time of your choosing by calling the method `FirmwareActivate`. Two Step refers to the download step + the activation step. You can choose to download the firmware update files to all of your devices that support this feature, while they are controlling your process and are subsequently in RUN mode. Then after all devices have accepted the firmware update file, you can choose to activate all the devices at the same time. The process of activation puts the CPUs in STOP mode

Two step firmware update with download in STOP mode and no activation

You can choose to download the firmware to the device and not immediately activate it. You can choose to activate later at a time of your choosing by calling the method `FirmwareActivate`. Two step refers to the download step + the activation step. You can choose to download the firmware update files to all of your devices that support this feature by placing them in STOP mode prior to the download. Then after all devices have accepted the firmware update file, you can choose to activate all the devices are the same time

Restrictions

The API only supports the CPU network interface for a firmware update. You cannot update firmware through a CM or CP interface.

Precautions

Activating firmware puts the CPU in STOP mode if it is not already in STOP mode.

Example: Two step firmware activation

```
//-----------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-----------------------------------------------------
#region Two Step Firmware Update

IProfinetDevice myDevice = scannedDevices.FindDeviceByIP(0xC0A80001)
; // 192.168.0.1
if (myDevice != null)
{
   //----------------------------------------
   // Updating CPU or PROFINET Device Firmware
   //----------------------------------------
   myDevice.Selected = true;
   retVal = myDevice.SetFirmwareFile("C:\\DeviceFirmwareFile.upd");

   // STEP 1
   retVal = myDevice.FirmwareUpdate(myDevice.ID, false, FirmwareUpda
teType.DownloadWithoutActivation);

   // STEP 2 (This can be done later, but is shown here for simplici
ty)
```

```
   retVal = myDevice.FirmwareActivate(myDevice.ID);
   myDevice.Selected = false;

   //-----------------------------------------------
   // Updating CPU Local Modules or PROFINET Device
   // Local Modules Firmware
   //-----------------------------------------------
   foreach (IModule module in myDevice.Modules)
   {
      module.Selected = true;
      retVal = module.SetFirmwareFile("C:\\LocalModuleFirmwareFile.u
pd");

      // STEP 1
      retVal = myDevice.FirmwareUpdate(module.ID, false, FirmwareUpd
ateType.DownloadWithoutActivation);

      // STEP 2 (This can be done later but is shown here for simpli
city)
      retVal = myDevice.FirmwareActivate(module.ID);
      module.Selected = false;
   }

   //-----------------------------------------------
   // Updating Distributed I/O configured in a CPU
   //-----------------------------------------------
   ICPU myCPU = myDevice as ICPU;
   if (myCPU != null)
   {
      foreach (IRemoteInterface interFace in myCPU.RemoteInterfaces)
      {
         foreach (IBaseDevice remoteDevice in interFace.Devices)
         {
            //--------------------------------
            // Updating a Distributed I/O IM
            //--------------------------------
            remoteDevice.Selected = true;
            retVal = remoteDevice.SetFirmwareFile("C:\\RemoteDeviceF
irmwareFile.upd");

            // STEP 1
            retVal = myCPU.FirmwareUpdate(remoteDevice.ID, false, Fi
rmwareUpdateType.DownloadWithoutActivation);

            // STEP 2 (This can be done later but is shown here for
simplicity)
            retVal = myCPU.FirmwareActivate(remoteDevice.ID);
            remoteDevice.Selected = false;

            //-------------------------------------------
            // Updating a Distributed I/O IM Local Modules
            //-------------------------------------------
            foreach (IModule module in remoteDevice.Modules)
            {
               module.Selected = true;
```

```
                retVal = module.SetFirmwareFile("C:\\RemoteModuleFirm
wareFile.upd");

                // STEP 1
                retVal = myCPU.FirmwareUpdate(module.ID, false, Firmw
areUpdateType.DownloadWithoutActivation);

                // STEP 2 (This can be done later but is shown here f
or simplicity)
                retVal = myCPU.FirmwareActivate(module.ID);
                module.Selected = false;
            }
        }
    }
}

/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.10.2.3 Identify method

The `Identify` method flashes a device LED or HMI screen for a specific network device. The flashing light helps identify the physical location of the device. You cannot use the `Identify` method to identify devices behind routers.

Return type	Method name
`Result`	`Identify`

Example: Identifying a device by flashing

```
//-------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-------------------------------------------------------------

#region Identifying a device by flashing LEDs or screen
IProfinetDevice myDevice = scannedDevices.FindDeviceByIP(0xC0A80001)
as IProfinetDevice;
// 192.168.0.1
if (myDevice != null)
{
   retVal = myDevice.Identify();
}

/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.10.2.4 IsFirmwareUpdateAllowed method

The `IsFirmwareUpdateAllowed` method determines whether a firmware update can be performed on the device based on the parent and child device capabilities.

Return type	Method name
`bool`	`IsFirmwareUpdateAllowed`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`hardwareID`	`uint`	In	Hardware identifier of the module

4.10.2.5 IsFirmwareUpdate2StepAllowed method

The `IsFirmwareUpdate2StepAllowed` method determines whether a two-step firmware update can be performed on the device based on the parent and child device capabilities.

Return type	Method name
`bool`	`IsFirmwareUpdate2StepAllowed`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`hardwareID`	`uint`	In	Hardware identifier of the module

4.10.2.6 IsIPAddressOnNetwork method

The `IsIPAddressOnNetwork` method determines whether the IP address uniquely exists in the `IProfinetDeviceCollection` (Page 90).

Return type	Method name
`bool`	`IsIPAddressOnNetwork`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`nIP`	`uint`	In	IP address to check

Example: Checking whether an IP address is on the collection

```
//-----------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-----------------------------------------------------------------

#region Checking whether an IP address is in the collection
//---------------------------------
// Find a device by its MAC address
//---------------------------------
IProfinetDevice dev =
scannedDevices.FindDeviceByMAC(0x112233445566);
// MAC Address 11:22:33:44:55:66
//---------------------------------
// Check whether the IP address for this
// device is in the collection
//---------------------------------

if (dev != null)

{
   if (dev.IsIPAddressOnNetwork(dev.IP)
   {
      //-------------------------------------------------------------
      // A device at the specified IP address is in the collection.
      // Continue operations.
      //-------------------------------------------------------------
   }
}
/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.10.2.7 IsPROFINETNameOnNetwork method

The `IsProfinetNameOnNetwork` method determines whether a PROFINET name uniquely exists in the `IProfinetDeviceCollection` (Page 90).

Return type	Method name
`bool`	`IsProfinetNameOnNetwork`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`strName`	`string`	In	PROFINET name to check

Example: Checking whether a PROFINET name is in the collection

```
//-----------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-----------------------------------------------------------------

#region Checking whether a PROFINET name is in the collection
//---------------------------------
// Find a device by its MAC address
//---------------------------------
IProfinetDevice dev =
scannedDevices.FindDeviceByMAC(0x112233445566);
// MAC Address 11:22:33:44:55:66
//-----------------------------------------------------------------

// Check the PROFINET name for the device is in the collection

//-----------------------------------------------------------------

if (dev != null)

{

   if (dev.IsProfinetNameOnNetwork(dev.ProfinetName))

   {

      //-------------------------------------------------------
      // A device with the Profinet name is in the collection.
      // Continue operations.
      //-------------------------------------------------------
    }

}
/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.10.2.8 IsUploadServiceDataAllowed method

Use the `IsUploadServiceDataAllowed` method to determine whether you can upload the service data from a device based on the parent and child device capabilities.

Return type	Method name
`bool`	`IsUploadServiceDataAllowed`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`hardwareID`	`uint`	In	Hardware identifier of the module

4.10.2.9 RefreshStatus method

When the the `ScanNetworkDevice` (Page 84)`s` method creates an `IProfinetDeviceCollection` (Page 90), the scan returns only a minimal amount of information about each device. To get all the available information for the device, call the `RefreshStatus` method. This method makes a connection to the device, queries for various information, and then disconnects from the device.

Method Name	**Return type**	**Description**	
RefreshStatus()	Result	Make a connection to this device on the network and read available device information.	
Parameters			
Parameter name	**Data type**	**Parameter type**	**Description**
`bAdvancedRefresh`	`bool`	In (optional)	Set memory card advanced refresh (Reserved for SIMATIC Automation Tool)

Using RefreshStatus

Objects in the `IProfinetDeviceCollection` (Page 90) that represent devices have only partial data from each device after a network scan. To obtain all the data about a device and to use the API properly, you must do the following:

1. Call `SetPassword` (Page 164) for each protected CPU. The CPU password must provide sufficient privileges to read all the device data.
2. Call the `RefreshStatus` method for each device in the `IProfinetDeviceCollection` (Page 90).

The `RefreshStatus` method updates all the data that represents the status of the device.

Critical error exceptions

If the API throws a critical error exception, then you are not using the API correctly for the status of the device. Remember to call `RefreshStatus` to keep the data about a device up to date before using additional API functions.

Example: Refreshing status following a network scan

Refer to Getting started with the API (Page 45) for an example of the RefreshStatus method in the context of a network scan with protected CPUs.

4.10.2.10 ResetCommunicationParameters method

Use the `ResetCommunicationParameters` method to reset communication parameters of a PROFINET device to its factory settings. This sets the following parameters:

- NameOfStation to "" empty string
- IP suite parameter to 0.0.0.0
- DHCP parameters (if available) to factory values

- All P Dev parameters (PD IR Data, PD Port Data Adjust, PD Interface MRP Data Adjust ...) to factory values
- Parameters adjusted by SMNP, like sysContact, sysName, and sysLocation from MIB-II to factory values

Return type	Method name
`Result`	`ResetCommunicationParameters`

Note

You cannot use this method to reset a CPU, unless you configured it as an I-Device. The `ICPU interface` (Page 130) provides a `ResetToFactoryDefaults` method (Page 156) for resetting CPUs. You also cannot reset communication parameters for devices behind routers.

Example: Resetting communication parameters for a device

```
//------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//------------------------------------------------------------

#region Resetting communication parameters for a device

    //------------------------------------------------
    // Search for the device at 192.168.0.1 and reset
    //------------------------------------------------
    IProfinetDevice dev = scannedDevices.FindDeviceByIP(0xC0A80001);
    // 192.168.0.1
    if (dev != null)
    {
         retVal = dev.ResetCommunicationParameters();
    }

/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.10.2.11 SetIP method

Use the `SetIP` method to set or modify the IP address of a device.

For this operation to be successful, the device port configuration in the STEP 7 project must be set to "IP address is set directly on the device". You cannot use the `SetIP` method to set the IP address of a device behind a router.

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

Example: Setting an IP address

```
//-------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-------------------------------------------------------------

#region Setting the IP address for a device
//--------------------------------
// Search for the device at a MAC
// address and set its IP address
//--------------------------------
IProfinetDevice dev =
scannedDevices.FindDeviceByMAC(0x112233445566);
// MAC Address 11:22:33:44:55:66
if (dev != null)
{
    retVal = dev.SetIP(0xC0A80001, 0xFFFFFF00, 0x0);
    // 192.168.0.1
}
/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

Converting addresses from string format to encoded uint format

The `SetIP` method expects the addresses to be in encoded format as shown above. You can convert the addresses from string format to encoded uint format using the following C# code:

```
string userEnteredAddress = @"192.168.0.1"; // For example

//--------------------------------
// Convert string address to uint
//--------------------------------

System.Net.IPAddress ip =
System.Net.IPAddress.Parse(userEnteredAddress);
byte[] bytes = ip.GetAddressBytes();
Array.Reverse(bytes);

uint encodedIp = BitConverter.ToUInt32(bytes, 0); // encoded IP
address available for use
```

4.10.2.12 SetProfinetName method

Use the `SetProfinetName` method to set or modify the PROFINET station name for the device.

For this operation to be successful, the device port configuration in the STEP 7 project must be set to "PROFINET name is set directly on the device". You cannot use the `SetProfinetName` method to set the PROFINET name of a device behind a router.

Return type	Method name
`Result`	`SetProfinetName`

Parameters			
Name	Data type	Parameter type	Description
strName	String	In	New name for the PROFINET station

Example: Setting the PROFINET name

```
//-------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-------------------------------------------------------------

#region Setting the PROFINET name of a device
//--------------------------------
// Search for the device at a MAC
// address and set its IP address
//--------------------------------
IProfinetDevice dev =
scannedDevices.FindDeviceByMAC(0x112233445566);
// MAC Address 11:22:33:44:55:66
if (dev != null)
{
    retVal = dev.SetProfinetName("new name");
}
/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.10.2.13 UploadServiceData method

The `UploadServiceData` method can upload the service data from a defective CPU.

Return type	Method name
`Result`	`UploadServiceData`

Parameters			
Name	Data type	Parameter type	Description
`strPath`	`string`	In	A fully-qualified path to the folder containing the program card contents
`format`	`TimeFormat`	In (optional)	Format for displaying date and time. Possible values are UTC and Local. If not provided, the format is Local.
`hardwareID`	`uint`	In (optional)	Hardware ID for the device

Example: Uploading 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:

```
//------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//------------------------------------------------------------

#region Uploading service data


string strDiagFolder = @"c:\Diagnostics";
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
if (myCPU != null)

{
   myCPU.SetPassword(new EncryptedString("ReadAccessPassword"));
   myCPU.Selected = true;
   if (myCPU.OperatingMode == OperatingState.Defective)
   {
       // Get service data with default local timestamps format
       retVal = myCPU.UploadServiceData(strDiagFolder);

       Get service data with UTC timestamps format
       retVal = myCPU.UploadServiceData(strDiagFolder,
TimeFormat.UTC);
   myCPU.Selected = false;
   }
}
```

```
/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.10.2.14 ValidateIPAddressSubnet method

Use the `ValidateIPAddressSubnet` method to validate that a combination of IP address and Subnet mask is compatible.

Return type	Method name
`Result`	`ValidateIPAddressSubnet`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`nIP`	`uint`	In	IP Address
`nSubnetMask`	`uint`	In	Subnet Mask

Example: Validating the IP address and subnet mask for a device

```
//-------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-------------------------------------------------------------

#region Validating the IP address and subnet


//-----------------------------------------------
// Search for the device at a specific MAC
// address and validate its IP address and subnet
//-----------------------------------------------
IProfinetDevice dev =
scannedDevices.FindDeviceByMAC(0x112233445566);
// MAC Address 11:22:33:44:55:66

if (dev != null)
{
   retVal = dev.ValidateIPAddressSubnet(dev.IP, dev.SubnetMask);
}


/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.10.2.15 ValidatePROFINETName method

The `ValidatePROFINETName` method validates the provided PROFINET name.

Return type	Method name
`Result`	`ValidatePROFINETName`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`strName`	`string`	In	PROFINET Name to validate

Example: Validating a PROFINET name before setting it

```
//----------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//----------------------------------------------------------------

#region Validating the IP address and subnet


//-----------------------------------------------
// Search for a device at a MAC address and
// validate that a given PROFINET name is valid
// before assigning it to the device
//-----------------------------------------------

IProfinetDevice dev =
scannedDevices.FindDeviceByMAC(0x112233445566);
// MAC Address 11:22:33:44:55:66

if (dev != null)
{
   string name = "Valid Name";
   retVal = dev.ValidatePROFINETName(name);
   if (retVal.Succeeded)
   {
      retVal = dev.SetProfinetName(name);
   }
}


/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.10.3 IProfinetDevice events

4.10.3.1 DataChanged event

The `IProfinetDevice` interface supports the `DataChanged` event.

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 can detect certain changes to the device.

In the following example, the code attaches to the `DataChanged` event for every device on the network:

```
#region Monitoring data changes
void AttachEvents(IProfinetDeviceCollection devices)
{
  foreach (IProfinetDevice dev in devices)
  {
    dev.DataChanged += new DataChangedEventHandler(Dev_DataChanged);
  }
}

void DetachEvents(IProfinetDeviceCollection devices)
{
  foreach (IProfinetDevice dev in devices)
  {
    dev.DataChanged -= new DataChangedEventHandler(Dev_DataChanged);
 }
}


void Dev_DataChanged(object sender, DataChangedEventArgs e)
{
  if (e.Type == DataChangedType.OperatingState)
  {
   // Device operating state has changed. Respond to changes here.
  }
}


#endregion
```

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 receives a `DataChangedEventArgs` (Page 70) object. As shown in the example above, this class has a single property, "`Type`", of data type `DataChangedType` (Page 205).

4.10.3.2 ProgressChanged event

The `IProfinetDevice` interface supports the `ProgressChanged` event.

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, attach an event handler to the event. The event handler is called when there is a change in the progress of the operation.

The following example shows how you can monitor execution progress. This example shows a method that updates the firmware for a device on the network. This operation can take noticeable time. To monitor the progress of the action, the method defines and attaches an event handler to the `ProgressChanged` event. When the firmware update is complete, the event handler is detached from the event:

```
#region Monitoring progress
void UpdateCpuAtAddress(IProfinetDeviceCollection devices,uint
targetIPAddress, string updateFile)
{
    IProfinetDevice dev = devices.FindDeviceByIP(targetIPAddress);
    if (dev != null)
    {
        dev.ProgressChanged += new

        ProgressChangedEventHandler(Dev_ProgressChanged);
        dev.SetFirmwareFile(updateFile);
        dev.FirmwareUpdate(dev.ID, true);

        dev.ProgressChanged -= new

        ProgressChangedEventHandler(Dev_ProgressChanged);
    }
}

void Dev_ProgressChanged(object sender, ProgressChangedEventArgs e)
{
    IProfinetDevice device = sender as IProfinetDevice;
    double percent = 0;
    if (device != null)
    {
        if (e.Count != 0)
        {
            string sPercent = e.Index.ToString() + " %";
        }
    }
}
```

```
#endregion
```

The ProgressChangedEventArgs class

The `ProgressChanged` event handler receives a `ProgressChangedEventArgs` (Page 70) object.

Property Name	Return Type	Description
`Action`	`ProgressAction` (Page 213)	Description of the current action
`Cancel`	`bool`	Was the action canceled?
`Count`	`int`	Total amount of data to transfer
`ID`	`uint`	Hardware ID
`Index`	`int`	Current amount of data transferred

4.11 IModuleCollection class and module properties

4.11.1 Modules property and IModuleCollection class

The `IProfinetDevice` interface provides information about any modules such as signal modules, signal boards, CMs, and CPs that are on the station. The `Modules` property returns a collection of these modules (Page 76).

The following code shows how to access this information, given an IProfinetDevice that already exists:

```
//------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//------------------------------------------------------------

#region Collecting data from all modules


//-----------------------------------------------------
// The Modules property returns a collection of IModule
//-----------------------------------------------------
IModuleCollection modules = scannedDevices[0].Modules;
foreach (IModule mod in modules)
{
    //-----------------------------------------------------
    // Get article number for every module on the station
    //-----------------------------------------------------
    string displayArticleNum = mod.ArticleNumber;
}
```

```
/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.11.2 IModule interface

Each module on the station is represented as an `IModule` interface. This interface provides a subset of the properties available for a device.

The `IModule` interface extends the `IHardware` interface (Page 74). The `IModule` interface provides no methods. All operations on a module must be initiated at the device.
The `IModule` interface supports the following properties:

Property Name	Return Type	Description
`ArticleNumber`	`string`	The order number for the module. This is also known as MLFB or "article number".
`Comment`	`string`	This allows the user to specify a comment for the device and is used in the SIMATIC Automation Tool user interface. The comment is not relevant for API operations.
`Configured`	`bool`	True when the device has a valid configuration
`ConfiguredVersion`	`string`	Configured version description
`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 and in the Device Catalog. (for example, "CPU-1215 DC/DC/DC")
`Failsafe`	`bool`	Based on its ArticleNumber, is this a failsafe device?
`FirmwareUpdateAllowed`	`bool`	Is a firmware update possible for this device?
`FirmwareVersion`	`string`	Current firmware version of the device
`HardwareNumber`	`int`	Hardware identification number
`ID`	`uint`	The unique identifier for every device and module in the station. This is used as the unique identifier when executing a FirmwareUpdate.
`Name`	`string`	Name of the device
`NewFirmwareNameErrorCode`	`Result`	`ErrorCode` attached to new Firmware name
`NewFirmwareNameIsValid`	`bool`	True when the firmware file is valid for this device or module
`NewFirmwareNameIsValid`	`bool`	When the firmware file is valid for this device or module.
`NewFirmwareVersion`	`string`	New firmware version
`NewFirmwareFile`	`string`	New firmware path and file name
`Selected`	`bool`	Is the device currently selected? This is the checkbox state in the GUI.
`SerialNumber`	`string`	Unique serial number for the device
`Slot`	`uint`	Slot number for the hardware item
`SlotName`	`string`	This property is used in the SIMATIC Automation Tool user interface. It is not relevant for API operations.
`StationNumber`	`uint`	Station number of the device

Property Name	Return Type	Description
`SubSlot`	`uint`	The subslot of the device. This is relevant for pluggable submodules such as S7-1200 SB modules.
`Supported`	`bool`	Does the API support this device?
`Visible`	`bool`	This property is used in the SIMATIC Automation Tool user interface. It is not relevant for API operations.

4.12 ICPU interface

4.12.1 Identifying CPU devices in an IProfinetDeviceCollection

The `ScanNetworkDevices` method (Page 84) generates an `IProfinetDeviceCollection` (Page 90). This collection contains an item for every accessible device on the network interface. These devices can include CPUs, HMIs, and other devices. The `IProfinetDevice` interface provides properties and methods that apply to all categories of devices.The `ICPU` interface inherits from `IProfinetDevice` and therefore supports all the `IProfinetDevice` properties and methods (Page 103).

To determine whether a given `IProfinetDevice` interface actually represents a CPU device, cast it to an `ICPU`. If this cast is successful, then the network device is a CPU, and you can use the properties/methods on the `ICPU` interface.

Note

You must set the `Selected` (Page 103) flag for any operation on the `ICPU`.

You must set `SelectedConfirmed` (Page 131) if the operation on the `ICPU` is a safety-relevant operation (Page 56).

Example: Determining whether a PROFINET device is a CPU

```
//------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//------------------------------------------------------------

#region Determining whether a device is a CPU
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
if (myCPU != null)
{
   //-----------------------------------------------------
   // The device is a CPU.
   // You can use the ICPU interface to interact with it.
   //-----------------------------------------------------
}
#endregion
```

4.12.2 ICPU properties

The `ICPU` interface extends `IProfinetDevice` (Page 103) by adding the following properties. To ensure they will return the current information, your code should first call the `RefreshStatus` (Page 119) method.

Property name	Return type	Description
`CertificateStore`	`ICertificateStore`	Contains information about certificates and whether they are being used
`ConfigurationProtectionData`	`ConfigurationDataProtection`	Enum that represents the current configuration data protection state
`ConnectedToCPCM`	`bool`	This CPU is connected through a CP or CM
`CPUProtectionLevel`	`ProtectionLevel`	The protection level that is set on the CPU, independent of the password
`DataLogFolder`	`IRemoteFolder`	Information about any Data Logs found on the SIMATIC Memory Card of the CPU
`FileSystemFolder`	`ICardFolder`	This property is used in the SIMATIC Automation Tool user interface. It is not relevant for API operations.
`IdentityCrisis`	`bool`	True when the identity of the device cannot be determined
`Initialized`	`bool`	True when the device or module has a valid configuration
`InterfaceNumber`	`uint`	Interface through which the device is connected
`LastRefreshSuccessful`	`bool`	True when the last call to `RefreshStatus` completed successfully
`OperatingMode`	`OperatingState`	Designates the current mode of the CPU. This value is read-only.
`Password`	`EncryptedString`	CPU Password used in functions performed on the device
`PasswordProtectionLevel`	`ProtectionLevel`	Protection level of a legitimized CPU password
`PasswordValid`	`bool`	Is the call to `SetPassword()` valid?
`Protected`	`bool`	Is the CPU currently protected? This means a password is required to access some or all features depending on access level (Page 217).
`RedundantCPU`	`bool`	True if this CPU is a redundant CPU. Might not be configured as a redundant CPU.
`RecipeFolder`	`IRemoteFolder`	Information about any Recipes found on the SIMATIC Memory Card of the CPU
`RemoteInterfaces`	`List<IRemoteInterface>`	A list of any remote I/O interfaces (Page 173) configured for the CPU. The usage of this property is described in a later section of this document.
`SDCard`	`SDCardType`	Gets the SDCard type of the CPU, Read, Write, Unknown, or None
`SelectedConfirmed`	`bool`	Methods that perform safety-relevant operations must set the `SelectedConfirmed` Flag to TRUE, when the user reselects one or more devices from a confirmation dialog for the operation and confirms the operation. `SelectedConfirmed` means that the operation is selected and confirmed.
`SiblingSerialNumber`	`String`	The serial number of a redundant CPU's sibling. Otherwise empty.

Property name	Return type	Description
TIAPVersion	String	TIA Portal version associated with a CPU project
TrustCertificatesStore	TrustCertificateType	Enum used to represent the current trusting of communications.

4.12.3 ICPU flags

4.12.3.1 Program Update flags

To successfully perform safety-relevant functions on a device, more information is needed from the device. The following flags ensure that the Program Update function can be performed on a safety device correctly and securely.

Property name	Return type	Description
HasSafetyProgram	bool	Boolean value set if the device has a safety program present on the device. This is determined when connecting to a CPU.
NewProgramFolder	string	What is the folder location for the new program? Value is set through the SetProgramFolder (Page 166) method.
NewProgramName	string	What is the name of the new program?
NewProgramNameErrorCode	Result	Accessible way to find issues that may be present in validating the new program, such as if the program is invalid for the device or if the IP found in the program already exists on the network
NewProgramNameFSignature	uint	What is the FSignature of the new project? Used in the comparison process to determine if ProgramUpdate (Page 153) finished successfully
NewProgramNameGateway	uint	Gateway of the device in the new program
NewProgramNameHasSafetyPassword	bool	True when the method SetProgramFolder (Page 166) is called with a valid safety program that has a safety password (Page 217).
NewProgramNameIP	uint	IP address that is stored in the new program
NewProgramNameIsSafety	bool	True when the method SetProgramFolder (Page 166) is called with a valid a safety program
NewProgramNameIsValid	bool	True when the method SetProgramFolder (Page 166) is called with a valid program
NewProgramNamePassword	EncryptedString	CPU Password used to attempt a connection after ProgramUpdate (Page 153) has finished. Value is set through the use of SetProgramPassword(EncryptedString) (Page 167).
NewProgramNamePasswordErrorCode	Result	Stores the error code of the last call to SetProgramPassword (Page 167)

Property name	Return type	Description
NewProgramNamePasswordIsSafety	bool	True when the method SetProgramPassword (Page 167) is called with a valid password and the password for the new program is the safety password (Page 217)
NewProgramNamePasswordIsValid	bool	True when the method SetProgramPassword (Page 167) is called with a valid password
NewProgramNamePasswordLevel	ProtectionLevel	What is the protection level (Page 217) of the CPU password for the new program?
NewProgramNamePasswordPresent	bool	True when the method SetProgramFolder (Page 166) is called and the program is protected
NewProgramNameSubnetMask	uint	Subnet mask of the device in the new program
ProgramUpdateSucceeded	bool	True when the ProgramUpdate (Page 153) method succeeds. The program update can still return an error.

4.12.3.2 Restore flags

The API includes the following flags so that the Restore (Page 158) method can be performed on a safety device correctly and securely. A backup file does not contain all Program Update information because a backup file differs in content from a program file.

Property name	Return type	Description
NewRestoreFile	string	What is the file location for the new program? Value is set through the SetBackupFile (Page 161) method
NewRestoreName	string	What is the name of the new program?
NewRestoreNameErrorCode	Result	Accessible way to find issues that may be present in validating the new program, such as if the program is invalid or incompatible with the device
NewRestoreNameFSignature	uint	What is the FSignature of the new project? Used in the comparison process to determine if Restore (Page 158) finished successfully
NewRestoreNameIsSafety	bool	True when the method SetBackupFile (Page 161) is called and the restore file is a safety program
NewRestoreNameIsValid	bool	True when the method SetBackupFile (Page 161) is called and the restore file is valid
NewRestoreNamePassword	EncryptedString	CPU Password used to attempt connection after Restore (Page 158) has finished. Value is set through the use of SetBackupFilePassword(EncryptedString) (Page 161).
NewRestoreNamePasswordIsSafety	bool	True when the call SetBackupFilePassword (Page 161) contains a valid safety password (Page 217)

Property name	Return type	Description
NewRestoreNamePasswordIsValid	bool	True when the call SetBackupFilePassword (Page 161) contains a valid password
RestoreNamePasswordIsSafety	bool	True when the call SetBackupFilePassword (Page 161) contains a valid safety password (Page 217)
RestoreNamePasswordIsValid	bool	True when the call SetBackupFilePassword (Page 161) contains a valid password.
RestoreSucceeded	bool	Did the Restore (Page 158) operation succeed?

4.12.3.3 ICPU supported and allowed flags

The SIMATIC Automation Tool includes the feature flags in the ICPU interface and IHMI interface. The return type of these flags is bool.

Property name	Return type	Description
BackupAllowed	bool	TRUE if this device currently allows backups
BackupSupported	bool	TRUE if this device supports backups
ChangeModeAllowed	bool	TRUE if this device currently allows cpu run mode change
ChangeModeSupported	bool	TRUE if this device supports cpu run mode change
FormatMCSAllowed	bool	TRUE if this device currently allows MCS Format
FormatMCSupported	bool	TRUE if this device supports MCS Format
MemoryResetAllowed	bool	TRUE if this device currently allows memory reset
MemoryResetSupported	bool	TRUE if this device supports memory reset
PasswordAllowed	bool	TRUE if this device currently allows passwords
PasswordSupported	bool	TRUE if this device supports passwords
ProgramUpdateAllowed	bool	TRUE if this device currently allows program updates
ProgramUpdateSupported	bool	TRUE if this device supports program updates
RemoteDataLogsAllowed	bool	TRUE if this device currently allows data log reads
RemoteDataLogsSupported	bool	TRUE if this device supports data log reads
RemoteRecipesAllowed	bool	TRUE if this device currently allows recipe access
RemoteRecipesSupported	bool	TRUE if this device supports recipe access
RestoreAllowed	bool	TRUE if this device currently allows restore
RestoreSupported	bool	TRUE if this device supports restore
SecuritySupported	bool	This device supports TLS security and configuration data protection
SecurityAllowed	bool	This device currently allows TLS security and configuration data protection.
ServiceDataAllowed	bool	TRUE if this device currently allows service data reads
ServiceDataSupported	bool	TRUE if this device supports service data reads

Property name	Return type	Description
SetTimeAllowed	bool	TRUE if this device currently allows setting system time
SetTimeSupported	bool	TRUE if this device supports setting system time

4.12.4 ICPU methods

4.12.4.1 Backup method (ICPU interface)

Use the Backup method to back up the data in a CPU. The CPU creates the backup file when you call the Backup method. The Backup method stores the resulting backup file at the location you provide in the `strFile` input parameter. This backup file is available for a `Restore` (Page 158) operation. Backup files contain the entire CPU program (hardware configuration + software). You cannot back up and restore a partial program.

Return type	Method name
Result	Backup

Parameters			
Name	Data type	Parameter type	Description
strFile	string	In	A fully-qualified path and filename where the backup should be stored

Restrictions

The API only supports the CPU network interface for backing up a CPU. You cannot backup a CPU through a CM or CP interface.

Backing up a CPU

To back up a CPU to a file, the application must perform these steps:

1. Select the CPU by setting the `Selected` (Page 131) property.
2. Call `Backup`.
3. Clear the `Selected` (Page 131) property.

Example: Backing up a CPU

```
//------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//------------------------------------------------------------

#region Backing up a CPU
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
```

```
// 192.168.0.1
if (myCPU != null)
{
   myCPU.Selected = true;

   retVal = myCPU.Backup("C:\\MyBackup.s7pbkp");

   myCPU.Selected = false;
}

/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.12.4.2 CreateConfigurationDataProtectionCard method

Use the CreateConfigurationDataProtectionCard method to transfer configuration data protection password onto a memory card that can then be inserted into the CPU. Power cycling the CPU will then transfer the configuration data protection password to the CPU.

Method Name		**Return Type**	**Description**
`CreateConfigurationDataProtectionCard ()`		`Result`	Create configuration data protection card.
Parameters			
Name	**Data type**	**Parameter type**	**Description**
`password`	`EncryptedString`	In	Password
`strDestinationDrive`	`String`	In	Drive letter path of memory card (“E:”)

Example: Create configuration data protection card

```
//-------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-------------------------------------------------------------
#region Create configuration data protection card
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
if (myCPU != null)
{
   String strDrive = @"E:"; EncryptedString password = new
EncryptedString("password");

   // Create configuration data protection card
   retVal = myCPU.CreateConfigurationDataProtectionCard(password,
strDrive);
}
/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.12.4.3 CreateFirmwareUpdateCard method

The CreateFirmwareUpdateCard method is used to transfer one or more firmware upd files onto a memory card that can then be inserted into the CPU. Power cycling the CPU will then perform a firmware update

Method Name		**Return Type**	**Description**
`CreateFirmwareUpdateCard()`		`Result`	Create firmware update card from array of firmware files.
Parameters			
Name	**Data type**	**Parameter type**	**Description**
`aFirmwareFiles`	`String[]`	in	Array of firmware upd files with full path.
`strDestinationDrive`	`String`	in	Drive letter path of memory card (“E:”)

Example: Create firmware update card

```
//-------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-------------------------------------------------------------
#region Create firmware update card
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
if (myCPU != null)
{
   String[] aFirmwareFiles = new String[2] { @"C:\Firmware1.upd",
@"C:\Firmware2.upd" };
   String strDrive = @"E:";
   // Create firmware update card
   retVal = myCPU.CreateFirmwareUpdateCard(aFirmwareFiles,
strDrive);
}

/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */
#endregion
```

4.12.4.4 CreateProgramUpdateCard method

The CreateProgramUpdateCard method is used to transfer a cCPU program onto a memory card that can then be inserted into the CPU

Method Name		**Return type**	**Description**
`CreateProgramUpdateCard()`		`Result`	Create program update card.
Parameters			
Name	**Data type**	**Parameter type**	**Description**
`strProgramFolder`	`String`	in	Folder path to a valid CPU program card project.
`strDestinationDrive`	`String`	in	Drive letter path of memory card (“E:”)

Example: Create program update card

```
//-------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-------------------------------------------------------------
#region Create program update card
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
if (myCPU != null)
{
   String strProgramPath = @"C:\MyProgram"; // Memory card image
created using TIA Portal
   String strDrive = @"E:";


   // Create program update card
   retVal = myCPU.CreateProgramUpdateCard(strProgramPath, strDrive);
}
/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */
#endregion
```

4.12.4.5 DeleteConfigurationDataProtectionPassword method

Use the DeleteConfigurationDataProtectionPassword method to the configuration data protection password in the CPU.

Method Name	Return type	Description
`DeleteConfigurationDataProtectionPassword()`	`Result`	Delete the configuration data protection password in the CPU.

Example: Delete configuration data protection password

```
//-------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-------------------------------------------------------------
#region
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
if (myCPU != null)
{
   //-------------------------------------------------
   // Check that the CPU supports security
   //-------------------------------------------------
   if (myCPU.SecurityAllowed)
   {
      //-------------------------------------------------
      // Delete configuration data protection password
      //-------------------------------------------------
      retVal = myCPU.DeleteConfigurationDataProtectionPassword();
   }
}

/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.12.4.6 DeleteDataLog method

The `DeleteDataLog` method is used to delete a Data Log file from a CPU's memory card.

Return type	Method name
`Result`	`DeleteDataLog`

Parameters			
Name	Data type	Parameter type	Description
`strFileName`	`string`	In	Filename of Data Log file to delete from a CPU memory card

Example: Deleting data logs

```
//-------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-------------------------------------------------------------
#region Deleting data logs
```

```
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1                    if (myCPU != null)
{
   //-----------------------------------------------
   // Check that the CPU supports remote data logs (Page 134)   //-----
-----------------------------------------

   if (myCPU.RemoteDataLogsAllowed)
   {
      //-------------------------------------
      // Check that data logs are available
      // on the memory card
      //-------------------------------------
      if (myCPU.DataLogFolder.Exists)
      {
         //--------------------------------
         // Search for all data log files
         //--------------------------------
         foreach (IRemoteFile datalog in myCPU.DataLogFolder.Files)
         {
            datalog.Selected = true;
            //---------------------
            // Delete the data log
            //---------------------
            retVal = myCPU.DeleteDataLog(datalog.Name);
            datalog.Selected = false;
         }
      }
   }
}

/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

See also

ICPU properties (Page 131)

4.12.4.7 DeleteRecipe method

Use the `DeleteRecipe` method to delete a recipe file from a CPU's memory card.

Return type	Method name
`Result`	`DeleteRecipe`

Parameters			
Name	Data type	Parameter type	Description
`strFileName`	`string`	In	Filename of Recipe file to delete from a CPU memory card

Example: Deleting a recipe

```
//---------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//---------------------------------------------------------------
#region Deleting recipes

ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1

if (myCPU != null)
{
   //-----------------------------------------
   // Check that the CPU supports remote recipes (Page 134)
   //-----------------------------------------

   if (myCPU.RemoteRecipesAllowed)
   {
      //-----------------------------------------------------
      // Check that recipes are available on the memory card
      //-----------------------------------------------------
      if (myCPU.RecipeFolder.Exists)
      {
         //------------------------
         // Search for all recipes
         //------------------------
         foreach (IRemoteFile recipe in myCPU.RecipeFolder.Files)
         {
            recipe.Selected = true;
            //--------------------
            // Delete the recipe.
            //--------------------
            retVal = myCPU.DeleteRecipe(recipe.Name);
            recipe.Selected = false;
         }
      }
   }
}
```

```
#endregion
```

4.12.4.8 DetermineConfirmationMessage

Use the DetermineConfirmationMessage method to determine the safety action to display to the user before calling a safety-relevant operation on an F-CPU. Refer to User interface programming guidelines for safety-relevant operations (Page 58) before authoring any code that operates on an F-CPU.

Return type	Method name
`ConfirmationType`	`DetermineConfirmationMessage`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`operation`	`FailsafeOperation`	In	Operation to evaluate

Restrictions

The API contains a series of defensive coding checks. These checks are present to be sure you are using the API correctly for fail-safe CPUs. If you encounter a critical error that throws an exception, you are not using the API correctly. The code examples show a specific calling order of the API methods. Follow this order to use the API properly.

Preconditions for `ResetToFactoryDefaults` and `FormatMemoryCard`

The safety-relevant operations `ResetToFactoryDefaults` (Page 156) and `FormatMemoryCard` (Page 148) share the same preconditions before calling DetermineConfirmationMessage:

- Do NOT call DetermineConfirmationMessage for a standard CPU or a critical error will result. The DetermineConfirmationMessage method is ONLY for an F-CPU.
- Call DetermineConfirmationMessage for an F-CPU if any of these conditions are true:
  - The CPU has a safety program.
  - The CPU is protected.

  You have no need for a confirmation message otherwise.

Preconditions for `ProgramUpdate` and `Restore`

The safety-relevant operations `ProgramUpdate` (Page 153) and `Restore` (Page 158) share the same preconditions before calling DetermineConfirmationMessage:

- Do NOT call DetermineConfirmationMessage for a standard CPU or a critical error will result. The DetermineConfirmationMessage method is ONLY for an F-CPU.
- Call DetermineConfirmationMessage for an F-CPU if any of these conditions are true:
  - The CPU has a safety program.
  - The CPU is protected.
  - The program to update or the backup program to restore is a safety program.

  You have no need for a confirmation message otherwise.

Checking the return value to determine the text to display

The DetermineConfirmationMessage method accepts an input operation parameter of type `FailsafeOperation` (Page 212), which is one of the safety-relevant operations. DetermineConfirmationMessage, when called correctly, returns a `ConfirmationType` (Page 204) enum type that you can use to know what text to show for the confirmation message. The following table shows the text to display for each `ConfirmationType` (Page 204) enumeration:

Returned ConfirmationType	Confirmation message to display
`SafetyPasswordIsBeingUsed`	An operation to a standard program is about to be initiated using the safety password.
`DeletingExistingSafetyProgram`	An existing safety program is about to be deleted.
`ReplacingExistingSafetyProgram`	An existing safety program is about to be updated with another safety program.
`ReplacingExistingSafetyProgramWithNonSafetyProgram`	An existing safety program is about to be replaced by a standard program.
`LoadingSafetyProgram`	A safety program is about to be loaded for the first time.

Display the message to the user in the confirmation message box. If you want to operate on multiple F-CPUs and do not want to present a series of message boxes, you can implement a dialog with a table that displays the confirmation messages for each F-CPU. Refer to User interface programming guidelines for safety-relevant operations (Page 58) to see an example of SIMATIC Automation Tool's confirmation dialog.

Example: DetermineConfirmationMessage

```
#region Determine Confirmation Message

if (myCPU.Failsafe == false)
{
   //-----------------------------------------------
   // Do not call this method if it is not necessary
   //-----------------------------------------------
   return;
}
```

```
//---------------
// Fail-Safe CPU
//---------------
if ((operation == FailsafeOperation.FormatMCOperation) ||
   (operation == FailsafeOperation.ResetToFactoryOperation))
{
   if ((myCPU.HasSafetyProgram == false) && (myCPU.Protected == fals
e))
   {
   //-----------------------------------------------
   // Do not call this method if it is not necessary
   //-----------------------------------------------
      return;
   }
}
else if (operation == FailsafeOperation.ProgramUpdateOperation)
{
   if ((myCPU.HasSafetyProgram == false) &&
       (myCPU.NewProgramNameIsSafety == false) &&
       (myCPU.Protected == false))
   {
   //-----------------------------------------------
   // Do not call this method if it is not necessary
   //-----------------------------------------------
      return;
   }
}
else if (operation == FailsafeOperation.RestoreOperation)
{
   if ((myCPU.HasSafetyProgram == false) &&
         (myCPU.NewRestoreNameIsSafety == false) &&
         (myCPU.Protected == false))
   {
      //-----------------------------------------------
      // Do not call this method if it is not necessary
      //-----------------------------------------------
      return;
   }
}
else
{
   //-----------------------------------------------
   // Do not call this method if it is not necessary
   //-----------------------------------------------
   return;
}

//-----------------------------------
// Obtain safety confirmation message
//-----------------------------------
String messageText = "";
ConfirmationType confirmType = myCPU.DetermineConfirmationMessage(op
```

```
eration);
switch (confirmType)
{
   case ConfirmationType.SafetyPasswordIsBeingUsed:
      messageText = "An operation to a standard program is about to
be initiated using the safety password";
      break;
   case ConfirmationType.DeletingExistingSafetyProgram:
      messageText = "An existing safety program is about to be delet
ed";
      break;
   case ConfirmationType.ReplacingExistingSafetyProgram:
      messageText = "An existing safety program is about to be updat
ed with another safety program";
      break;
   case ConfirmationType.ReplacingExistingSafetyProgramWithNonSafety
Program:
      messageText = "An existing safety program is about to be repla
ced by a standard program";
      break;
   case ConfirmationType.LoadingSafetyProgram:
      messageText = "A safety program is about to be loaded for the
first time";
      break;
}

   //------------------------------------------------------------
   // Display safety confirmation message and process user input
   //------------------------------------------------------------
   System.Windows.Forms.DialogResult result = System.Windows.Forms.M
essageBox.Show(messageText, "Title", System.Windows.Forms.MessageBox
Buttons.YesNo);
   if (result == System.Windows.Forms.DialogResult.No)
      return;

   //------------------------------------------------------
   // Set SelectedConfirmed (Page 131) if the user confirmed
   // proceeding with the safety relevant operation.
   //------------------------------------------------------
myCPU.SelectedConfirmed = true;

/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

See also

CPU password access levels (Page 217)

4.12.4.9 DownloadRecipe method

Use the `DownloadRecipe` method to download a recipe file from the programming device to the CPU memory card.

Return type	Method name
`Result`	`DownloadRecipe`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`strFile`	`string`	In	The complete path and filename of the recipe file to download from the programming device to the CPU memory card

Example: Downloading recipes

```
//----------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//----------------------------------------------------------------

#region Downloading recipes
string rcpFile = @"C:\NewRecipe.csv";
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
if (myCPU != null)
{
   IRemoteFolder recipes = myCPU.RecipeFolder;
   recipes.Selected = true;
   retVal = recipes.SetRemoteFile(rcpFile);
   retVal = myCPU.DownloadRecipe(rcpFile);
   recipes.Selected = false;
}

/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

Note

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

See also

ICPU supported and allowed flags (Page 134)

4.12.4.10 FormatMemoryCard method

Use the FormatMemoryCard method to format the removable SIMATIC memory card that is plugged into a CPU.

Method Name	Return type	Description
`FormatMemoryCard()`	`Result`	Format the CPU removable memory card.
`FormatMemoryCard(ResetOptions options)`	`Result`	Optional: Format the CPU removable memory card with additional options.

Restrictions

The API only supports the CPU network interface for formatting a memory card. You cannot format a memory card through a CM or CP interface.

Precautions

Formatting a memory card puts the CPU in STOP mode.

Formatting a memory card on a standard CPU

To format the memory card on a standard CPU to factory defaults, the application must perform these steps:

1. Select the CPU by setting the `Selected` (Page 103) property.
2. If the CPU is protected, call `SetPassword` (Page 164) with a password that provides write access (Page 217).
3. Call `FormatMemoryCard`.
4. Clear the `Selected` (Page 103) property.

Formatting a memory card on an F-CPU

To format the memory card on an F-CPU, the application must perform these steps:

1. Select the CPU by setting the `Selected` (Page 103) property.
2. If the CPU is protected, call `SetPassword` (Page 164) with the safety password (Page 217).

3. Determine the confirmation message (Page 143) to display and obtain user confirmation before formatting the memory card.
4. Set the `SelectedConfirmed` (Page 131) property.
5. Call `FormatMemoryCard`.
6. Clear the `Selected` (Page 103) and `SelectedConfirmed` (Page 131) properties.

Example: Formatting a memory card

```
//-------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-------------------------------------------------------------
#region Format memory card

ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
if (myCPU != null)
{
   myCPU.Selected = true;
   if (myCPU.Failsafe == false)
   {
      //--------------
      // Standard CPU
      //--------------
      if (myCPU.Protected)
      {
         retVal = myCPU.SetPassword(new EncryptedString("WriteAccess
Password"));
      }
      retVal = myCPU.FormatMemoryCard();
      myCPU.Selected = false;
   }
   else
   {
      //---------------
      // Fail-Safe CPU
      //---------------
      if (myCPU.Protected)
      {
         retVal = myCPU.SetPassword(new EncryptedString("SafetyAcces
sPassword"));
      }

      FailsafeOperation operation = FailsafeOperation.FormatMCOperat
ion;

      //-----------------------------------------------------
      // Use DetermineConfirmationMessage to obtain the
      // safety confirmation message to show the user.
      //
      // Insert code here for determining, displaying and
```

```
        // verifying the confirmation. If not confirmed, then
        // abort and do not call the safety-relevant operation
        //-----------------------------------------------------

        retVal = myCPU.FormatMemoryCard();
        //-----------------------------------------
        // Reset the confirmed flag after the safety
        // operation is complete
        //-----------------------------------------
        myCPU.Selected = false;
        myCPU.SelectedConfirmed = false;
    }
}

/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned */
/* from methods is vital to a program's overall quality  */

#endregion
```

4.12.4.11 GetCurrentDateTime method

The `GetCurrentDateTime` method gets the current date and time for the CPU.

Return type	Method name
`Result`	`GetCurrentDateTime`

Parameters			
Name	Data type	Parameter type	Description
`DateTime`	`System.DateTime`	Out	Current date and time returned from the CPU

Example: Getting the date and time from the CPU

```
//------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//------------------------------------------------------------

#region Getting CPU data and time
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
if (myCPU != null)
{
   myCPU.SetPassword(new EncryptedString("ReadAccessPassword"));
   myCPU.Selected = true;
   DateTime curTime = new DateTime();
   retVal = myCPU.GetCurrentDateTime(out curTime);
   myCPU.Selected = false;
}
```

```
/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.12.4.12 GetDiagnosticsBuffer method

The `GetDiagnosticsBuffer` method reads the current diagnostic entries from the CPU. The `GetDiagnosticsBuffer` method returns a collection of `DiagnosticsItem` (Page 69) objects.

Use the `Language enum` (Page 212) parameter to get diagnostic entries in a specific language:

Return type	Method name
`Result`	`GetDiagnosticsBuffer`

Parameters			
Name	Data type	Parameter type	Description
`DiagnosticsItems`	`List<DiagnosticsItem>`	Out	A collection of Diagnostics Items: Each item in the collection represents an entry in the diagnostics buffer.
`Language`	`Language`	In	Requested language for the diagnostics buffer entries.

Example: Getting CPU diagnostics

```
//------------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//------------------------------------------------------------------

#region Getting CPU diagnostics
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
List<DiagnosticsItem> aLogs = new List<DiagnosticsItem>();
if (myCPU != null)
{

   myCPU.SetPassword(new EncryptedString("ReadAccessPassword"));
   myCPU.Selected = true;
   retVal = myCPU.GetDiagnosticsBuffer(out aLogs, Language.English);
   if (retVal.Succeeded)
   {
       for (int idxLog = 0; idxLog < aLogs.Count; idxLog++)
       {
       //--------------------------
       //Get information time stamp
       //--------------------------
```

```
            string descrTimes = aLogs[idxLog].TimeStamp.ToString();

            //---------------------------------
            //Get information basic description
            //---------------------------------
            string descrBasic = aLogs[idxLog].Description1;

            //--------------------------------------
             //Get information detailed description
            //--------------------------------------
            string descrDetail = aLogs[idxLog].Description2;

            //----------------------------------------
            //Get information state (incoming/outgoing)
            //----------------------------------------
            string descrState = aLogs[idxLog].State.ToString();

            }
        }
        myCPU.Selected = false;
    }

    /* For simplicity the code examples above do not check for */
    /* errors. Checking for and processing errors returned     */
    /* from methods is vital to a program's overall quality    */

    #endregion
```

4.12.4.13 MemoryReset method

The `MemoryReset` method resets CPU memory.

Return type	Method name
`Result`	`MemoryReset`

Example: Resetting CPU memory

```
//-------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-------------------------------------------------------------

#region Resetting CPU memory
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
if (myCPU != null)
{


   if (myCPU.Protected)
   {
      myCPU.SetPassword(new EncryptedString("WriteAccessPassword"));
```

```
    }
    myCPU.Selected = true;
    retVal = myCPU.MemoryReset();

    myCPU.Selected = false;
}


/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.12.4.14 ProgramUpdate method

Use the ProgramUpdate method to download a new CPU program.

Return type	Method name
`Result`	`ProgramUpdate`

Restrictions

The API only supports the CPU network interface for program update. You cannot update a program through a CM or CP interface.

Precautions

Program update puts the CPU in STOP mode.

Creating a Program Update folder

Use the TIA Portal programming software to create your HMI runtime software. After your program is complete, you can use the ProgramUpdate method to download it. Within the TIA Portal, use the Card Reader function located in the TIA Portal Project tree to create a folder. This folder will contain your CPU program that the ProgramUpdate method can use. After you have created the folder, drag and drop the entire contents of the PLC into this new folder. The ProgramUpdate method only downloads entire CPU programs (hardware configuration + software). You cannot download a partial program.

Program update of a standard CPU

To update the program of a standard CPU, the application must perform these steps:

1. Select the CPU by setting the `Selected` (Page 103) property.
2. If the CPU is protected, call `SetPassword` (Page 164) with a password that provides write access (Page 217).
3. Call `SetProgramFolder` (Page 166) to select the folder that contains the new program to be downloaded to the CPU.

4. If the new program has a CPU protection level set, call `SetProgramPassword` (Page 167) to use it after the download. You must supply a write access password (Page 217) when calling `SetProgramPassword` (Page 167).
5. Call `ProgramUpdate`.
6. Clear the `Selected` (Page 103) property.

Program update of an F-CPU

To update the program of an F-CPU, the application must perform these steps:

1. Select the CPU by setting the `Selected` (Page 103) property.
2. If the CPU is protected, call `SetPassword` (Page 164) with the safety password (Page 217).
3. Call `SetProgramFolder` (Page 166) to select the folder that contains the new program to be downloaded to the CPU.
4. If the new program has a CPU protection level set, call `SetProgramPassword` (Page 167) to use it after the download. If the new program is a fail-safe program, you must supply the safety password (Page 217) when calling `SetProgramPassword` (Page 167). If the new program is not a fail-safe program, you must supply a write access password (Page 217) when calling `SetProgramPassword` (Page 167).
5. Determine the confirmation message (Page 143) to display obtain user confirmation before updating the program.
6. Set the `SelectedConfirmed` (Page 131) property.
7. Call `ProgramUpdate`.
8. Clear the `Selected` (Page 103) and `SelectedConfirmed` (Page 131) properties.

Example: Program update

```
//-----------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-----------------------------------------------------
#region Program update

ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU; // 1
92.168.0.1
if (myCPU != null)
{
   myCPU.Selected = true;
   if (myCPU.Failsafe == false)
   {
      //--------------
      // Standard CPU
      //--------------
      if (myCPU.Protected)
      {
         retVal = myCPU.SetPassword(new EncryptedString("WriteAccess
Password"));
      }
```

```
      FailsafeOperation operation = FailsafeOperation.ProgramUpdateO
peration;

      retVal = myCPU.SetProgramFolder("C:\\MyProgramFolder");
      if (retVal.Failed && retVal.Error ==
ErrorCode.ProgramPasswordNeeded)
      {
         retVal = myCPU.SetProgramPassword(new
EncryptedString("WriteAccessProgramPassword"));
      }

      retVal = myCPU.ProgramUpdate();

      myCPU.Selected = false;
   }
   else
   {
      //---------------
      // Fail-Safe CPU
      //---------------
      if (myCPU.Protected)
      {
         retVal = myCPU.SetPassword(new
EncryptedString("SafetyAccessPassword"));
      }

      retVal = myCPU.SetProgramFolder("C:\\MyProgramFolder");
      if (retVal.Failed && retVal.Error ==
ErrorCode.ProgramPasswordNeeded)
      {
         retVal = myCPU.SetProgramPassword(new
EncryptedString("SafteyAccessProgramPassword"));
      }


      //-----------------------------------------------------
      // Use DetermineConfirmationMessage to obtain the
      // safety confirmation message to show the user.
      //
      // Insert code here for determining, displaying and
      // verifying the confirmation. If not confirmed, then
      // abort and do not call the safety-relevant operation
      //-----------------------------------------------------
      //-------------------------------------
      // Display safety confirmation message
      //-------------------------------------


      retVal = myCPU.ProgramUpdate();

      //-----------------------------------------------------
      // Reset the confirmed flag after the safety operation
      // is complete
      //-----------------------------------------------------
```

```
            myCPU.Selected = false;
            myCPU.SelectedConfirmed = false;
        }
    }
    #endregion
```

4.12.4.15 ResetToFactoryDefaults method

This method resets a CPU to its factory default values.

Method Name	Return Type	Description
`ResetToFactoryDefaults()`	`Result`	Reset the CPU to its factory defaults.
`ResetToFactoryDefaults(ResetOptions options)`	`Result`	Optional: Reset the CPU to its factory defaults with optional reset options

Restrictions

The API only supports the CPU network interface for a reset to factory defaults. You cannot reset a CPU to factory defaults through a CM or CP interface.

Precautions

Reset to factory defaults puts the CPU in STOP mode.

Resetting a standard CPU to factory defaults

To reset a standard CPU to factory defaults, the application must perform these steps:

1. Select the CPU by setting the `Selected` (Page 103) property.
2. If the CPU is protected, call `SetPassword` (Page 164) with a password that provides write access (Page 217).
3. Call `ResetToFactoryDefaults`.
4. Clear the `Selected` (Page 103) property.

Resetting an F-CPU to factory defaults

To reset an F-CPU to factory defaults, the application must perform these steps:

1. Select the CPU by setting the `Selected` (Page 103) property.
2. If the CPU is protected, call `SetPassword` (Page 164) with the safety password (Page 217).
3. Determine the confirmation message (Page 143) to display and obtain user confirmation before resetting the F-CPU to factory defaults.
4. Call `ResetToFactoryDefaults`.
5. Clear the `Selected` (Page 103) and `SelectedConfirmed` (Page 131) properties.

Example: Resetting a CPU to factory defaults

```
//-------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-------------------------------------------------------------
#region Reset CPU to factory defaults
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU; // 1
92.168.0.1
if (myCPU != null)
{
   myCPU.Selected = true;
   if (myCPU.Failsafe == false)
   {
      //--------------
      // Standard CPU
      //--------------
      if (myCPU.Protected)
      {
         retVal = myCPU.SetPassword(new EncryptedString("WriteAccess
Password"));
      }
      retVal = myCPU.ResetToFactoryDefaults();

      myCPU.Selected = false;
   }
   else
   {
      //---------------
      // Fail-Safe CPU
      //---------------
      if (myCPU.Protected)
      {
         retVal = myCPU.SetPassword(new EncryptedString("SafetyAcces
sPassword"));
      }

      FailsafeOperation operation = FailsafeOperation.ResetToFactory
Operation;

      //-----------------------------------------------------
      // Use DetermineConfirmationMessage to obtain the
      // safety confirmation message to show the user.
      //
      // Insert code here for determining, displaying and
      // verifying the confirmation. If not confirmed, then
      // abort and do not call the safety-relevant operation
      //-----------------------------------------------------

      retVal = myCPU.ResetToFactoryDefaults();
```

```
            //-----------------------------------------------------
            // Reset the confirmed flag after the safety operation
            // is complete
            //-----------------------------------------------------
            myCPU.Selected = false;
            myCPU.SelectedConfirmed = false;
        }
    }

    /* For simplicity the code examples above do not check for */
    /* errors. Checking for and processing errors returned     */
    /* from methods is vital to a program's overall quality    */

    #endregion
```

4.12.4.16 Restore method (ICPU interface)

Use the Restore method to restore a backup file to the CPU.

Return type	Method name
`Result`	`Restore`

Restrictions

The API only supports the CPU network interface for restoring a backup file. You cannot restore a backup file through a CM or CP interface.

Precautions

The Restore method puts the CPU in STOP mode.

Creating a backup file

The CPU creates the backup file when you call the `Backup` (Page 135) method. The `Backup` (Page 135) method stores the resulting backup file at the location you specify. This backup file is available for a `Restore` operation. Backup files contain the entire CPU program (hardware configuration + software). You cannot back up and restore a partial program.

Restoring from a backup file of a standard CPU

To restore a standard CPU backup file, the application must perform these steps:

1. Select the CPU by setting the `Selected` (Page 103) property.
2. If the CPU is protected, call `SetPassword` (Page 164) with a password that provides write access (Page 217).
3. Call `SetBackupFile` (Page 161) to select the backup file to be downloaded to the CPU.
4. If the backup file has a CPU protection level set, then call `SetBackupFilePassword` (Page 161) to use it after download. You must supply a write access password (Page 217) when calling `SetBackupFilePassword` (Page 161).

5. Call `Restore`.
6. Clear the `Selected` (Page 103) property.

Restoring from a backup file of an F-CPU

To restore an F-CPU backup file, the application must perform these steps:

1. Select the CPU by setting the `Selected` (Page 103) property.
2. If the CPU is protected, call `SetPassword` (Page 164) with the safety password (Page 217).
3. Call `SetBackupFile` (Page 161) to select the backup file to be downloaded to the CPU.
4. If the backup file has a CPU protection level set, then call `SetBackupFilePassword` (Page 161) to use it after download. You must supply a safety password (Page 217) when calling `SetBackupFilePassword` (Page 161).
5. Determine the confirmation message (Page 143) to display and obtain user confirmation before restoring the backup file.
6. Set the `SelectedConfirmed` (Page 131) property.
7. Call `Restore`.
8. Clear the `Selected` (Page 103) and `SelectedConfirmed` (Page 131) properties.

Example: Restoring from a backup file

```
//-------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-------------------------------------------------------------

#region Restoring a CPU backup

ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU; //
192.168.0.1
if (myCPU != null)
{
   myCPU.Selected = true;

   if (myCPU.Failsafe == false)
   {
      //--------------
      // Standard CPU
      //--------------
      if (myCPU.Protected)
      {
         retVal = myCPU.SetPassword(new EncryptedString("WriteAccess
Password"));
      }
      retVal = myCPU.SetBackupFile("C:\\MyBackup.s7pbkp");

      //-----------------------------------------------
      // Set a backup file password if the CPU program
      // in the backup file is protected
```

```
      //-----------------------------------------------
      retVal = myCPU.SetBackupFilePassword(new
EncryptedString("WriteAccessBackupFilePassword"));

      retVal = myCPU.Restore();

      myCPU.Selected = false;
   }
else
   {
      //---------------
      // Fail-Safe CPU
      //---------------
      if (myCPU.Protected)
      {
         retVal = myCPU.SetPassword(new
EncryptedString("SafetyAccessPassword"));
      }

      FailsafeOperation operation =
FailsafeOperation.RestoreOperation;

      retVal = myCPU.SetBackupFile("C:\\MyBackup.s7pbkp");

      //-------------------------------------------------
      // Set a backup file password if the CPU program in
      // the backup file is protected
      //-------------------------------------------------
      retVal = myCPU.SetBackupFilePassword(new
EncryptedString("SafteyAccessBackupFilePassword"));

      //-------------------------------------------------
      // Use DetermineConfirmationMessage to obtain the
      // safety confirmation message to show the user.
      //
      // Insert code here for determining, displaying and
      // verifying the confirmation. If not confirmed, then
      // abort and do not call the safety-relevant operation
      //-------------------------------------------------

      retVal = myCPU.Restore();

      //-------------------------------------------------
      // Reset the confirmed flag after the safety operation
      // is complete
      //-------------------------------------------------
      myCPU.Selected = false;
      myCPU.SelectedConfirmed = false;
   }
}

/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned */
/* from methods is vital to a program's overall quality */

#endregion
```

4.12.4.17 SetBackupFile method

The `SetBackupFile` method sets the full path and file name for a backup file to be restored with the `Restore` (Page 158) method.

The `SetBackupFile` method sets the following `ICPU` restore flags (Page 133):

- `NewRestoreName`
- `NewRestoreFile`
- `NewRestoreNameIsValid`
- `NewRestoreNameIsSafety`
- `NewRestorenameFSignature`

Return type	Method name
Result	SetBackupFile

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`strFile`	`string`	In	Location for a backup file

Example: Setting the backup file

`SetBackupFile` is one part of the process of restoring a backup file to a CPU.

See the example and description in the `Restore method topic` (Page 158)for the correct and complete sequence of steps for restoring a backup file.

See also

Getting started with the API (Page 45)

4.12.4.18 SetBackupFilePassword method

When you restore a backup file, the application attempts to reconnect to the device. If the backup program that you restored to the CPU is-protected, use the `SetBackupFilePassword` method to set the updated CPU password. With the updated password, the application can regain access to the device.

The method sets the `NewRestoreNamePassword` flag on the `ICPU` object:

Return type	Method name
`Result`	`SetBackupFilePassword`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`password`	`EncryptedString`	In	Sets the password for the project that is passed to the CPU during a restore

Example: Setting the CPU password for the backup file

`SetBackupFilePassword` is one part of the process of restoring a backup file to a CPU.

See the example and description in the `Restore` method topic (Page 158)for the correct and complete sequence of steps for restoring a backup file.

4.12.4.19 SetConfigurationDataProtectionPassword method

Use the SetConfigurationDataProtectionPassword method to set the configuration data protection password in the CPU.

Method name		**Return type**	**Description**
`SetConfigurationDataProtectionPassword()`		`Result`	Set the configuration data protection password in the CPU.
Parameters			
Name	**Data type**	**Parameter type**	**Description**
`password`	`EncryptedString`	In	New configuration data protection password.

Example: Set configuration data protection password

```
//-----------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-----------------------------------------------------------
#region Set configuration data protection password

ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
if (myCPU != null)
{
   //-------------------------------------------------
   // Check that the CPU supports security
   //-------------------------------------------------
   if (myCPU.SecurityAllowed)
   {
      //-------------------------------------------------
      // Set configuration data protection password
      //-------------------------------------------------
      retVal = myCPU.SetConfigurationDataProtectionPassword(new
```

```
EncryptedString("password"));
   }

}
/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */
#endregion
```

4.12.4.20 SetCurrentDateTime method

The `SetCurrentDateTime` method sets the current date and 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.

Return type	Method name
`Result`	`SetCurrentDateTime`

Parameters			
Name	Data type	Parameter type	Description
`time`	`System.DateTime`	In	New value for the CPU current time

Example: Setting CPU date and time

```
//-------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-------------------------------------------------------------

#region Setting CPU date and time
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
if (myCPU != null)
{
    myCPU.SetPassword(new EncryptedString("Password"));
    myCPU.Selected = true;
    retVal = myCPU.SetCurrentDateTime(DateTime.UtcNow);
    myCPU.Selected = false;
}


/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.12.4.21 SetOperatingState method

The `SetOperatingState` method sets the operating mode of a CPU.

Return type	Method name
`Result`	`SetOperatingState`

Some CPUs do not support this feature. Check the property `ChangeModeAllowed` to ensure that the current CPU supports this feature.

Example: Setting the operating mode

```
//------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//------------------------------------------------------------

#region Setting the CPU operating mode
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
if (myCPU != null)
{
   myCPU.Selected = true;
   if (myCPU.ChangeModeAllowed)
   {
      retVal = myCPU.SetOperatingState(OperatingStateREQ.Run);
   }
   myCPU.Selected = false;
}

/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.12.4.22 SetPassword method

Use the `SetPassword` method to set the password for a protected CPU.

Return type	Method name
`Result`	`SetPassword`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`password`	`EncryptedString`	In	CPU password for the device

Working with protected CPUs

If a CPU is protected, the `Protected` (Page 131) property is `true` for the `ICPU` interface. You must first check whether a CPU is protected before calling the `SetPassword` method. If the CPU is NOT protected, do NOT call the `SetPassword` method. If you call the `SetPassword` method for a CPU that is not protected, the API throws a critical error exception. The critical error exception lets you know that you are using the API incorrectly.

For CPUs that are protected, your application must call the `SetPassword` method with a password that provides a sufficient access level (Page 217) for the operation. After you set a valid password, you can refresh the status of the device.

The best practice is to call `SetPassword` on all devices for which you wish to communicate immediately after scanning the network. Set the passwords on standard CPUs to the write access password (Page 217) and for F-CPUs to the safety password (Page 217) to avoid API errors.

Example: SetPassword

```
//------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//------------------------------------------------------------

#region Setting a password
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU; // 1
92.168.0.1
if (myCPU != null)
{
   if (myCPU.Failsafe == false)
   {
      //--------------
      // Standard CPU
      //--------------
      if (myCPU.Protected)
      {
         retVal = myCPU.SetPassword(new EncryptedString("WriteAccess
Password"));
      }
   }
   else
   {
      //----------------
      // Fail-Safe CPU
      //----------------
      if (myCPU.Protected)
      {
         retVal = myCPU.SetPassword(new EncryptedString("SafetyAcces
sPassword"));
      }
   }
}
```

```
/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.12.4.23 SetProgramFolder method

Call `SetProgramFolder` to select the folder that contains the new program for the `ProgramUpdate` (Page 153) method to download to the CPU.

Return type	Method name
`Result`	`SetProgramFolder`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`strFolder`	`string`	In	Sets the folder location for the downloaded program

The `SetProgramFolder` method sets the following program update flags (Page 132) on the `ICPU` object:

- `NewProgramFolder`
- `NewProgramName`
- `NewProgramNameIP`
- `NewProgramNameSubnetMask`
- `NewProgramNameGateway`
- `NewProgramNameIsValid`

The `SetProgramFolder` method can also set the following program update flags (Page 132) on the `ICPU` object:

- `NewProgramNameIsSafety`
- `NewProgramNameHasSafetyPassword`

Example: Setting the program folder for a program update

`SetProgramFolder` is one part of the process of a CPU program update.

See the example and description in the `ProgramUpdate` method topic (Page 153)for the correct and complete sequence of steps for updating a CPU program.

4.12.4.24 SetProgramPassword method

Return type	Method name
`Result`	`SetProgramPassword`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`password`	`EncryptedString`	In	Sets the CPU password for the project that is to be passed to CPU during `ProgramUpdate` (Page 153)

The `SetProgramPassword` method sets the following program update flags (Page 132) on the `ICPU` object:

- `NewProgramNamePasswordIsValid`
- `NewProgramNamePasswordIsSafety`
- `NewProgramNamePasswordLevel`

Example: Setting the program password for a program update

`SetProgramPassword` is one part of the process of a CPU program update.

See the example and description in the `ProgramUpdate method topic` (Page 153)for the correct and complete sequence of steps for updating a CPU program.

4.12.4.25 SetTrustCertificate method

Use the SetTrustCertificate method to set the trust value of the current certificate in the CPU.

Method Name		Return Type	Description
`SetTrustCertificate()`		`Result`	Call this method to trust the current certificate.
Parameters			
Name	**Data type**	**Parameter type**	**Description**
`type`	`TrustCertificateType`	in	Type of trust (Always, Never or selection needed)

Example: Set trusting of certificate

```
//-------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-------------------------------------------------------------
#region Set trusting of certificate
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
if (myCPU != null)
{
   //-----------------------------------------------------
   // Check that the CPU supports security
   //-----------------------------------------------------


   if (myCPU.CertificateStore.Count > 1)
   {
      // Show certificate chain in windows dialog
      myCPU.CertificateStore.ShowDialog();
      //-------------------------------------------------
      // Should we trust this certificate?
      //-------------------------------------------------
      bool bTrust = true;
      if (bTrust)
         retVal =
myCPU.SetTrustCertificateStore(TrustCertificateType.Always);
      else
         retVal =
myCPU.SetTrustCertificateStore(TrustCertificateType.Never);
   }
}
/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */
#endregion
```

4.12.4.26 UploadDataLog method

The `UploadDataLog` method uploads a copy of specified Data Log file from a CPU's memory card to your programming device.

Return type	Method name
`Result`	`UploadDataLog`

Parameters			
Name	Data type	Parameter type	Description
strFileName	string	In	Filename of the Data Log to upload from a CPU's removable SIMATIC memory card
strDestinationFolder	string	In	Fully-qualified path where the uploaded file Data Log file is stored

Example: Uploading data logs

```
//-------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-------------------------------------------------------------
#region Uploading data logs

ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
if (myCPU != null)
{
   //-----------------------------------------------------
   // Check that the CPU supports remote data logs (Page 134)
   //-----------------------------------------------------

   if (myCPU.RemoteDataLogsAllowed)
   {
      //-----------------------------------
      // Check that data logs are available
      // on the memory card
      //-----------------------------------
      if (myCPU.DataLogFolder.Exists)
      {
         //-------------------------------
         // Search for all data log files
         //-------------------------------
         foreach (IRemoteFile datalog in myCPU.DataLogFolder.Files)
         {
            datalog.Selected = true;
            //--------------------------------
            // Upload a copy of each data log
            //--------------------------------
            retVal = myCPU.UploadDataLog(datalog.Name,
                @"C:\MyDataLogs");
            datalog.Selected = false;
         }
      }
   }
}
```

```
/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.12.4.27 UploadRecipe method

Use the `UploadRecipe` method to upload a copy of a recipe file from a CPU's memory card to the programming device.

Return type	Method name
`Result`	`UploadRecipe`

Parameters			
Name	Data type	Parameter type	Description
`strFileName`	`string`	In	Filename of the recipe to upload from the CPU memory card
`strDestinationFolder`	`string`	In	Fully-qualified path where the uploaded Recipe file is written

Example: Uploading recipes

```
//------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//------------------------------------------------------------
#region Uploading recipes

ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
if (myCPU != null)
{
   //-----------------------------------------------------
   // Check that the CPU supports remote recipes (Page 134)
   //-----------------------------------------------------
   if (myCPU.RemoteRecipesAllowed)
   {
      //---------------------------------
      // Check that recipes are available
      // on the memory card.
      //---------------------------------
      if (myCPU.RecipeFolder.Exists)
      {
         //-----------------------------
         // Search for all recipe files
         //-----------------------------
         foreach (IRemoteFile recipe in myCPU.RecipeFolder.Files)
         {
            recipe.Selected = true;
```

```
            //------------------------------
            // Upload a copy of each recipe.
            //------------------------------
            retVal = myCPU.UploadRecipe(recipe.Name, @"C:\MyRecipes"
);
            recipe.Selected = false;
         }
      }
   }
}

/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.12.4.28 ValidateProgramFolder method

Use the ValidateProgramFolder method to validate a CPU program folder and determine whether this program is a fail-safe program and whether it is valid for this CPU to be downloaded.

Method Name		**Return type**	**Description**
`ValidateProgramFolder()`		`Result`	Validates a CPU program folder and determines if this program is a failsafe program and if it is valid for this CPU to be downloaded.
Parameters			
Name	**Data type**	**Parameter type**	**Description**
`password`	`String`	In	Password
`bSafety`	`bool`	Out	True if this program is a safety program.
`bValid`	`bool`	Out	True if the program is valid for this CPU.

Example: Validating a program folder

```
//-------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-------------------------------------------------------------
#region Validating a program folder
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
if (myCPU != null)
{
   String strProgramPath = @"C:\MyProgram"; // Memory card image
created using TIA Portal
   bool bFailsafeProgram;
   bool bValid;

   // Validate program folder
   retVal = myCPU.ValidateProgramFolder(strProgramPath, out
bFailsafeProgram, out bValid);
}

/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.12.5 RemoteInterfaces property

4.12.5.1 Decentralized I/O modules

Each CPU can support multiple decentralized I/O interfaces. Information about the devices attached on these remote interfaces is available through the `RemoteInterfaces` property of the ICPU interface (Page 131).

Example: Inspecting remote interfaces

```
//-------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-------------------------------------------------------------

#region Inspect remote interfaces for a CPU
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
if (myCPU != null)
{
   myCPU.Selected = true;

        List<IRemoteInterface> decentralNets =
myCPU.RemoteInterfaces;
```

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


   myCPU.Selected = false;
}

/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.12.5.2 IRemoteInterface properties

The `IRemoteInterface` interface supports the following properties. These properties are read-only.

Property name	Return type	Description
`Devices`	`List<IBaseDevice>`	A list of any decentralized I/O stations connected to this remote interface
`InterfaceType`	`RemoteInterfaceType`	Communications protocol for this remote interface as defined in RemoteInterfaceType enumeration (Page 214)
`Name`	`string`	Configured name for the remote interface

You can use the `Devices` property 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 API.

The following properties are available on the `IBaseDevice` interface:

Property name	Return type	Description
`ArticleNumber`	`string`	The order number for the module. This is also known as MLFB or article number.
`Comment`	`string`	This allows the user to specify a comment for the device. This is used in the SIMATIC Automation Tool user interface. It is not relevant for API operations.
`Configured`	`bool`	Does the device have a valid configuration?

Property name	Return type	Description
`Description`	`string`	A description of the hardware item based on the article number. This is the same description that the user sees in TIA Portal. Example: "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 (Page 205) enum.
`FirmwareUpdateAllowed`	`bool`	Is firmware update possible for this device?
`FirmwareVersion`	`string`	Current firmware version of the device
`HardwareInFirmwareOrder`	`IHardwareCollection`	Hardware collection in firmware order
`HardwareInDisplayOrder`	`IHardwareCollection`	Hardware in displayed order
`HardwareNumber`	`short`	Number identifier
`ID`	`uint`	The unique identifier for every device and module in the station. This is used as the unique identifier when executing a FirmwareUpdate.
`Modules`	`IModuleCollection`	A collection of local modules (Page 128) connected on the station.
`Name`	`string`	Name of the device
`NewFirmwareFile`	`string`	File path to the new firmware file
`NewFirmwareVersion`	`string`	This property is used in the SIMATIC Automation Tool user interface. It is not relevant for API operations.
`NewFirmwareNameIsValid`	`bool`	Is the new firmware file valid?
`Selected`	`bool`	Is the device selected?
`SerialNumber`	`string`	Unique serial number for the device
`Slot`	`uint`	Slot number for the hardware item
`SlotName`	`string`	This property is used in the SIMATIC Automation Tool user interface. It is not relevant for API operations.
`StationNumber`	`uint`	Station number of the device
`SubSlot`	`uint`	The subslot of the device. This is relevant for pluggable submodules such as SB-1200.
`Supported`	`bool`	Is the detected network device supported by current SIMATIC Automation Tool API operations?

Using the Devices property of the `IRemoteInterface`, you can inspect all the stations on the decentralized network.

The following example extends the example in the Decentralized I/O modules (Page 172) topic:

This example traverses all remote PROFINET interfaces for a CPU and creates a list of the article numbers for all decentralized stations.

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

Example: Examining properties of stations on a remote interface

```
//------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//------------------------------------------------------------

#region Inspect properties of stations on remote interfaces for a
CPU
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
if (myCPU != null)
{
   myCPU.Selected = true;

   List<IRemoteInterface> decentralNets = myCPU.RemoteInterfaces;
   List<string> orderNumbers = new List<string>();
   foreach (IRemoteInterface net in decentralNets)
   {
      //------------------------------
      // Inspect the remote interface
      //------------------------------


      if (net.InterfaceType == RemoteInterfaceType.Profinet)
      {
         //-------------------------------------
         // Look at each decentralized station
         //-------------------------------------
         List<IBaseDevice> stations = net.Devices;


         //-----------------------------------------
         // Create list of the article numbers
         //-----------------------------------------
         foreach (IBaseDevice station in stations)
         {
             orderNumbers.Add(station.ArticleNumber);
         }
      }
   }
```

```
   myCPU.Selected = false;
}

/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.13 The ICPUClassic interface

4.13.1 Identifying classic CPU devices in an IProfinetDeviceCollection

The `ScanNetworkDevices` method (Page 84) generates an `IProfinetDeviceCollection` (Page 90) . This collection contains an item for every accessible device on the PROFINET network. These devices can include S7-300 and S7-400 (classic) CPUs.

The `IProfinetDevice interface` (Page 103) provides properties and methods that are applicable to all categories of devices. The `ICPUClassic` interface provides one method, `GetDiagnosticsBuffer` (Page 177), that is specific to classic CPU devices.

To determine whether a given `IProfinetDevice` interface represents a classic CPU device, cast it to an `ICPUClassic`. If this cast is successful, then the network device is a classic CPU, and you can use the method of the `ICPUClassic` interface.

Example: Determining whether a PROFINET device is a classic CPU

```
//------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//------------------------------------------------------------

#region Determining whether a device is a classic CPU
ICPUClassic myCPUClassic = scannedDevices.FindDeviceByIP(0xC0A80001)
as ICPUClassic;
// 192.168.0.1
if (myCPUClassic != null)
{
   //------------------------------------------------------------
   // The device is a classic CPU.
   // You can use the ICPUClassic interface to interact with it.
   //------------------------------------------------------------
}
#endregion
```

4.13.2 GetDiagnosticsBuffer method

The `GetDiagnosticsBuffer` method of the `ICPUClassic` interface reads the current diagnostic entries from the classic CPU. The `GetDiagnosticsBuffer` method returns a collection of `DiagnosticsItem` (Page 69) objects. The following example searches the `IProfinetDeviceCollection` for a classic CPU at a specific IP address. When found, it reads the diagnostic information from the classic CPU. Use the `Language enum` (Page 212) parameter to get diagnostic entries in a specific language.

Return type	Method name
`Result`	`GetDiagnosticsBuffer`

Parameters			
Name	Data type	Parameter type	Description
`DiagnosticsItems`	`List<DiagnosticsItem>`	Out	A collection of Diagnostics Items: Each item in the collection represents an entry in the diagnostic buffer.
`Language`	`Language`	In	Requested language for the diagnostics buffer entries.

```
//------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//------------------------------------------------------

#region Getting diagnostics from a classic CPU
ICPUClassic myCPUClassic = scannedDevices.FindDeviceByIP(0xC0A80001)
as ICPUClassic;
// 192.168.0.1
List<DiagnosticsItem> aLogs = new List<DiagnosticsItem>();
if (myCPUClassic != null)
{
   myCPUClassic.Selected = true;
   retVal = myCPUClassic.GetDiagnosticsBuffer(out aLogs,
Language.English);
   if (retVal.Succeeded)
   {
        for (int idxLog = 0; idxLog < aLogs.Count; idxLog++)
        {
        //get information time stamp
        string descrTimes = aLogs[idxLog].TimeStamp.ToString();

        //get information basic description
        string descrBasic = aLogs[idxLog].Description1;

        //get information detailed description
        string descrDetail = aLogs[idxLog].Description2;

        //get information state (incoming/outgoing)
        string descrState = aLogs[idxLog].State.ToString();
        }
```

```
   }
   myCPUClassic.Selected = false;
}

/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.14 IHMI interface

4.14.1 IHMI interface

The `ScanNetworkDevices` method (Page 84) generates an `IProfinetDeviceCollection` (Page 90). This collection contains an item for every accessible device on the network interface. These devices can include CPUs, HMIs, and other devices. The `IProfinetDevice` interface provides properties and methods that apply to all categories of devices.The `IHMI` interface inherits from `IProfinetDevice` and therefore supports all the `IProfinetDevice` properties and methods (Page 103).

The `IHMI` interface includes properties and methods that are unique to HMI devices.

To determine whether a given `IProfinetDevice` interface represents an 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.

Example: Determining whether a PROFINET device is an HMI

```
//------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//------------------------------------------------------------

#region Determining whether a device is an HMI
IHMI myHMI = scannedDevices.FindDeviceByIP(0xC0A80001) as IHMI;
// 192.168.0.1
if (myHMI != null)
{
   //-----------------------------------------------------
   // The device is an HMI.
   // You can use the IHMI interface to interact with it.
   //-----------------------------------------------------
}
#endregion
```

4.14.2 IHMI properties and flags

4.14.2.1 IHMI properties

The `IHMI` interface consists of the following properties:

Property name	Return Type	Description
`DeviceType`	`string`	Type of HMI that the object represents
`FirmwareDeviceVersion`	`string`	Firmware version that is present on the HMI
`RuntimeDeviceVersion`	`string`	Runtime version that is present on the HMI
`TransferChannel`	`HMITransferChannel` (Page 212)	Protocol used to communicate with the HMI device. Default: PN_IE

4.14.2.2 Program update flags

You can use these flags with the IHMI interface:

Property Name	Return Type	Description
`NewProgramNameIsValid`	`bool`	True when the method `SetProgramFolder` is a called with a valid program folder. False if program is not valid.
`ProgramUpdateSucceeded`	`bool`	True when program update is successful even though an error may return from internal refresh status
`NewProgramName`	`string`	Name of the new program
`NewProgramFolder`	`string`	Folder location for the new program: Value is set through the `SetProgramFolder` method
`NewProgramNameErrorCode`	`Result`	Codes to find issues that may be present in validating the new program, such as if the program is invalid for the device or if the IP assignment in the program already exists on the network

4.14.2.3 Restore flags

You can use these flags with the IHMI interface:

Property Name	Return Type	Description
`NewRestoreNameIsValid`	`bool`	True when the method SetBackupFolder is a called with a valid backup file. False if backup file is not valid.
`RestoreSucceeded`	`bool`	True when restore is successful even though an error could have been returned from internal refresh status
`NewRestoreName`	`string`	What is the name of the new program?

Property Name	Return Type	Description
`NewRestoreFile`	`string`	What is the file location for the new program? Value is set through the SetbackupFile method.
`NewRestoreNameErrorCode`	`Result`	Accessible way to find issues that may be present in validating the new program, such as if the program is invalid or incompatible with the device

4.14.2.4 Feature flags

You can use these flags with the IHMI interface:

Property Name	Return Type	Description
`BackupAllowed`	`bool`	True if the device allows backups
`BackupSupported`	`bool`	True if the device supports backups
`ProgramUpdateAllowed`	`bool`	True if the device allows program updates
`ProgramUpdateSupported`	`bool`	True if the device supports program updates
`RestoreAllowed`	`bool`	True if the device allows restores
`RestoreSupported`	`bool`	True if the device supports restores
`ScheduleFullBackup`	`bool`	Reserved for SIMATIC Automation Tool

4.14.3 IHMI methods

4.14.3.1 Backup method (IHMI interface)

Use the `Backup` method of the `IHMI` interface to back up the data for an HMI.

Return type	Method name
`Result`	`Backup`

Parameters			
Name	Data type	Parameter type	Description
`strFile`	`string`	In	A fully-qualified path and filename where the backup file is stored
`type`	`BackupType`	In (optional)	When present, specifies what data to back up: • Full backup (default) • Recipes • User administration data

Backing up an HMI

To back up an HMI, the application must perform these steps:

1. Select the HMI by setting the `Selected` (Page 103) property.
2. Call `Backup` with an optional parameter for the type of backup to perform.
3. Clear the `Selected` (Page 103) property.

Example: Backing up an HMI

```
//-------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-------------------------------------------------------------

#region Backing up an HMI
IHMI myHMI = scannedDevices.FindDeviceByIP(0xC0A80001) as IHMI;
// 192.168.0.1
if (myHMI != null)
{
    myHMI.Selected = true;
    // back up only recipes
    retVal = myHMI.Backup("C:\\MyHMIBackup.s7pbkp",
BackupType.Recipes);

    // or do full HMI backup
    retVal = myHMI.Backup("C:\\MyHMIBackup.s7pbkp");
    myHMI.Selected = false;
}

/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.14.3.2 FirmwareUpdate method

Use the `FirmwareUpdate` method of the `IHMI` interface to update on the HMI firmware.

Method Name	Return Type	Description
FirmwareUpdate()	Result	Perform a firmware update to the HMI.

Example: Updating HMI firmware

```
//-------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-------------------------------------------------------------

#region Updating HMI firmware
IHMI myHMI = scannedDevices.FindDeviceByIP(0xC0A80001) as IHMI;
```

```
// 192.168.0.1
if (myHMI != null)
{
   myHMI.Selected = true;
   myHMI.SetFirmwareFile(@"c:\myFolder\Firmware\Simatic.HMI\
KTP900_V16_00_00_03.fwf");
   retVal = myHMI.FirmwareUpdate();
   myHMI.Selected = false;
}

/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.14.3.3 ProgramUpdate method (IHMI interface)

The `ProgramUpdate` method of the `IHMI` interface updates the HMI device's operating system and run-time software.

Return type	Method name
`Result`	`ProgramUpdate`

Creating a Program Update folder

Use the TIA Portal programming software to create your HMI runtime software. Within the TIA Portal, use the Card Reader function located in the TIA Portal Project tree to create a folder. This folder will contain your HMI operating system and runtime software that the ProgramUpdate method can use. After you have created the folder, drag and drop the entire contents of the HMI device into this new folder. The ProgramUpdate method downloads both the HMI operating system and runtime software. You cannot update a partial program.

The new program folder must contain the following files for a successful program update:
`DownloadTask.xml`
`ProjectCharacteristics.rdf`

The folder that you create in the TIA Portal typically has the following format:
`{DeviceName)\Simatic.HMI\RT_Projects\{ProjectName}.{DeviceName}`

Example folder name:
`"C:\Desktop\hmim14000100a\Simatic.HMI\RT_Projects\DasBasicUndMobilePanelen.hmim14000100a[KTP700 Mobile]"`

Program update for an HMI

To update an HMI program, the application must perform these steps:

1. Select the HMI by setting the `Selected` (Page 103) property.
2. Call `SetProgramFolder` (Page 185) to select the folder that contains the new program to be downloaded to the HMI.

3. Call `ProgramUpdate`.
4. Clear the `Selected` (Page 103) property.

Example: Updating an HMI program

```
//-------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-------------------------------------------------------------

#region Updating an HMI program
IHMI myHMI = scannedDevices.FindDeviceByIP(0xC0A80001) as IHMI;
// 192.168.0.1
if (myHMI != null)
{
    myHMI.Selected = true;
    myHMI.SetProgramFolder(
@"c:\myFolder\ProgramUpdate\Simatic.HMI\RT_Projects\Project1");
   //----------------------------------------
   // The HMI project for program update is the
   // operating system and runtime software
   //----------------------------------------
    retVal = myHMI.ProgramUpdate();
    myHMI.Selected = false;
}
/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.14.3.4 Restore method (IHMI interface)

Use the `Restore` method of the `IHMI` interface to restore HMI device data from a previous backup of the device.

Return type	Method name
`Result`	`Restore`

Restoring from an HMI backup file

To restore an HMI backup file, the application must perform these steps:

1. Select the HMI by setting the `Selected` (Page 103) property.
2. Call `SetBackupFile` (Page 184) to select the backup file to be downloaded to the HMI.
3. Call `Restore`.
4. Clear the `Selected` (Page 103) property.

Example: Restoring an HMI backup file

```
//-------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-------------------------------------------------------------

#region Restoring an HMI backup

IHMI myHMI = scannedDevices.FindDeviceByIP(0xC0A80001) as IHMI; //
192.168.0.1
if (myHMI != null)
{
   myHMI.Selected = true;
   retVal = myHMI.SetBackupFile(@"C:\MyFolder\Backup.s7pbkp");
   retVal = myHMI.Restore();
   myHMI.Selected = false;
}

/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.14.3.5 SetBackupFile method

Use the `SetBackupFile` method of the `IHMI` interface to select the backup file to be restored to an HMI.

Return type	Method name
`Result`	`SetBackupFile`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`strFile`	`string`	in	Sets the folder location where the backup file source is stored

The method sets the following flags on the IHMI object:

- `NewRestoreName`
- `NewRestoreFile`
- `NewRestoreNameIsValid`

Example: Setting an HMI backup file

See the example in the `Restore` method topic (Page 183) for setting a backup file to restore to an HMI.

4.14.3.6 SetProgramFolder method

Use the `SetProgramFolder` method to set the folder for the program update (Page 182) for an HMI device:

Return type	Method name
`Result`	`SetProgramFolder`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`strFolder`	`string`	in	Sets the folder location where the program download source is stored

Example: Setting the program file to update

See the example in the `ProgramUpdate` method topic (Page 182) for setting a program file for updating an HMI.

4.14.3.7 SetTransferChannel method

HMI devices can support multiple protocols to exchange data with the API. The protocols are the transfer channels. The `SetTransferChannel` method sets the HMI transfer channel (Page 212):

Return type	Method name
`Result`	`SetTransferChannel`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`transferChannel`	`HMITransferChannel`	in	Sets the type of communications (Transfer Channel) for communication with the HMI device

The default transfer channel is `HMITransferChannel.PN_IE`.

Example: Setting the HMI transfer channel

```
//------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//------------------------------------------------------------

#region Setting the HMI transfer channel
IHMI myHMI = scannedDevices.FindDeviceByIP(0xC0A80001) as IHMI;
// 192.168.0.1
if (myHMI != null)
{
   // assign PN/IE as the protocol for communications to this HMI
device.
   retVal = myHMI.SetTransferChannel(HMITransferChannel.PN_IE);
}

/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.15 IScalance interface

4.15.1 IScalance interface

The `IScalance` interface includes properties and methods that are unique to SCALANCE devices.

The `ScanNetworkDevices` method (Page 84) generates an `IProfinetDeviceCollection` (Page 90). This collection contains an item for every accessible device on the network interface. These devices can include CPUs, HMIs, SCALANCE and other devices. The `IProfinetDevice` interface provides properties and methods that apply to all categories of devices.The `IScalance` interface inherits from `IProfinetDevice` and therefore supports all the `IProfinetDevice` properties and methods (Page 103).

To determine whether a device interface represents a SCALANCE device, cast the IProfinetDevice interface (Page 103) to an IScalance interface. If the cast is successful, the network device is a SCALANCE device, and you can use the IScalance methods (Page 187) and properties (Page 187). .

Example: Determining whether a PROFINET device is a SCALANCE device

```
//------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//------------------------------------------------------------

#region Determining whether a device is a SCALANCE device
IScalance myScalance = scannedDevices.FindDeviceByIP(0xC0A80001) as
IScalance;
// 192.168.0.1
if (myScalance != null)
{
   //------------------------------------------------------------
   // The device is a SCALANCE device.
   // You can use the IScalance interface to interact with it.
   //------------------------------------------------------------
}
#endregion
```

4.15.2 IScalance properties

The IScalance interface has the following properties:

Property name	Return type	Description
ProfileName	string	Returns the name of the SNMP profile
ProfileNameisValid	bool	TRUE if a valid profile name exists

4.15.3 IScalance methods

4.15.3.1 SetProfile method

Call the SetProfile method to establish the profile configuration for a SCALANCE device.

Return type	Method name
Result	SetProfile

Parameters			
Name	**Data type**	**Parameter type**	**Description**
Profile	ISNMPProflie	In	SNMP profile and TFTP (Trivial File Transfer Protocol) configuration for the SCALANCE device's FW update

Example: Setting an SNMP profile

You can see an example of the `SetProfile` method call in any of the following examples:

- Example: SNMP Version 1 configuration (Page 196)
- Example: SNMP Version 2 configuration (Page 198)
- Example: SNMP Version 3 configuration (Page 199)

In the examples, the `SetProfile` call is near the end of the example code.

4.15.3.2 FirmwareUpdate method

Call the `FirmwareUpdate` method to update the firmware of a SCALANCE device.

Return type	Method name
`Result`	`FirmwareUpdate`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`type`	`FirmwareUpdateType` (Page 212)	In	Optional input parameter that specifies the type of firmware update operation to perform

Example: Updating the firmware of a SCALANCE device

You can see an example of the `FirmwareUpdate` method call in any of the following examples:

- Example: SNMP Version 1 configuration (Page 196)
- Example: SNMP Version 2 configuration (Page 198)
- Example: SNMP Version 3 configuration (Page 199)

The examples show both types of firmware update:

- Downloading and activating the firmware in a single `FirmwareUpdate` method call
- Downloading the firmware with `FirmwareUpdate` as a first step and activating the firmware with `FirmwareActivate` as a second step.

4.15.3.3 FirmwareActivate method

The `FirmwareActivate` method activates the downloaded firmware update file for the specified SCALANCE device at `hardwareID`.

Return type	Method name
`Result`	`FirmwareActivate`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`hardwareID`	`uint32`	In	Hardware identifier of the SCALANCE device

Example: Activating downloaded firmware on a SCALANCE device

The following examples show how to perform a `FirmwareActivate`, after a `FirmwareUpdate` (Page 188) that downloads the firmware without activating it. The three examples show each of the three SNMP version types.

- Example: SNMP Version 1 configuration (Page 196)
- Example: SNMP Version 2 configuration (Page 198)
- Example: SNMP Version 3 configuration (Page 199)

4.16 ISNMPProfile interface

4.16.1 ISNMPProfile properties

The ISNMPProfile is a collection of properties that you use to perform a SCALANCE device firmware update using SNMP. You must set up a TFTP server to update firmware of a SCALANCE device.

The properties applicable for a given device's firmware update depend on the configured SNMP protocol version on that device. The property descriptions indicate for which SNMP version (V1, V2, V3) a property is applicable. Properties without a version number in the description are applicable to all SNMP versions.

Property name	Return type	Description
`ProfileName`	`string`	Name of the SNMP profile
`Version`	`SNMPVersion`	The SNMP protocol version used
`ServerIP`	`string`	String representation of the IP address
`ServerIPArray`	`byte[]`	IP address of the TFTP server
`ServerPort`	`UInt16`	TFTP port assignment
`ReadCommunity`	`string`	V1/V2 read privilege ID string (PW)
`WriteCommunity`	`string`	V1/V2 write privilege ID string (PW)
`UserName`	`string`	V3 user name

Property name	Return type	Description
ContextName	string	V3 management info context ID
SecurityLevel	SNMPSecurityLevel	V3 security level
AuthAlgorithm	SNMPAuthAlgorithm	V3 authentication Algorithm
AuthKey	byte[]	V3 authentication key
PrivAlgorithm	SNMPPrivAlgorithm	V3 privacy algorithm
PrivKey	byte[]	V3 privacy key

4.16.2 ISNMPProfile methods

4.16.2.1 Validate method

The Validate method evaluates the SNMP profile arguments to ensure that there is a consistent set of required firmware update parameters. The returned result identifies any inconsistencies.

Return type	Method name
Result	Validate

Parameters			
Name	**Data type**	**Parameter type**	**Description**
None			

You can see an example of the Validate method call in any of the following examples:

- Example: SNMP Version 1 configuration (Page 196)
- Example: SNMP Version 2 configuration (Page 198)
- Example: SNMP Version 3 configuration (Page 199)

In these examples, the Validate call occurs after setting the profile parameters.

4.16.3 SNMPProfile class

4.16.3.1 SNMPProfile class properties

The SNMPProfile class is for storing and editing SNMP profiles.

Property name	Return type	Description
ProfileName	string	Name of the SNMP Profile
Version	SNMPVersion	SNMP protocol version used
ServerIP	string	String representation of the IP address
ServerIPArray	byte[]	IP address of the TFTP server
ServerPort	UInt16	TFTP port assignment

Property name	Return type	Description
ReadCommunity	string	V1/V2 read privilege ID string
WriteCommunity	string	V1/V2 write privilege ID string
UserName	string	V3 user name
ContextName	string	V3 management info context ID
SecurityLevel	SNMPSecurityLevel	V3 security level
AuthAlgorithm	SNMPAuthAlgorithm	V3 authentication Algorithm
AuthKey	byte[]	V3 authentication key
PrivAlgorithm	SNMPPrivAlgorithm	V3 privacy algorithm
PrivKey	byte[]	V3 privacy key

4.16.3.2 SNMPProfile class methods

SetProfileName

The SetProfileName method assigns the input string value to the name of the profile.

Return type	Method name
Result	SetProfileName

Parameters			
Name	**Data type**	**Parameter type**	**Description**
strName	string	in	Profile name

You can see an example of the SetProfileName method call in any of the following examples:

- Example: SNMP Version 1 configuration (Page 196)
- Example: SNMP Version 2 configuration (Page 198)
- Example: SNMP Version 3 configuration (Page 199)

In these examples, the SetProfileName call occurs near the beginning of the code example.

SetSNMPVersion

The SetSNMPVersion method sets the SNMP protocol version. The SNMP protocol is either Version 1, 2, or 3.

Return type	Method name
Result	SetSNMPVersion

Parameters			
Name	**Data type**	**Parameter type**	**Description**
nVersion	SNMPVersion (Page 216)	in	SNMP version 1,2, 3

You can see an example of the SetSNMPVersion method call in any of the following examples:

- Example: SNMP Version 1 configuration (Page 196)
- Example: SNMP Version 2 configuration (Page 198)
- Example: SNMP Version 3 configuration (Page 199)

In these examples, the SetSNMPVersion call occurs near the beginning of the code example.

SetServerIP method

The SetServerIP method sets the IP address of the TFTP server. This method is applicable to all SNMP profiles.

Return type	Method name
`Result`	`SetServerIP`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`strIP`	`string`	in	IP assignment

You can see an example of the SetServerIP method call in any of the following examples:

- Example: SNMP Version 1 configuration (Page 196)
- Example: SNMP Version 2 configuration (Page 198)
- Example: SNMP Version 3 configuration (Page 199)

In these examples, the SetServerIP call occurs near the beginning of the code example.

SetServerPort method

The SetServerPort method sets the port assignment of the TFTP server. This method is applicable to all SNMP profiles.

Return type	Method name
`Result`	`SetServerPort`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`nVersion`	`UInt16`	in	TFTP port assignment

You can see an example of the SetServerPort method call in any of the following examples:

- Example: SNMP Version 1 configuration (Page 196)
- Example: SNMP Version 2 configuration (Page 198)
- Example: SNMP Version 3 configuration (Page 199)

In these examples, the SetServerPort call occurs just after the SetServerIP call.

SetReadCommunity method

The SetReadCommunity method assigns the read community to the input string. The read community string enables a remote device to retrieve read-only information from a device.

Return type	Method name
`Result`	`SetReadCommunity`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`strCommunity`	`string`	in	Read community assignment

You can see an example of the SetReadCommunity method call in the following examples:

- Example: SNMP Version 1 configuration (Page 196)
- Example: SNMP Version 2 configuration (Page 198)
- Example: SNMP Version 3 configuration (Page 199)

SetWriteCommunity method

The SetWriteCommunity method assigns the write community to the input string. The write community string permits both read operations and write operations for a device.

Return type	Method name
`Result`	`SetWriteCommunity`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`strCommunity`	`string`	in	Write community assignment

You can see an example of the SetWriteCommunity method call in any of the following examples:

- Example: SNMP Version 1 configuration (Page 196)
- Example: SNMP Version 2 configuration (Page 198)
- Example: SNMP Version 3 configuration (Page 199)

SetUserName method

The SetUserName method sets the user name for an SNMP Version 3 profile.

Return type	Method name
`Result`	`SetUserName`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`strName`	`string`	in	SNMP V3 user name

You can see an example of the SetUserName method call near the middle of the Example: SNMP Version 3 configuration (Page 199) code example.

SetContextName method

The SetContextName method sets the context name for an SNMP Version 3 profile.

Return type	Method name
`Result`	`SetContextName`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
`strName`	`string`	in	SNMP V3 management information context

You can see an example of the SetContextName method call near the middle of the Example: SNMP Version 3 configuration (Page 199) code example.

SetSecurityLevel method

The SetSecurityLevel method sets the security level for an SNMP Version 3 profile.

Return type	Method name
`Result`	`SetSecurityLevel`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
level	`SNMPSecurityLevel` (Page 216)	in	SNMP V3 security level

You can see an example of the SetSecurityLevel method call near the middle of the Example: SNMP Version 3 configuration (Page 199) code example.

SetAuthAlgorithm method

The SetAuthAlgorithm method sets the authentication algorithm for an SNMP Version 3 profile. The authentication algorithm is applicable when the security level (Page 194) requires "Authentication". The authentication password is applicable when the security level requires "Privacy" as well as "Authentication".

Return type	Method name
`Result`	`SetAuthAlgorithm`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
algorithm	`SNMPAuthAlgorithm` (Page 215)	in	SNMP V3 authentication algorithm
strPassword	`string`	in	SNMP V3 authentication password

You can see an example of the SetAuthAlgorithm method call near the middle of the Example: SNMP Version 3 configuration (Page 199) code example.

SetPrivAlgorithm method

The SetPrivAlgorithm method sets the privacy algorithm for an SNMP Version 3 profile. The privacy algorithm is applicable when the security level (Page 194) requires "Privacy".

Return type	Method name
`Result`	`SetPrivAlgorithm`

Parameters			
Name	**Data type**	**Parameter type**	**Description**
algorithm	`SNMPPrivAlgorithm` (Page 216)	in	SNMP V3 privacy algorithm
strPassword	`string`	in	SNMP V3 privacy password

You can see an example of the SetPrivAlgorithm method call near the middle of the Example: SNMP Version 3 configuration (Page 199) code example.

4.17 IScalance and ISNMP firmware update code examples

4.17.1 Example: SNMP Version 1 configuration

This example shows how to configure and initiate a firmware update for a SCALANCE device. In this example, the SCALANCE device uses SNMP Version 1.

The example shows both types of firmware update:

- Downloading and activating the firmware in a single `FirmwareUpdate` (Page 188) method call
- Downloading the firmware with `FirmwareUpdate` as a first step and activating the firmware with `FirmwareActivate` (Page 189) as a second step.

Example: SNMP Version 1 firmware update

```
//-------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-------------------------------------------------------------

#region Updating firmware for a SCALANCE device, SNMP Version 1
profile


SNMPProfile profileV1 = new SNMPProfile();

// Create a profile using SNMP version 1
retVal = profileV1.SetSNMPVersion(SNMPVersion.Version1);

// Set name for this profile
retVal = profileV1.SetProfileName("Profile_1");

// Set TFTP Server IP address, required
retVal = profileV1.SetServerIP("192.168.0.1");

// Set TFTP Server port, default is 69, optional
retVal = profileV1.SetServerPort(69);

// Set SCALANCE read community, default is "public", optional
retVal = profileV1.SetReadCommunity("public");

// Set SCALANCE write community, default is "private", optional
retVal = profileV1.SetWriteCommunity("private");

// Verify profile is valid; this assures we did not miss a
parameter
retVal = profileV1.Validate();


// Update firmware for a specific SCALANCE devices
```

```
IScalance myScalance = scannedDevices.FindDeviceByIP(0xC0A80001) as
IScalance;
if (myScalance != null)
{
   myScalance.Selected = true;
   // Set SNMP profile required to communicate with this
SCALANCE device
   retVal = myScalance.SetProfile(profileV1);

   // Set new firmware file to update SCALANCE
   retVal =
myScalance.SetFirmwareFile(@"C:\Firmware\FirmwareFile.lad");

   // ------------------------------------------------------
   // Update new firmware on SCALANCE device.
   // This method call downloads and activates the firmware.
   // ------------------------------------------------------
   retVal = myScalance.FirmwareUpdate();

   // ------------------------------------------------
   // Alternatively, use two-step operation to update
   // new firmware on SCALANCE device.
   // This operation downloads the firmware in one step
   // and activates the firmware in a second step.
   // ------------------------------------------------
   retVal =
myScalance.FirmwareUpdate(FirmwareUpdateType.DownloadWithoutActivati
on);

   // ------------------------------------------------------------
   // This step activates the downloaded firmware
   // on the SCALANCE device. Activation can be done later and might
   // be based on user input. For simplicity, this example activates
   // the firmware after the download.
   // ------------------------------------------------------------
   retVal = myScalance.FirmwareActivate();
   myScalance.Selected = false;
}
/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */
#endregion
```

4.17.2 Example: SNMP Version 2 configuration

This example shows how to configure and initiate a firmware update for a SCALANCE device. In this example, the SCALANCE device uses SNMP Version 2:

The example shows both types of firmware update:

- Downloading and activating the firmware in a single `FirmwareUpdate` (Page 188) method call
- Downloading the firmware with `FirmwareUpdate` as a first step and activating the firmware with `FirmwareActivate` (Page 189) as a second step.

Example: SNMP Version 2 firmware update

```
//-----------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//-----------------------------------------------------------------

#region Updating firmware for a SCALANCE device, SNMP Version 2
profile


SNMPProfile profileV2 = new SNMPProfile();

// Create a profile using SNMP version 2
retVal = profileV2.SetSNMPVersion(SNMPVersion.Version2);

// Set name for this profile
retVal = profileV2.SetProfileName("Profile_2");

// Set TFTP Server IP address, required
retVal = profileV2.SetServerIP("192.168.0.1");

// Set TFTP Server port, default is 69, optional
retVal = profileV2.SetServerPort(69);

// Set SCALANCE read community, default is "public", optional
retVal = profileV2.SetReadCommunity("public");

// Set SCALANCE write community, default is "private", optional
retVal = profileV2.SetWriteCommunity("private");

// Verify profile is valid, this assures we did not miss a parameter
retVal = profileV2.Validate();

//  Update firmware for a specific  SCALANCE  devices

IScalance myScalance = scannedDevices.FindDeviceByIP(0xC0A80001) as
IScalance;
if (myScalance != null)
{
   myScalance.Selected = true;
   //  Set  SNMP  profile  required  to  communicate  with  this
```

```
SCALANCE  device
   retVal  =  myScalance.SetProfile(profileV2);

   //  Set  new  firmware  file  to  update  SCALANCE
   retVal  =
myScalance.SetFirmwareFile(@"C:\Firmware\FirmwareFile.lad");

   // ------------------------------------------------------
   // Update new firmware on SCALANCE device.
   // This method call downloads and activates the firmware.
   // ------------------------------------------------------
   retVal  =  myScalance.FirmwareUpdate();

   // -------------------------------------------------
   // Alternatively, use two-step operation to update
   // new firmware on SCALANCE device.
   // This operation downloads the firmware in one step
   // and activates the firmware in a second step.
   // -------------------------------------------------
   retVal  =
myScalance.FirmwareUpdate(FirmwareUpdateType.DownloadWithoutActivati
on);

   // ------------------------------------------------------------
   // This step activates the downloaded firmware
   // on the SCALANCE device. Activation can be done later and might
   // be based on user input. For simplicity, this example activates
   // the firmware after the download.
   // ------------------------------------------------------------
   retVal  =  myScalance.FirmwareActivate();
   myScalance.Selected = false;
}

/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.17.3 Example: SNMP Version 3 configuration

This example shows how to configure and initiate a firmware update for a SCALANCE device. In this example, the SCALANCE device uses SNMP Version 3 with authentication and privacy:

The example shows both types of firmware update:

- Downloading and activating the firmware in a single `FirmwareUpdate` (Page 188) method call
- Downloading the firmware with `FirmwareUpdate` as a first step and activating the firmware with `FirmwareActivate` (Page 189) as a second step.

Example: SNMP Version 3 firmware update with authentication and privacy

```
//------------------------------------------------------------
// Insert the necessary code from Getting started with the API (Page 45)
// here if you want to compile the example
//------------------------------------------------------------

#region Updating firmware for a SCALANCE device, SNMP Version 3
profile


SNMPProfile profileV3 = new SNMPProfile();

// Create a profile using SNMP version 3
retVal = profileV3.SetSNMPVersion(SNMPVersion.Version3);

// Set name for this profile
retVal = profileV3.SetProfileName("Profile_3.1");

// Set TFTP Server IP address, required
retVal = profileV3.SetServerIP("192.168.0.1");

// Set TFTP Server port, default is 69, optional
retVal = profileV3.SetServerPort(69);

// Set SCALANCE read community, default is "public", optional
retVal = profileV1.SetReadCommunity("public");

// Set SCALANCE write community, default is "private", optional
retVal = profileV1.SetWriteCommunity("private");
// Set User name, required
retVal = profileV3.SetUserName("User1");

// Set Context name, optional
retVal = profileV3.SetContextName("Context1");

// Set SCALANCE security level AuthPriv, required
retVal = profileV3.SetSecurityLevel(SNMPSecurityLevel.AuthPriv);

// Set Authentication algorithm to MD5 and password, required
retVal = profileV3.SetAuthAlgorithm(SNMPAuthAlgorithm.MD5,
"Password1");

// Set Privacy algorithm to DES and password, required
retVal = profileV3.SetPrivAlgorithm(SNMPPrivAlgorithm.DES,
"Password2");

// Verify profile is valid, this assures we did not miss a parameter
retVal = profileV3.Validate();


// Update firmware for a specific SCALANCE device
```

```
IScalance myScalance = scannedDevices.FindDeviceByIP(0xC0A80001) as
IScalance;
if (myScalance != null)
{
   myScalance.Selected = true;
   // Set SNMP profile required to communicate with this
SCALANCE device
   retVal = myScalance.SetProfile(profileV3);

   // Set new firmware file to update SCALANCE
   retVal =
myScalance.SetFirmwareFile(@"C:\Firmware\FirmwareFile.lad");

   // ------------------------------------------------------
   // Update new firmware on SCALANCE device.
   // This method call downloads and activates the firmware.
   // ------------------------------------------------------
   retVal = myScalance.FirmwareUpdate();

   // ------------------------------------------------------
   // Alternatively, use two-step operation to update
   // new firmware on SCALANCE device.
   // This operation downloads the firmware in one step
   // and activates the firmware in a second step.
   // ------------------------------------------------------
   retVal =
myScalance.FirmwareUpdate(FirmwareUpdateType.DownloadWithoutActivati
on);

   // ------------------------------------------------------------
-
   // This step activates the downloaded firmware
   // on the SCALANCE device. Activation can be done later and might
   // be based on user input. For simplicity, this example activates
   // the firmware after the download.
   // ------------------------------------------------------------
   retVal = myScalance.FirmwareActivate();
   myScalance.Selected = false;
}

/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

SNMP Version 3 firmware update with no authentication, no privacy

To modify the example to use no authentication and no privacy, remove the example code that sets the security level, authentication algorithm and privilege algorithm. Replace the deleted code with the following code:

```
// Set SCALANCE security level NoAuthNoPriv, required
retVal = profileV3.SetSecurityLevel(SNMPSecurityLevel.NoAuthNoPriv);
```

SNMP Version 3 firmware update with with authentication, no privacy

To modify the example to use authentication but with no privacy, remove the example code that sets the security level, authentication algorithm and privilege algorithm. Replace the deleted code with the following code:

```
// Set SCALANCE security level AuthNoPriv, required
retVal = profileV3.SetSecurityLevel(SNMPSecurityLevel.AuthNoPriv);

// Set Author algorithm to MD5 and password, required
retVal = profileV3.SetAuthAlgorithm(SNMPAuthAlgorithm.MD5,
"Password1");
```

4.18 Exceptions

4.18.1 CriticalInternalErrorException

The API interface includes an exception when a critical condition has been detected.

When your application detects that the CriticalInternalErrorException exception has triggered, shut down the application that is using the API. A critical error has occurred.

```
#region Critical internal error exception
try
{
   //------------------------------------------------------------
   // Insert the necessary code from Getting started with the API (Page 45)
   // here if you want to compile the example
   //------------------------------------------------------------


   ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU; /
/ 192.168.0.1
   if (myCPU != null)
   {
      myCPU.SetPassword(new EncryptedString("Password"));
      myCPU.Selected = true;
      if (myCPU.Failsafe)
         myCPU.SelectedConfirmed = true;
      retVal = myCPU.ResetToFactoryDefaults();
```

```
    }
}
catch (CriticalInternalErrorException e)
{
    // A critical internal error has occurred within the API

    // Shut down the application.
    myCPU.Selected = false;
    myCPU.SelectedConfirmed = false;
}
catch (Exception e)
{
    // An exception has occurred within the API
    myCPU.Selected = false;
    myCPU.SelectedConfirmed = false;

}

/* For simplicity the code examples above do not check for */
/* errors. Checking for and processing errors returned     */
/* from methods is vital to a program's overall quality    */

#endregion
```

4.19 API enumerations

4.19.1 BackupType

The `BackupType` enumeration indicates the protocol used to communicate with an HMI device:

Description	Value
`Invalid`	0
`FullBackup`	1
`Recipes`	2
`UserAdministration`	3

4.19.2 CardSortDirection

The `CardSortDirection` enumeration indicates the sort direction of SD card contents:

```
None
Ascending
Descending
```

4.19.3 CardSortItem

The CardSortItem enumeration is the key for sorting data on the SD card:

```
DateCreated
DateModified
FileSize
Name
Type
```

4.19.4 CertificateStatusError

The CertificateStatusError enumeration contains possible types of certificate errors:

```
CertStatusAuthorityInvalid
CertStatusDateInvalid
CertStatusInvalid
CertStatusIPAddress
CertStatusRevoked
None
```

4.19.5 ConfigurationDataProtection

The ConfigurationDataProtection enumeration describes the state of the configuration of data protection in the CPU. It contains the following values:

```
ProtectionUnknown
NoProtection_CPUProgramMatch
NoProtection_CPUProgramMismatch
NoProtection_NoCPUProgram
ProtectionEnabled_CPUProgramMatch
ProtectionEnabled_CPUProgramMismatch
ProtectionEnabled_NoCPUProgram
```

4.19.6 ConfirmationType

This enumeration is used to indicate the status of fail-safe CPUs:

Description	Value
Invalid	0
SafetyPasswordIsBeingUsed	0x2f161717
DeletingExistingSafetyProgram	0x40232122
ReplacingExistingSafetyProgram	0x492a282b
ReplacingExistingSafetyProgramWithNonSafetyProgram	0x4a2c2b2d
LoadingSafetyProgram	0x46292728

4.19.7 DataChangedType

This enumeration defines the possible argument values for the DataChangedEventHandler: (Page 126)

```
Invalid
OperatingState
RackInformation
Folders
File
ProfinetName
IPAddress
Password
FileSystem,
StopModeTransition
```

4.19.8 DeviceFamily

This enumeration specifies the product family for a hardware item:

```
None
CPU1200
CPU1500
CPU300
CPU400
ET200SP
ET200MP
ET200AL
ET200PRO
ET200ECO
ET200S
ET200M
HMI
SITOPUPS
SCALANCE
SIMOCODE
Unsupported
SIRIUS_ACT
Gateway
NetworkDevice
MicroDrive
SinamicsDrive
SoftStarter
CommunicationsModule
OpticalReader
RFID
```

4.19.9 ErrorCode

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

```
OK
AccessDenied
ServiceTimeout
Disconnected
FailedToDisconnect
ServiceNotConnected
TooManySessions
SessionDelegitimated
NotChangableInRun
InvalidFileName
MultiESNotSupported
ServiceAborted
MultiESLimitExceeded
MultiESIncompatibleOtherESVersion
MultiESConflict
WriteProtected
DiskFull
InvalidVersion
PathNotFound

Failed
CPUFailedToEnterRunMode
MACAddressIsNotValid
IPAddressIsNotValid
SubnetMaskIsNotValid
GatewayIsNotValid
ProfinetNameIsNotValid
NewIPAddressIsNotValid
NewSubnetMaskIsNotValid
NewGatewayIsNotValid
NewProfinetNameIsNotValid
InvalidPointer
SetIPErrorDueProjectSettings
UnsupportedDevice
SetNameErrorDueProjectSettings
OperationNotSupportedByThisDevice
DeviceNotOnNetwork
FirmwareVersionMatch
FirmwareFileNotCompatibleToNew
FirmwareFileNotCompatibleToOld
FirmwareFileNotCompatibleNotSame
FirmwareFileNotCompatibleSame
FirmwareFileNotCompatible
FirmwareModuleNotReachable
FirmwareModuleNotAccepted
FirmwareIDNotFound
WriteBlockFailed
InvalidProjectVersion
DeviceIsNotAcceptingChanges
```

```
InvalidSignature
ParmeterOutOfRange
FailedToZipFolderContents
ErrorWritingToFile
ErrorCreatingFile
ErrorCreatingFolder
NoSATLicensePresent
InvalidTimeoutValue
NoDataToBackup
ErrorWritingToStream
ErrorReadingFromStream
InvalidProjectPath
ProjectNotCompatibleWithDevice
FailedToSetProfinetName
FailedToSetIPAddress
DownloadInvalidRecipe
IdentityFailure
DeviceMismatch
InvalidInterface
DeviceNotSelected
FailsafeAccessRequired
InternalApplicationError
InvalidPassword
DuplicateIPAddress
DuplicateProfinetName
SafetyDeviceMustBeConfirmed
NoSDCardPresent
InvalidProgramFolder
FSignaturesDoesNotMatch
FSignaturesMatch
DeviceDoesNotSupportProject
ProjectsUpdateIPNotReachable
RestoreIPNotReachable
ProjectIPNotUnique
SafetyProjectDownloadedToStandardNotAllowed
PasswordDiversityFailed
InvalidBackupFile
InvalidBackupFileExtension
IncompatibleBackupFile
InvalidFirmwareFile
OperationWasNotSuccessful
CouldNotValidatePassword
IPAddressAlreadyExistsOnNetwork
MissingProgramFilePassword
InvalidProgramFilePassword
OperationCanceledByUser
InvalidProgramForDevice
InvalidProgramFilePasswordLegitimizationLevel
DeviceNotFound
DeviceAlreadyExists
IPAddressAlreadyOnNetwork
ProfinetNameAlreadyOnNetwork
```

```
FailedToConnect
DeviceNotInitialized
CPUNewerVersionNotSupported
IPSuitNotValid
IPAddressChanged
ScanNoDevicesFound
DeviceCannotBeInserted
InsertDeviceDuplicateIP
IPNotReachable
CouldNotReadFSignature
InvalidNetworkInterface
InsufficientLegitimizationLevel
NoProgramPassword
ProjectVersionV1NotSupported
ProjectOpenCanceled
ProgramPasswordNeeded
RestoreError
IncompatibleProgramFile
UnsupportedProgramFile
ProgramFileFamilyMismatch
DuplicateNewIPAddress
SNMPErrorNoAccess
SNMPErrorReadOnly
SNMPErrorNotWritable
SNMPErrorAuthorizationError
SNMPError
InvalidProfileName
InvalidSNMPVersion
InvalidServerIP
InvalidServerPort
InvalidReadCommunity
InvalidWriteCommunity
InvalidUserName
InvalidContextName
InvalidSecurityLevel
InvalidAuthAlgorithm
InvalidAuthPassword
InvalidPrivAlgorithm
InvalidPrivPassword
InvalidProfile
ProfileNameAlreadyExists
ObsoleteMethod
FailedToInitiateFirmwareTransfer
DeviceDefinedError
ErrorDeletingFile
ErrorDeletingFolder
ErrorInvalidMAC
FailedToUpdateDuplicates
DuplicateNewProfinetName
ErrorBackingupData
FailsafeControlObjectIncorrectType
InvalidIPAddressOrUsedByNIC
```

```
FirmwareIntegrityFailed
ExportDiagnosticsBufferError
AlmFailedInitialize
AlmFailedCleanup
AlmFailedSessionInitialize
AlmFailedSessionCleanup
AlmSessionIDMissing
AlmSessionIDUnknown
AlmCouldNotConnect
AlmOutOfMemory
AlmOperationTimeout
AlmFunctionNotFound
AlmAborted
AlmBadfunctionArgument
AlmUnkownOption
AlmSendError
AlmReceiveError
AlmNoConnectionAvailable
AlmOpenSession
AlmResources
AlmService
AlmCryptography
AlmTaskAlreadyRunning
AlmInvalidPointer
AlmResultMismatch
AlmBadResult
AlmBatchWrongArgumentNumber
AlmBatchWrongArgument
AlmBatchWrongInputFile
AlmBatchWrongInputStream
AlmBatchAPILoadFailed
AlmBatchOutputfileExists
AlmBatchWrongOutputParmater
AlmBatchOutputCreationFailure
AlmBatchAccessDenied
AlmUnknownError
PowerCycleRequired
DuplicateFolders
DuplicateFiles
FeatureRequiresValidLicense
APIFeatureNotSupported
CPUProtectionLevelWeaker
CouldNotReadProtectionLevel
ChangingIPNotAllowedNatRouter
IPAndRouterIPCannotBeTheSame
OperationNotAllowedThroughCPCM
FolderOrFileDoesNotExist
DCPOperationNotSupportedBehindNATRouters
OperationNotSupportedThroughCPCM
CPUFailedToEnterStopMode
NoFirmwareToActivate
FailedToActivateFirmware
```

```
InvalidOperation
ErrorRefreshingFolder
ErrorCannotRenameFileAlreadyExists
FileWriteProtected
FolderWriteProtected
MemoryCardNotPresent
OperationRequiresStopMode
ErrorWritingFileMemoryCard
ErrorCreatingFolderMemoryCard
ErrorReadingServiceData
NoFilesInListToDownload
InsufficientLegitimizationLevelRefreshStatus
APIFeatureAdvancedNotSupported

TLSConnectionTrustRequired
TLSConnectionNever
TLSCertificateDisabled
TLSCertificateEnabled
TLSCertificateEnabledBootstrap
ConfigurationDataProtectionDoesNotMatch
ErrorInvalidMemoryCardPath
ErrorInvalidMemoryCard
ErrorInvalidProgramFolder
DeviceDuplicateSerialNumberRemovedInsert
DeviceDuplicateSerialNumberRemovedScan
InvalidScanFilter
BootStrapCertificateAlreadyTrusted
CPUGeneratedCertificate
MultipleInterfacesWarning
DeviceDoesNotSupportDeleteOption
OnlineOfflineCompareFailed
PreconditionsFAddressAssignmentFailed
FAddressParameterizationCRCFailed
FailedEstablishConnection
FailedEstablishConnectionSNMP
NotConnectedToFCPU

//HMI
UnexpectedOperatingSystemError
ServiceActive
RemoteTransferDisabled
HardwareSoftwareNotComplete
LogicalVolumneMissing
LogicalVolumneOutOfSpace
Abort
FirwareTypeNotSupported
FirwareTypeNotInstalled
StoreReadFailed
StoreWriteFailed
RescueBackupNotPossible
RescueRestoreNotPossible
ConnectionRequired
```

```
ObjectNotFound
BufferToSmall
InvalidArguements
AttributeNotFound
InvalidPath
TypeConversionFailed
FileReadFailed
FileWriteFailed
OutOfResources
OutOfSpace
UnknownAddon
IncompatibleAddon
AddonsUnsupported
LicenseFailed
UnknownApp
UnknownAppAddon
UnknownReferenceApp
RuntimeMissing
RuntimeBroken
SignatureRequired
SignatureInvalid
SignatureFailure
CertificateInvalid
CertificateFailure
CertificateNotReady
CertificateExpired
CertificateRevoked
SecurityLib
WrongRuntimeVersion
MajorRuntimeDowngrade
MajorRuntimeUpgrade
MajorImageDowngrade
MajorImageUpgrade
WrongRuntime
NotEnoughMemory
ProjectCharacteristicsMissing
ProjectCharacteristicsInvalid
PanelOrientationIsPortrait
PanelOrientationIsLandscape
WrongDevicetype
NoRuntimeInstalled
RuntimeCorrupt
InvalidTransferChannel
InvalidBackupType
HMIUnknownError
```

4.19.10 FailsafeOperation

The `FailSafeOperation` enumeration indicates operations that are safety-relevant:

Description	Value
`Invalid`	0
`ResetToFactoryOperation`	0x2f161717
`FormatMCOperation`	0x46292728
`ProgramUpdateOperation`	0x43252224
`RestoreOperation`	0x45262427

4.19.11 FirmwareUpdateType

The `FirmwareUpdateType` specifies the type of firmware update to perform: a standard firmware update or a firmware update in two steps. The first step is the firmware update file download. The second step activates the firmware update on the device:

Description	Value
`Invalid`	0
`DownloadWithActivation`	1
`DownloadWithoutActivation`	2
`DownloadWithoutActivationInStopMode`	3

4.19.12 HMITransferChannel

The `HMITransferChannel` enumeration indicates the protocol used to communicate with an HMI device:

Description	Value
`Invalid`	-1
`PN_IE`	0
`Ethernet`	1

4.19.13 Language

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

```
English
German
French
Spanish
Italian
Chinese
```

4.19.14 OperatingState

This enumeration defines the possible states of the OperatingState property:

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

4.19.15 OperatingStateREQ

This enumeration defines the possible state transitions that can be requested on a call to the SetOperatingState (Page 164) method:

```
Stop
Run
```

4.19.16 ProgressAction

This enumeration defines the possible argument values that can be sent to a ProgressChangedEventHandler (Page 127):

```
Invalid
Connecting
Reconnecting
Disconnecting
Initializing
Updating
Processing
Downloading
Uploading
Deleting
Reseting
Rebooting
Verifying
Formatting
Refreshing
Finished
```

```
UpdatingFirmware
InstallingRuntime
InstallingAddOns
UninstallingAddOns
UpdatingProgram
Renaming
Creating
FileReplaceOrSkip
FileWriteProtected
FolderWriteProtected
```

4.19.17 ProtectionLevel

The `ProtectionLevel` enumeration gives the protection level of a CPU password:

```
Unknown
Failsafe
Full
Read
HMI
NoAccess
NoPassword
```

4.19.18 RedundancyRole

This enumeration defines the possible states of the RedundancyRole property:

```
Backup
NotValid
Primary
```

4.19.19 RemoteFolderType

The `RemoteFolderType` enumeration indicates the remote folder type:

Description	Value
`None`	0
`Recipe`	1
`Datalog`	2
`Files`	3

4.19.20 RemoteInterfaceType

This enumeration defines the possible states that can be returned from a call to the `InterfaceType` property on the IRemoteInterfaces (Page 172) interface:

```
None
Profinet
Profibus
ASi
```

4.19.21 ResetOptions

The ResetOptions enumeration contains the options for resetting to factory defaults and formatting a memory card :

```
DeleteConfigurationDataProtectionPassword
DeleteIPAddress
FormatMemoryCard
None
```

4.19.22 ScanErrorType

The `ScanErrorType` enumeration indicates the type of error returned by a device scan:

Description	Value
`Invalid`	-1
`Success`	0
`Error`	1
`Warning`	2
`Information`	3

4.19.23 SDCardType

The SDCardType enumeration contains the characteristics of the SD card:

```
NotPresent
ReadOnly
ReadWrite
Unknown
```

4.19.24 SNMPAuthAlgorithm

The `SNMPAuthAlgorithm` enumeration indicates the authorization algorithm for a SCALANCE device that uses an SNMP Version 3 profile:

Description	Value
`NotSupported`	-1
`MD5`	0
`SHA`	1

4.19.25 SNMPPrivAlgorithm

The `SNMPPrivAlgorithm` enumeration indicates the privacy algorithm for a SCALANCE device that uses an SNMP Version 3 profile:

Description	Value
`NotSupported`	-1
`DES`	0
`AES`	1

4.19.26 SNMPSecurityLevel

The `SNMPSecurityLevel` enumeration indicates the security level for a SNMP version 3 profile. A SCALANCE device that uses an SNMP version 3 profile has security level settings:

Description	Value
`NotSupported`	-1
`NoAuthNoPriv`	0
`AuthNoPriv`	1
`AuthPriv`	2

4.19.27 SNMPVersion

The `SNMPVersion` enumeration indicates the SNMP version number for SCALANCE devices:

Description	Value
`NotSupported`	-1
`Version1`	0
`Version2`	1
`Version3`	2

4.19.28 TimeFormat

The `TimeFormat` specifies a time format for time strings:

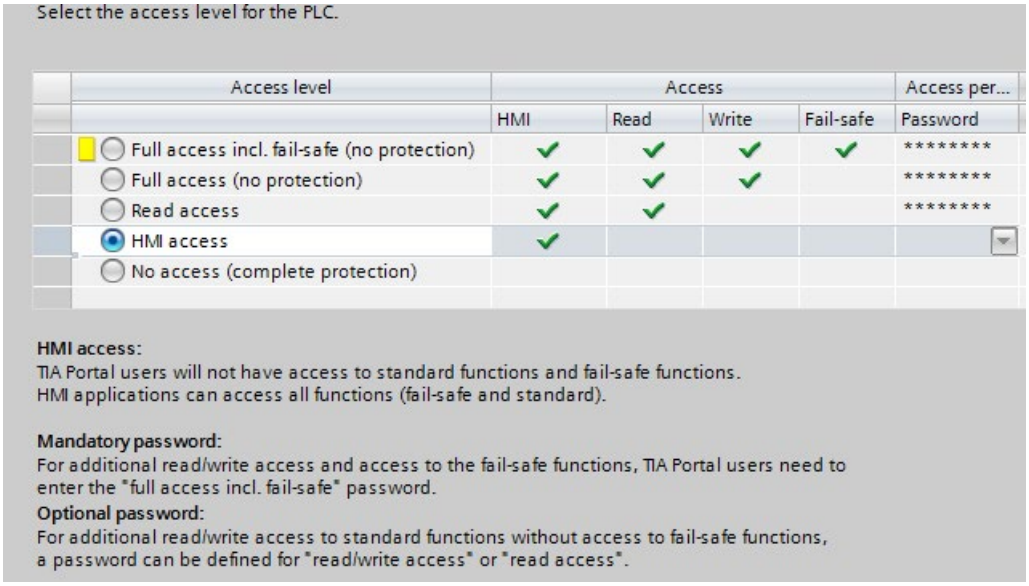
Description	Value
`Local`	0
`UTC`	1

UTC indicates Coordinated Universal Time.

4.20 CPU password access levels

A standard CPU has four password access levels. A fail-safe CPU has five levels.

In the TIA Portal, you can see the CPU access levels in the Protection & Security section of the CPU device Properties:



Device operations that require read access require any access level that provides the "Read" access.

Device operations that require write access require any access level that provides the "Write" access.

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

Password requirement for safety-relevant operations on F-CPUs

You can perform some operations on a fail-safe CPU using read access or write access. Safety-relevant operations require the safety password, which corresponds to "Full access incl. fail-safe (no protection)" access level.
The safety-relevant operations are:

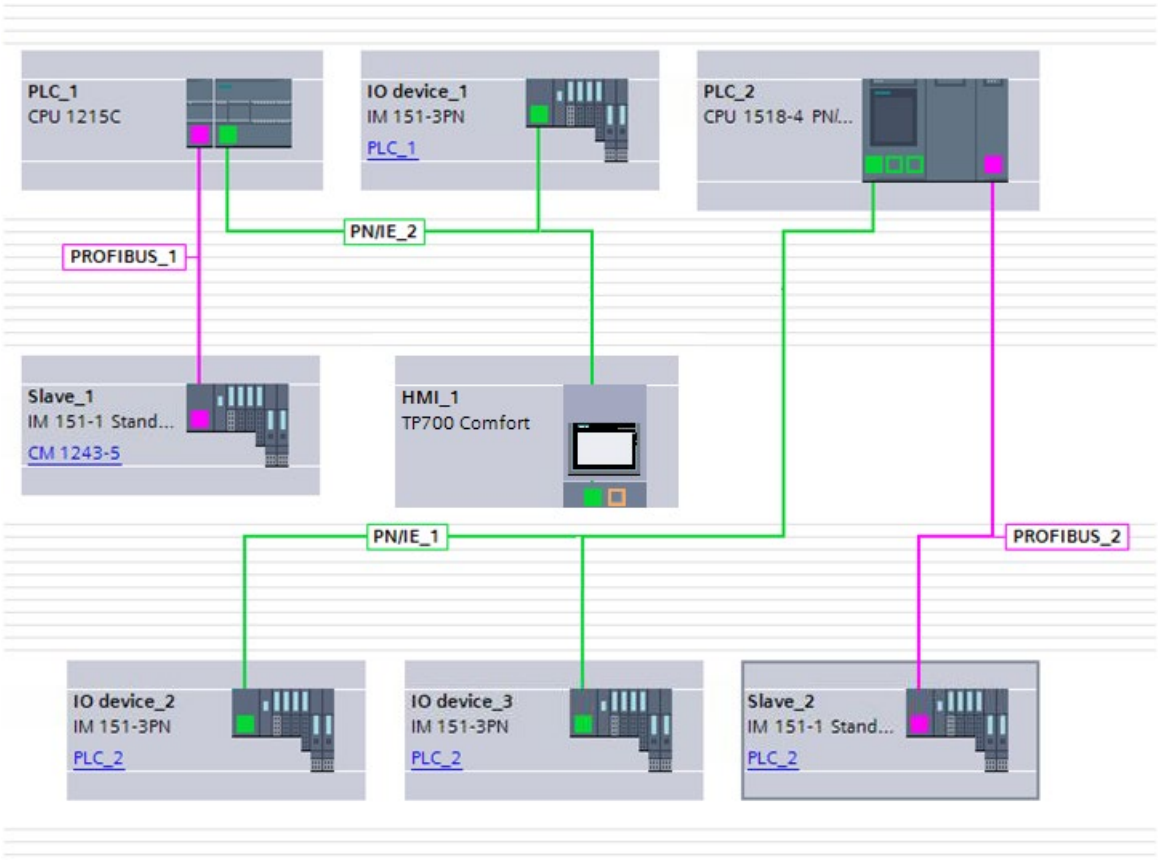
- Program Update
- Restore from Backup
- Reset to Factory Defaults
- Format Memory Card

If a fail-safe CPU does not use password protection, safety-relevant operations do not require the safety password to initiate the operation.

Throughout this user guide, the term "safety password" is the password with the "Full access incl. fail-safe (no protection)" access level.

4.21 Network example

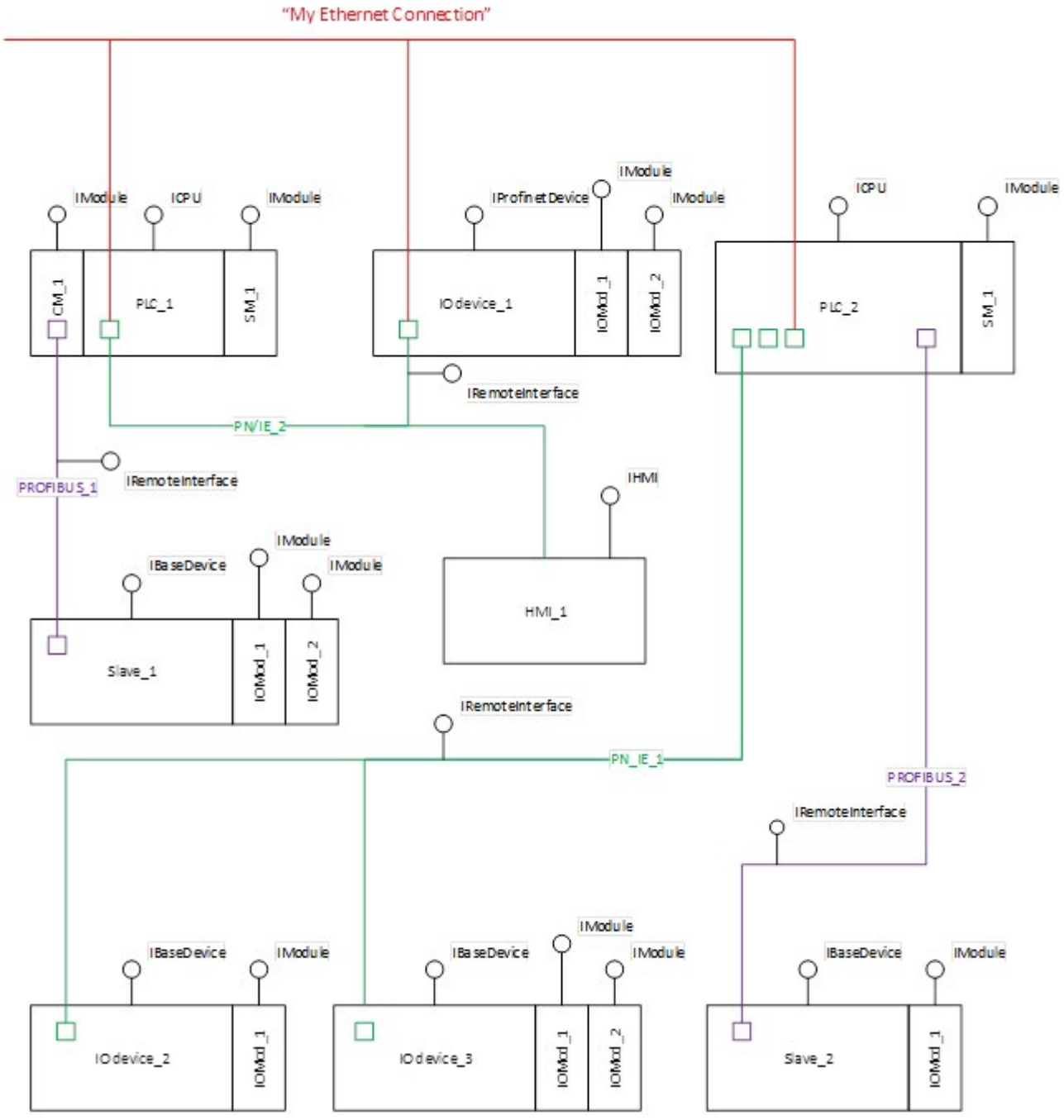
This example shows a TIA Portal network configuration and the API interfaces that represent the networked devices:



Assume that all the devices in the top row (PLC_1, IO device_1, and PLC_2) are connected to an external PROFINET network (not shown). They 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.

4.22 Reference information

4.22.1 About devices behind routers

A network scan cannot find devices behind routers. To insert a device behind a router, insert it by entering the device's IP address (Page 100).

Open port requirements for NAT routers

To use CPUs and HMIs behind NAT routers you must open the following ports, based on the device.

Device type	Ports to open
CPU	102 161
HMI with Ethernet transfer channel (Page 185)	5001 161
HMI with PN/IE transfer channel (Page 185)	102 161

Limitations of devices behind IP and NAT routers

The following DCP (Discovery and Configuration Protocol) operations are not supported for devices behind routers:

- Set IP Address (including Subnet and Gateway) (Page 120)
- Set PROFINET Name (Page 122)
- Identify device (Page 115)
- Reset Communication Parameters (Page 119)

If a device behind the router connects to a network through a CM (Communications Module) or CP (Communication Processor), then additional restrictions (Page 221) are in effect.

You must have an advanced or unexpired trial license to insert a device behind a NAT router.

The program update (Page 153) and restore (Page 158) operations have additional limitations for devices behind NAT routers.

4.22.2 About devices connected to CMs or CPs

The API can communicate with CPUs through a CM (Communications Module) or CP (Communications Processor) connection. To enable this type of communication, follow these steps:

1. From the TIA Portal, open the Device Configuration for the CM or CP in the STEP 7 project.
2. In the "Communication types" setting of the Properties, select "Enable online functions".
3. In the "SNMP" setting of the properties, select "Enable SNMP".
4. Download the STEP 7 project to the CPU.

You can connect your PROFINET network to the Ethernet interface of the CM or CP. The API will find the connected CPU on a network scan (Page 84).

Properties of a CPU connected through a CM or CP

When communicating to a CPU through the Ethernet interface of a CM or CP, the MAC address and IP address in the IProfinetDevice properties (Page 103) are the MAC address and IP address of the CM or CP. The serial number and all other data in the IProfinetDevice properties (Page 103) are the data from the CPU.

Limitations

The API prohibits the following device operations to CPUs connected through a CM or CP:

- Program Update (Page 153)
- Firmware Update (Page 108)
- Backup (Page 135)
- Restore (Page 158)
- Reset to Factory Defaults (Page 156)
- Format Memory Card (Page 148)

Index

A

Adding items to a collection (API), 100
API (application programming interface)
  architectural overview, 52
  designing UI, 58
  files and installation, 54
  getting started, 45
  provided safety features, 57
  S7 communications, 56
  version compatibility, 56

B

Backing up and restoring a CPU, 38
Backup (API method, ICPU interface), 135
Backup (API method, IHMI interface), 180
BackupType (API enumeration), 203

C

CardSortDirection (API enumeration), 203
CardSortItem (API enumeration), 204
CertificateStatusError (API enumeration), 204
Changing CPU operating mode, 22
Changing IP address, subnet, and gateway, 24
Classic CPUs, accessing through API, 176
Clear (API method), 103
Color coding safety fields in UI development, 60
  CPU device icon, 61
  CPU password, 63
  device data, 62
  program folder, 64
  program password, 65
ConfigurationDataProtection (API enumeration), 204
ConfirmationType (API enumeration), 204
Connecting to a CPU on the network, 16
CopyUserData (API method), 102
CPU
  password access level, 217
CreateConfigurationDataProtectionCard (API method), 136
CreateFirmwareUpdateCard (API method), 137
CreateProgramUpdateCard (API method), 138
Creating a custom application with the SDK, 11
Creating a project that uses the SIMATIC Automation Tool API, 14, 55
CriticalInternalErrorException, 202
CurrentNetworkInterface (API property), 84

D

DataChanged (API event), 126
DataChangedEventArgs (API class), 70
DataChangedType (API enumeration), 205
Decentralized modules, accessing through API, 172
DeleteConfigurationDataProtectionPassword (API method), 139
DeleteDataLog (API method), 140
DeleteRecipe (API method), 142
DetermineConfirmationMessage (API method), 143
DeviceFamily (API enumeration), 205
DiagnosticsItem (API class), 69
Displaying CPU information on user interface, 21
DownloadRecipe (API method), 147

E

EncryptedString (API class), 66
ErrorCode(API enumeration), 206
Example network, 218
Example program
  Backing up and restoring a CPU, 38
  Changing CPU IP address, subnet, and gateway, 24
  reading and writing PROFINET name, 27
  Resetting CPU to factory defaults, 29
  selecting CPU and changing operating mode, 13
  Updating CPU firmware, 31
Examples, SNMP profiles
  Version 1, 196
  Version 2, 198
  Version 3, 199
ExportDeviceDiagnostics (API method), 98
ExportDeviceInformation (API method), 97
ExportPCData (API method), 88
ExportProgressEventArgs (API class), 71

F

FailSafeOperation (API enumeration), 212
FilterByDeviceFamily (API method), 93
FilterOnlyCPUs (API method), 93

FindDeviceByIP (API method), 94
FindDeviceByMAC (API method), 95
FindDevicesBySerialNumber (API method), 91
FirmwareActivate (API method, IProfinetDevice interface), 112
FirmwareActivate (API method, IScalance interface), 189
FirmwareUpdate (API method, IHMI interface), 181
FirmwareUpdate (API method, IProfinetDevice interface), 108
FirmwareUpdate (API method, IScalance interface), 188
FirmwareUpdateType (API enumeration), 212
FormatMemoryCard (API method), 148

G

GetCommunicationsTimeout (API method), 86
GetCurrentDateTime (API method), 150
GetDiagnosticsBuffer (API method, ICPU interface), 151
GetDiagnosticsBuffer (API method, ICPUClassic interface), 177
GetEmptyCollection (API method), 87
GetEnumerator (API method), 91

H

Hamming codes, 66
HealthCheck (API class), 87
HealthCheckProgressEventArgs (API class), 72
HMITransferChannel (API enumeration), 212

I

IBaseDevice (API interface), 75
ICertificate (API interface), 82
ICertificateStore (API interface), 81
ICPU (API interface), 130
    feature flags, 134
    program update flags, 132
    properties, 131
    restore flags, 133
ICPUClassic (API interface), 176
Identify (API method), 115
Identifying CPUs, API, 130
IDiagnosticBuffer (API interface), 80
IHardware (API interface), 74
IHardwareCollection (API interface), 76
IHealthCheckErrorEvent (API class), 80
IHMI (API interface), 178
    feature flags, 180
    program update flags, 179
    properties, 179
    restore flags, 179
IModule (API interface), 129
IModule (API interface)
    Modules property and IModuleCollection class, 128
IModuleCollection (API interface), 76
IModuleCollection (API interface), using, 128
InsertDeviceByIP (API method), 100
InsertDeviceByMAC (API method), 101
Inserting items into a collection (API), 100
IProfinetDevice (API)
    collection items, 92
    properties, 103
IProfinetDeviceCollection (API class)
    [ ] property, 92
    count property, 91
    FindDevicesBySerialNumber method, 91
    GetEnumerator method, 91
    SortByIP method, 91
IProfinetDeviceCollection class (API)
    iterating collection items, 90
IRemoteFile (API interface), 73
IRemoteFolder (API interface), 73
IRemoteInterface (API interface), 74
IRemoteInterface (API), properties, 173
IResult (API interface), 80
IScalance (API interface), 186
    methods, 187
    properties, 187
IScanErrorCollection (API class), 76
IScanErrorEvent (API class), 79
IsFirmwareUpdate2StepAllowed (API method, IProfinetDevice interface), 116
IsFirmwareUpdateAllowed (API method, IProfinetDevice interface), 116
IsIPAddressOnNetwork (API method, IProfinetDevice interface), 116
ISNMPProfile (API interface)
    method, 190
    properties, 189
IsProfinetNameOnNetwork (API method, IProfinetDevice interface), 117
IsUploadServiceDataAllowed (API method, IProfinetDevice interface), 118

L

Language (API enumeration), 212
license terms and conditions, 55

M

MemoryReset (API method), 152

N

Network
example, 218
Network constructor (API), 82
Network interface, selecting, 15

O

OperatingState (API enumeration), 213
OperatingStateReq (API enumeration), 213

P

Password
CPU access level, 217
Programming guidelines using API, 58
ProgramUpdate (API method, ICPU interface), 153
ProgramUpdate (API method, IHMI interface), 182
ProgressAction (API enumeration), 213
ProgressChanged (API event), 127
ProgressChangedEventArgs (API class), 70, 71
ProtectionLevel (API enumeration), 214

Q

QueryNetworkInterfaceCards (API method), 83

R

ReadFromStream (API method), 96
Reading and writing a PROFINET name, 27
RedundancyRole (API enumeration), 214
RefreshStatus (API method), 119
RemoteFolderType (API enumeration), 214
RemoteInterfaces (API), 172
RemoteInterfaceType (API enumeration), 214
Remove (API method), 103
ResetCommunicationParameters (API method, IProfinetDevice interface), 120
ResetOptions (API enumeration), 215
Resetting a CPU to factory defaults, 29
ResetToFactoryDefaults (API method), 156
Restore (API method, ICPU interface), 158
Restore (API method, IHMI interface), 183
Result (API class), 67

S

S7-300 and S7-400, accessing through API, 176
Safety-relevant operations in UI design with the API, 57, 58
ScanErrorType (API enumeration), 215
ScanNetworkDevices (API method), 84
SDCardType (API enumeration), 215
Selecting an avalable network interface, 15
SetAuthAlgorithm (API method, SNMPProfile class), 195
SetBackupFile (API method, ICPU interface), 161
SetBackupFile (API method, IHMI interface), 184
SetBackupFilePassword (API method), 161
SetCommunicationsTimeout (API method), 85
SetConfigurationDataProtectionPassword (API method), 162
SetContextName (API method, SNMPProfile class), 194
SetCurrentDateTime (API method), 163
SetCurrentNetworkInterface (API method), 83
SetIP (API method), 120
SetOperatingState (API method), 164
SetPassword (API method), 164
SetPrivAlgorithm (API method, SNMPProfile class), 195
SetProfile (API method, IScalance interface), 187
SetProfileName (API method, SNMPProfile class), 191
SetProfinetName (API method), 122
SetProgramFolder (API method, ICPU interface), 166
SetProgramFolder (API method, IHMI interface), 185
SetProgramPassword (API method), 167
SetReadCommunity (API method, SNMPProfile class), 193
SetSecurityLevel (API method, SNMPProfile class), 194
SetServerIP (API method, SNMPProfile class), 192
SetServerPort (API method, SNMPProfile class), 192
SetSNMPVersion (API method, SNMPProfile class), 191
SetTransferChannel (API method, IHMI interface), 185
SetTrustCertificate (API method), 167
SetUserName (API method, SNMPProfile class), 194
SetWriteCommunity (API method, SNMPProfile class), 193
SNMP Version 1, API example, 196
SNMP Version 2, API example, 198
SNMP Version 3, API example, 199
SNMPAuthAlgorithm (API enumeration), 215
SNMPPrivAlgorithm (API enumeration), 216
SNMPProfile (API class)
methods, 191
properties, 190
SNMPSecurityLevel (API enumeration), 216
SNMPVersion (API enumeration), 216
SortByIP (API method), 91

T

TimeFormat (API enumeration), 216
Tutorial
creating a custom application, 11
example program to back up and restore a CPU, 38
example program to change CPU IP address, subnet, and gateway, 24
example program to read and write PROFINET name, 27
example program to reset CPU to factory defaults, 29
example program to select CPU and change operating mode, 13
example program to update CPU firmware, 31

U

Updating CPU firmware, 31
UploadDataLog (API method), 168
UploadRecipe (API method), 170
UploadServiceData (API method), 123

V

Validate (API method, ISNMPProfile interface), 190
ValidateIPAddressSubnet (API method), 124
ValidateNetworkInterface (API method), 87
ValidatePROFINETName (API method), 125
ValidateProgramFolder (API method), 171

W

WriteToStream (API method), 96