SIEMENS

SIMATIC

SIMATIC Automation Tool SDK V3.1 SP4 用户指南

编程手册

V3.1 SP4, 12/2019

A5E45594609-AC

法律资讯

警告提示系统

为了您的人身安全以及避免财产损失，必须注意本手册中的提示。人身安全的提示用一个警告三角表示，仅与财产损失有关的提示不带警告三角。警告提示根据危险等级由高到低如下表示。

⚠危险
表示如果不采取相应的小心措施，**将会**导致死亡或者严重的人身伤害。

⚠警告
表示如果不采取相应的小心措施，**可能**导致死亡或者严重的人身伤害。

⚠小心
表示如果不采取相应的小心措施，可能导致轻微的人身伤害。

注意
表示如果不采取相应的小心措施，可能导致财产损失。

当出现多个危险等级的情况下，每次总是使用最高等级的警告提示。如果在某个警告提示中带有警告可能导致人身伤害的警告三角，则可能在该警告提示中另外还附带有可能导致财产损失的警告。

合格的专业人员

本文件所属的产品/系统只允许由符合各项工作要求的**合格人员**进行操作。其操作必须遵照各自附带的文件说明，特别是其中的安全及警告提示。 由于具备相关培训及经验，合格人员可以察觉本产品/系统的风险，并避免可能的危险。

按规定使用 Siemens 产品

请注意下列说明：

⚠警告
Siemens 产品只允许用于目录和相关技术文件中规定的使用情况。如果要使用其他公司的产品和组件，必须得到 Siemens 推荐和允许。正确的运输、储存、组装、装配、安装、调试、操作和维护是产品安全、正常运行的前提。必须保证允许的环境条件。必须注意相关文件中的提示。

商标

所有带有标记符号 ® 的都是 Siemens AG 的注册商标。本印刷品中的其他符号可能是一些其他商标。若第三方出于自身目的使用这些商标，将侵害其所有者的权利。

责任免除

我们已对印刷品中所述内容与硬件和软件的一致性作过检查。然而不排除存在偏差的可能性，因此我们不保证印刷品中所述内容与硬件和软件完全一致。印刷品中的数据都按规定经过检测，必要的修正值包含在下一版本中。

Siemens AG
Digital Industries
Postfach 48 48
90026 NÜRNBERG
德国

A5E45594609-AC
℗ 12/2019 本公司保留更改的权利

Copyright © Siemens AG 2019.
保留所有权利

目录

前言

1

Siemens 为其产品及解决方案提供了工业信息安全功能，以支持工厂、系统、机器和网络的安全运行。

为了防止工厂、系统、机器和网络受到网络攻击，需要实施并持续维护先进且全面的工业信息安全保护机制。Siemens 的产品和解决方案构成此类概念的其中一个要素。

客户负责防止其工厂、系统、机器和网络受到未经授权的访问。只有在有必要连接时并仅在采取适当安全措施（例如，防火墙和/或网络分段）的情况下，才能将该等系统、机器和组件连接到企业网络或 Internet。

关于可采取的工业信息安全措施的更多信息，请访问 (https://www.siemens.com/industrialsecurity)。

Siemens 不断对产品和解决方案进行开发和完善以提高安全性。Siemens 强烈建议您及时更新产品并始终使用最新产品版本。如果使用的产品版本不再受支持，或者未能应用最新的更新程序，客户遭受网络攻击的风险会增加。

要及时了解有关产品更新的信息，请订阅 Siemens 工业信息安全 RSS 源，网址为 (https://www.siemens.com/industrialsecurity)。

1.1 SIMATIC Automation Tool SDK 概述

SIMATIC Automation Tool 软件开发工具包 (SDK) 使您能够使用 SIMATIC Automation Tool 应用程序编程接口 (API) 创建应用程序。您可以将您的应用程序（包括 API 软件）分发给第三方。您的用户可使用您的自定义应用程序执行多种设备自动化任务。您的用户无需从西门子获得许可证即可使用您的自定义应用程序。西门子将提供 Windows 安装程序包，您可以使用该程序包为自定义应用程序创建安装程序。安装程序包括分发软件和与 S7 设备通信所需的所有组件。

本用户指南提供以下内容：

- 使用示例.NET 解决方案创建自定义应用程序的教程 (页 12)

  SIMATIC Automation Tool SDK 以 C# 语言提供执行多种设备操作的完整示例程序。示例程序解决方案包括简单的用户界面，其中包含用于操作用户控件 API 方法调用。用户指南的教程部分展示了如何在使用 API 的 .NET 框架中设置应用程序。然后，将在示例程序中逐步展示如何对 API 方法调用进行编程以执行设备操作。完整的示例程序解决方案包含在您的安装文件夹中。

- 完整的 API 参考 (页 46)

  虽然示例程序说明了许多可用的 API 方法，但它仅使用 API 提供的全套功能的一部分。此参考部分包括您可以使用的所有类别、接口、方法、特性、枚举类型。在许多情况下，API 参考包括特定编程任务的代码示例。

1.2 SDK 设备目录

SIMATIC Automation Tool SDK 安装程序用于安装设备目录。设备目录是 Microsoft Excel 文件，其中列出了 SDK 应用程序可访问的设备。可以查看每台设备支持的固件版本以及支持的操作。

安装程序会在“文档”(Documents) 文件夹的 SAT SDK V3.1 SP4\Bin\Catalog 文件夹中安装设备目录。

Windows 7

在 Windows 资源管理器中，“文档”(Documents) 文件夹位于“库”(Libraries) 文件夹下。

Windows 10

在文件管理器中，“文档”(Documents) 文件夹位于“快速访问”(Quick Access) 下。

教程：使用 SIMATIC Automation Tool SDK 创建自定义应用程序

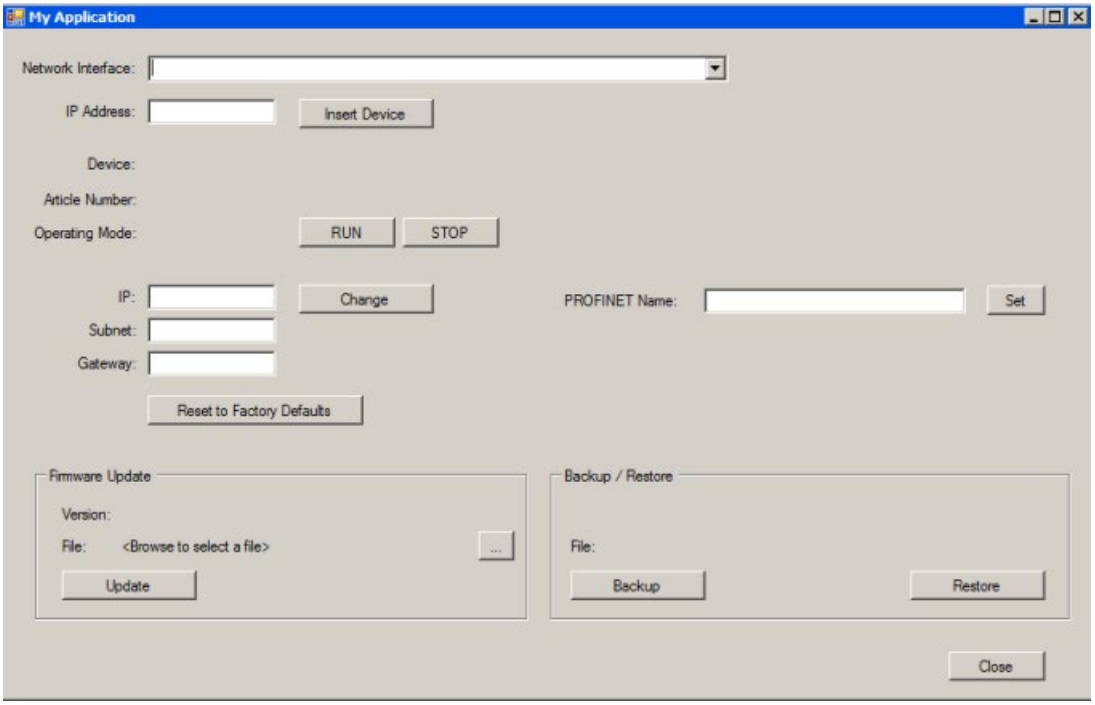
2

2.1 示例应用程序简介

在本教程中，您的目标是创建使用 SIMATIC Automation Tool API 中的某些接口和方法的 Windows 应用程序。继续学习本教程后，将逐步向解决方案中添加控件和逻辑。您的解决方案将使用几种典型的 API 接口和相关方法。该应用程序允许您的用户在您的设备网络中处理单个 CPU。您将开发一款使用户能够执行以下任务的软件：

- 选择用于 CPU 通信的网络接口
- 按 IP 地址插入 CPU
- 显示 CPU 的识别信息
- 更改 CPU 的操作模式
- 更改 CPU 的 IP 地址、子网和网关
- 设置 CPU 的 PROFINET 名称
- 将 CPU 复位为出厂设置
- 更新 CPU 的固件
- 将 CPU 备份至文件
- 将备份文件恢复到 CPU

Windows 应用程序的用户界面将类似于以下示例：



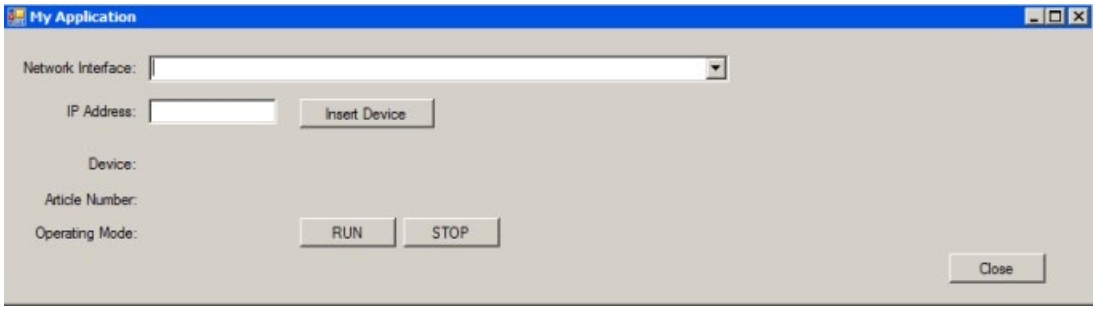
西门子建议您在执行编程任务时，不要查看安装文件夹中提供的 SDK 解决方案。按照本教程的指导，了解如何使用 API。在开发工作应用程序之后，您可以查看提供的解决方案以进行比较。

为了最大限度地减少要学习的代码量，提供的解决方案对每个 API 操作实施了最少的结果检查。在实际操作中，应使用良好的防御性编程技巧并检查操作结果。对应用程序进行编程，使其在操作成功时继续执行，并适当地处理错误。示例程序和 API 方法调用使用 Result 对象返回操作状态。在程序逻辑中调用 API 方法后，检查 Result 对象的值。此外，为您的用户添加可见反馈，例如长时间操作的等待光标或进度指示条。

请注意，示例应用程序的用户界面仅使用英语。另请注意，示例应用程序仅运行全套 API 功能 (页 46)的一部分。在您熟悉使用 API 编程后，您可以使用创建其它应用程序所需的任何 API 接口、类别和方法。

2.2 示例 1：从网络接口选择 CPU 并更改 CPU 操作模式

本章将指导您编写示例应用程序的控件，如下所示：



学完本章后，您将可以使用 API 提供以下功能：

- 用户用于选择网络接口的选择器 (页 16)
- 按 IP 地址插入 CPU 的方法 (页 18)
- 输入的 CPU 的设备信息 (页 22)
- 用于更改 CPU 操作模式的按钮 (页 24)

先决条件

要开发您的应用程序，必须满足以下先决条件。运行您的应用程序的用户也必须满足这些先决条件：

- 您通过网络接口将 CPU 连接到您的编程设备。
- 您知道 CPU 的 IP 地址。
- 您可以对网络上的 CPU 执行 ping 操作并验证它是否响应。

2.2.1 创建使用 API 的项目

您必须使用 Microsoft Visual Studio 2015 Update 3 开发使用 SIMATIC Automation Tool API (页 46) 的应用程序。您的应用程序和 SIMATIC Automation Tool API .dll 文件必须位于同一文件夹中。

创建使用 API 的项目

要创建可以使用 SIMATIC Automation Tool API 的项目，请按照下列步骤操作：

1. 在 Visual Studio 中，创建一个新项目，即 Visual C# Windows 窗体应用程序。将其命名为“我的应用程序”(My Application)。
2. 从“解决方案平台”(Solutions Platforms) 下拉列表中，选择“组态管理器”(Configuration Manager)。
3. 在“组态管理器”(Configuration Manager) 中，单击“平台”(Platform) 下拉列表并创建一个新平台。
4. 从“新建项目平台”(New Project Platform) 对话框，为新平台选择“x64”。单击“确定”(OK)，关闭“组态管理器”(Configuration Manager)。
5. 在项目特性中，将目标框架设置为 .NET Framework 4.6.2。

说明

SIMATIC Automation Tool API 使用 .NET Framework 4.6.2。如果您的编程设备上没有此框架，则安装程序会安装此框架。

设置项目以使用 API

要设置项目以使用 API，请按照下列步骤操作：

1. 运行 SAT SDK File Extractor，将 API 文件和文件夹复制到默认位置。您可以从“开始”(Start) 菜单中访问 Siemens Automation 组中的 SAT SDK File Extractor。
2. 打开解压缩文件的位置，并将“Bin”文件夹的内容复制到 \bin\x64\Debug 文件夹中。
3. 在项目中添加对 `bin\x64\Debug\SIMATICAutomationToolAPI.dll` 的引用。
4. 在文件 `Form1.cs` 中，添加以下语句：

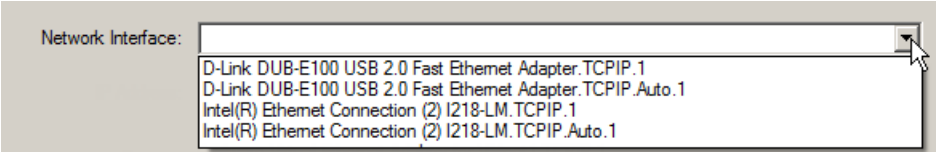
   ```
   using Siemens.Automation.AutomationTool.API;
   ```

   应用程序可通过此语句使用 API dll。您将通过 API 类别和接口在程序中创建对象。
5. 在窗体中添加“关闭”(Close) 按钮，以便用户可以关闭应用程序。
6. 构建您的解决方案并验证它是否成功构建。

现在已准备好使用 API 开发应用程序。

2.2.2 接受来自可用网络接口的选定接口

您的应用程序必须确定编程设备可用的网络接口。借助可用接口，显示可用接口，并允许用户选择一个用于通信的接口。



为用户提供一种选择网络接口的方法

要读取和显示网络接口，并使用户能够选择接口，请按照以下这些步骤操作：

1. 使用文本标签（如“网络接口”）向窗体添加 ComboBox。
2. 将 ComboBox 命名为“SelectNetworkInterface”。
3. 向 `Form1.cs` 中的 `public partial class Form1` 类别输入网络的全局变量：

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

请注意，“Network (页 75)”是 API (页 57) 提供的类别。

4. 在 `Form1` 类别中为全局辅助方法设置一个区域：

```
#region 辅助方法

#endregion
```

5. 在辅助方法的 `#region` 中添加一个全局空方法：

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

稍后您将向此方法添加代码。每当用户选择网络接口时，应用程序将清除任何现有的 CPU 状态信息。

6. 在 `Form1` 的构造函数中，添加以下代码以读取可用的网络接口并在 ComboBox 中加载它们：

```
List<string> NetworkInterfaces;

Result res = Net.QueryNetworkInterfaceCards(out NetworkInterfaces);

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

7. 双击窗体上的 ComboBox 控件以添加允许用户选择网络接口的方法。在方法中插入如下所示的代码：

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

此时，编译您的项目并验证它是否已成功编译。测试下拉框以确保它显示可用的网络接口。确认可以选择每个网络接口。根据需要添加其它错误检查。

如果您对自己的工作感到满意，请转到通过 IP 地址连接到网络上的 CPU (页 18)。

2.2.3 通过 IP 地址连接到网络上的 CPU

如果网络接口使用 S7-1200 或 S7-1500 CPU 连接到网络，则下一步是使用户能够连接到网络上的 CPU。随后，该 CPU 将成为设备指令的连接设备。如果您根据提供的示例程序实现您的应用程序，则必须执行以下任务：

1. 接受并验证 IP 地址的输入内容。
2. 将 IP 地址的设备插入设备集合。在此示例中，设备集合将是单个设备的集合。
3. 从与输入的地址对应的网络设备中读取数据。
4. 确认 IP 地址处的设备是 S7-1200 或 S7-1500 CPU。
5. 显示 CPU 的设备信息。

本章节将指导您完成每项任务的窗体设计和编程。

说明

此任务序列假定之前的每项任务都已成功。在您的应用程序中，执行验证和错误检查。只有前一项任务成功后，才能继续执行下一任务。

提供输入 IP 地址的文本框

用户在文本框中输入 CPU 的 IP 地址。要为您的用户提供此文本框，请按照以下步骤操作：

1. 在您的窗体上放置一个带有相应标签的文本框控件，例如“IP 地址：”(IP Address:)。
2. 将文本框命名为“InsertDeviceIPAddress”。
3. 确保文本框足够宽，能够容纳 IPv4 地址。
4. 将以下全局声明添加到 `Form1.cs` 中的 `public partial class Form1` 类别。于在 接受来自可用网络接口的选定接口 (页 16) 中添加的公共网络声明后添加这些声明：

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

请注意，“IProfinetDeviceCollection (页 85)”是 API 提供的类别，“ICPU (页 123)”是接口。本教程应用程序在所有示例中均使用 `Devices` 集合和 `CurrentCPU` 对象。

5. 在辅助方法区域中，添加名为 `ParseIP` 的 IP 地址解析器方法。`ParseIP` 会将基于文本的 IP 地址转换为无符号整数：

```
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;
   }
}
```

实现处理输入的 IP 地址的按钮

您需要为用户提供输入 IP 地址后可单击的按钮。该按钮尝试通过 IP 地址连接到网络上的 CPU。要实现此按钮，请按以下步骤操作：

1. 在窗体上放置一个按钮并将其命名为“`InsertDevice`”。
2. 为按钮编写单击事件：

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

3. 在辅助方法的 `#region` (页 16) 中，添加一个全局空方法：

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

稍后您将向此方法添加代码。

4. 在 `InsertDevice_Click` 方法内，添加代码以清除任何现有设备数据并清除 CPU 状态：

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

5. 调用您的 `ParseIP` 方法，将输入的 IP 地址从文本转换为 32 位大端无符号整数：

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

在整个应用程序中，根据需要添加错误检查。为简单起见，本教程省略了典型的错误处理代码。

使用 IP 地址以连接至 CPU

现在您的窗体上有一个带标签的文本框和相应的按钮，下面编写代码以查找网络上的 CPU。找到 CPU 后，您的应用程序可以从 CPU 读取设备信息。

API 提供了设备集合类别，`IProfinetDeviceCollection` (页 85)。您将把 CPU 插入集合 (`Devices`)，以便可以在 CPU 上执行后续设备操作。`Network` (页 75) 类别的 `ScanNetworkDevices` (页 78) 方法用于将 CPU 输入集合。

要编写 `InsertDevice_Click` 方法以将 CPU 插入设备集合，请按照以下步骤操作：

1. 创建一个设备集合：

```
IScanErrorCollection scn = Net.ScanNetworkDevices(out Devices);
```

2. 忽略网络扫描的内容，并使用 `InsertDeviceByIP` 方法将您的设备插入集合：

```
Devices.Clear();
Result res = Devices.InsertDeviceByIP(0, ipAddress);
```

3. 确认设备是 S7-1200 或 S7-1500 CPU：

```
res = Devices[0].RefreshStatus();
if (Devices[0].Family != DeviceFamily.CPU1200 && Devices[0].Family !=
DeviceFamily.CPU1500)
   {
      MessageBox.Show("错误：插入的设备不是 CPU", "Insert Device",
MessageBoxButtons.OK);
      return;
   }
```

4. 将设备转换为 CPU（`ICPU` (页 123) 类别）并保存引用。应用程序通过 `CurrentCPU` 在 CPU 上执行所有设备操作：

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

5. 更新设备的状态：

```
UpdateCPUStatus();
```

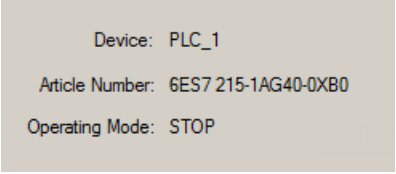
6. 同样，添加对返回的操作结果的适当错误检查。

7. 构建解决方案并验证代码是否可编译。

在设备集合中成功插入 S7-1200 或 S7-1500 CPU 后，您的下一个任务是 显示 CPU 信息 (页 22)。

2.2.4 显示 CPU 信息

此时在应用程序中，您已实现连接到网络上 CPU 的控件。既然您的应用程序已成功建立此连接，接下来添加控件和逻辑以显示有关设备的信息：



添加 CPU 设备信息的标签

将标签控件添加到您的窗体中，并对每个标签进行描述：

窗体上的文本	标签控件名称
设备：	DeviceName
订货号：	ArticleNumber
操作模式：	OperatingMode

将每个标签控件的 AutoSize 设置为 FALSE，并将宽度设置得足够大，以容纳每个字段的最大预期文本。

编写 CPU 状态方法

在之前的任务中，您创建了两个空帮助方法：`ClearCPUStatus` (页 16) 和 `UpdateCPUStatus` (页 18)。您的下一个任务是为这些方法添加代码以获取 CPU 设备信息：

1. 将以下代码插入 `ClearCPUStatus()`：

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

   请注意，当用户尚未连接到 CPU 时，每个字段最初显示“...”。

2. 在 `UpdateCPUStatus()` 中，插入以下代码：

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

   回想一下，`CurrentCPU` 包含您在通过 IP 地址连接到网络上的 CPU (页 18)中从设备读取的设备信息。当用户单击“插入设备”(Insert Device) 按钮时，`InsertDevice_Click` 方法将调用 `ClearCPUStatus` 方法和 `UpdateCPUStatus` 方法。

3. 在继续下一步之前编译并测试您的程序：更改 CPU 的操作模式 (页 24)

2.2.5 更改 CPU 的操作模式

到目前为止，您已完成以下任务：

- 接受来自可用网络接口的选定接口 (页 16)
- 通过 IP 地址连接到网络上的 CPU (页 18)
- 显示 CPU 信息 (页 22)

此示例程序的下一个任务是添加按钮控件和逻辑以更改操作模式：

说明

此示例中的操作需要满足以下条件：

- CPU 没有密码保护
- CPU 中具有用户程序。（未复位为出厂设置。）

编写 RUN 按钮

要编写 RUN 按钮，请按以下步骤操作：

1. 向窗体添加一个名为“RunButton”的按钮。
2. 添加按钮单击方法并使用以下代码进行填充：

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

编写 STOP 按钮

要编写 STOP 按钮，请按以下步骤操作：

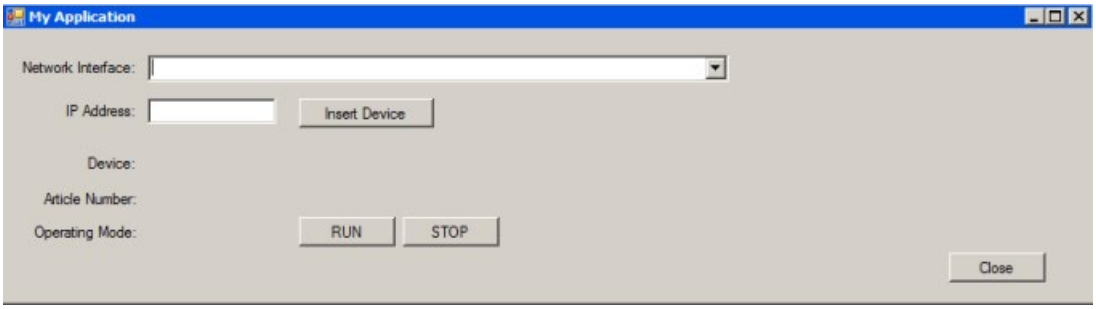
1. 向窗体添加另一个名为“StopButton”的按钮。
2. 添加按钮单击方法并使用以下代码进行填充：

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

使用这两种按钮方法，检查操作的结果。

2.2.6 测试示例 1

编译并测试您的应用程序。如果您按照此示例中的步骤操作，则已创建如下所示的应用程序：



验证以下用户操作是否正确运行：

- 单击网络接口的下拉按钮可显示编程设备的可用网络接口。
- 选择一个网络接口并填写“网络接口”(Network Interface) 字段。
- 插入 IP 地址，单击“插入设备”(Insert Device)，填写设备信息字段。在此示例中，IP 地址必须对应于所选网络接口上的 S7-1200 或 S7-1500 CPU。
- 单击“运行”(RUN) 按钮可使 CPU 进入或保持 RUN 模式。
- 单击“停止”(STOP) 按钮可使 CPU 进入或保持 STOP 模式。

此外，测试各种错误情况，例如：

- 尝试在选择网络接口之前插入设备
- 尝试插入不是 S7-1200 或 S7-1500 CPU 的设备
- 在 IP 地址字段中使用无效文本

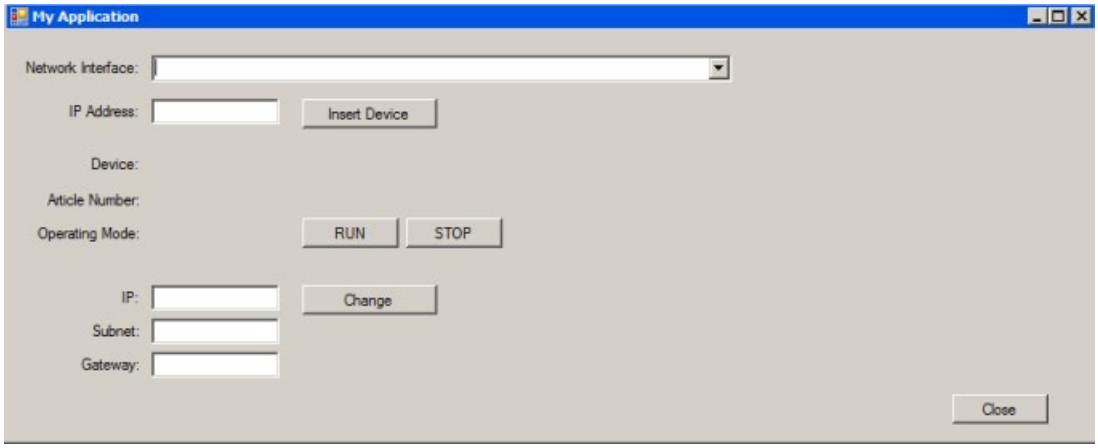
您可以按照意愿执行其它测试。根据需要优化您的程序，以提供强大的错误处理功能。如果您对示例 1 的结果感到满意，可转到示例 2：读取并更改 CPU 的 IP 地址、子网和网关 (页 26)。

2.3 示例 2：读取并更改 CPU 的 IP 地址、子网和网关

示例 2 将扩展您在示例 1 (页 14) 中开发的应用程序。示例 2 将添加控件和逻辑，以显示所连接 CPU 的以下 IP 协议信息：

- IP 地址 (IP)
- 子网
- 网关

对于每个字段，示例 2 还允许用户输入新值并更改 CPU 中的值。



窗体上的三个文本框将显示连接的 CPU 的 IP 协议值。您的用户成功连接到 CPU 设备 (页 18)后，三个文本框将显示 CPU 的 IP 协议信息。这三个文本框也是可编辑的字段。您的用户可以为以下 IP 协议值输入新值：

- IP 地址 (IP)
- 子网
- 网关

添加 IP 协议值的文本框

要为 IP 协议字段添加文本框并编写逻辑，请按照下列步骤操作：

1. 向您的窗体添加三个 TextBoxes，其名称和标签描述如下：

窗体上的文本	文本框名称
IP：	SuiteIp
子网：	SuiteSn
网关：	SuiteGw

2. 将以下代码添加到现有 `ClearCPUStatus` (页 16) 辅助方法，清除 IP 协议文本字段：

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

3. 将以下代码添加到现有 `UpdateCPUStatus` (页 18) 辅助方法，将当前连接的 CPU 的值加载到 IP 协议文本字段：

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

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

添加按钮以更改 IP 协议值

要添加和编写用于更改 IP 协议字段的按钮，请按照下列步骤操作：

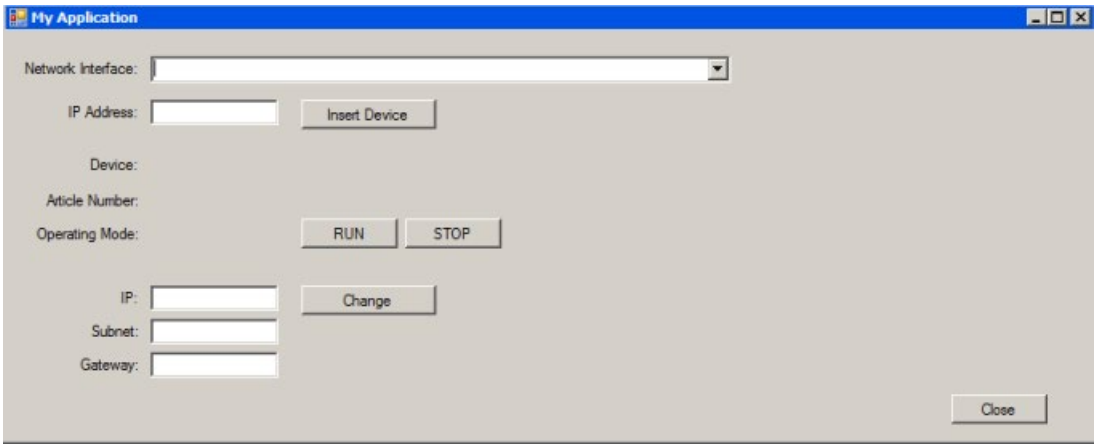
1. 向窗体添加一个名为“ChangeIpSuite”的按钮。
2. 为按钮文本添加“更改”(Change) 文本。
3. 为“更改”(Change) 按钮添加按钮单击事件并插入以下代码：

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

if (newIp == 0xffffffff || newSn == 0xffffffff || newGw == 0xffffffff)
{
   MessageBox.Show("错误：指定的 IP、子网或网关地址无效", "Change IP Suite",
MessageBoxButtons.OK);
   return;
}

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

示例程序未提供检查 IP 地址、子网和网关是否相互兼容的逻辑。您可以添加此逻辑以使您的解决方案更完善。

2.3.1 测试示例 2

编译并测试您的应用程序。如果您按照此示例中的步骤操作，则已创建如下所示的应用程序：



验证以下用户操作是否正确运行：

- 输入网络上 CPU 的 IP 地址并单击“插入设备”(Insert Device)，填写 IP 协议信息字段。在此示例中，IP 地址必须对应于所选网络接口上的 S7-1200 或 S7-1500 CPU。
- 输入有效的 IP 地址并单击“更改”(Change) 按钮后，CPU IP 地址会更改。关闭应用程序并重新打开，进行测试。通过新 IP 地址插入 CPU，并验证应用程序是否能够找到 CPU 并显示设备信息和 IP 协议字段。
- 按照同样的操作步骤输入并更改子网。
- 按照同样的操作步骤输入并更改网关。

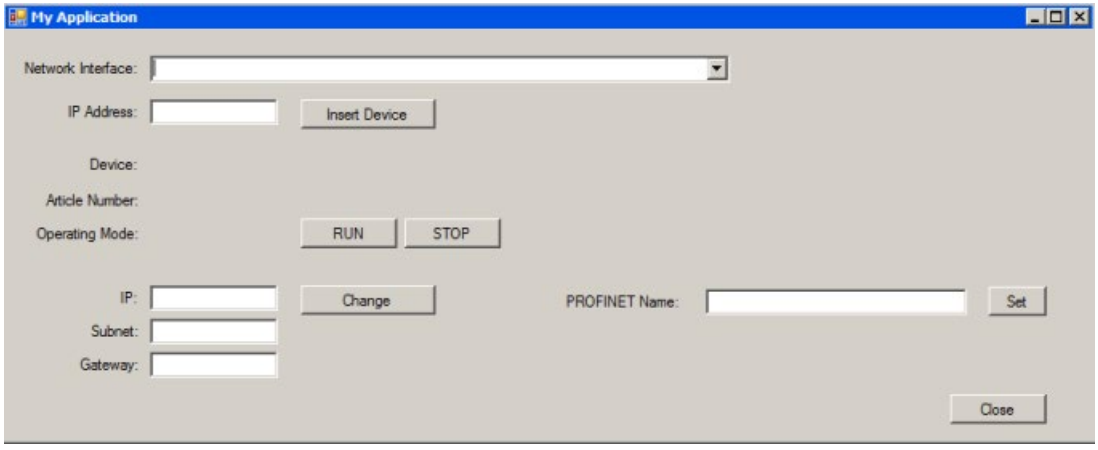
此外，测试各种错误或无效使用情况，例如：

- 试图更改为非有效格式的 IP 地址。
- 在未连接 CPU 的情况下单击“更改”(Change) 按钮。

您可以按照意愿执行其它测试。根据需要优化您的程序，以提供强大的错误处理功能。如果您对结果感到满意，可转到示例 3：读取和写入 CPU 的 PROFINET 名称 (页 30)

2.4 示例 3：读取和写入 CPU 的 PROFINET 名称

示例 3 将扩展您在示例 2 (页 26) 中开发的应用程序。示例 3 将添加 PROFINET 名称的文本框。程序使用此文本框显示 PROFINET 名称并接受用于设置 PROFINET 名称的用户输入。“设置”(Set) 按钮将连接的 CPU 的 PROFINET 名称设置为 PROFINET 名称文本框中的名称。



说明

此示例中的操作需要满足以下条件：

- CPU 没有密码保护
- CPU 中具有用户程序。（未复位为出厂设置。）
- CPU 的 PROFINET 接口 IP 协议的设备组态指定“在设备中直接设置 IP 地址”(IP address is set directly at the device)。根据使用的 TIA Portal 版本，此选项可能具有其它名称：
  - 在设备中设置 IP 地址 (Set IP address on the device)
  - 使用不同方法设置 IP 地址 (Set IP address using a different method)

添加逻辑以显示和设置 PROFINET 名称

要添加和编写 PROFINET 名称文本框和设置按钮，请按照下列步骤操作：

1. 向窗体添加一个文本框并将其命名为“ProfinetName”。
2. 在文本框旁边添加一个按钮，并将其命名为“SetProfinetName”。
3. 为 SetProfinetName 按钮添加按钮单击事件并插入以下代码：

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

4. 将以下代码添加到现有 `ClearCPUStatus` (页 16) 辅助方法：

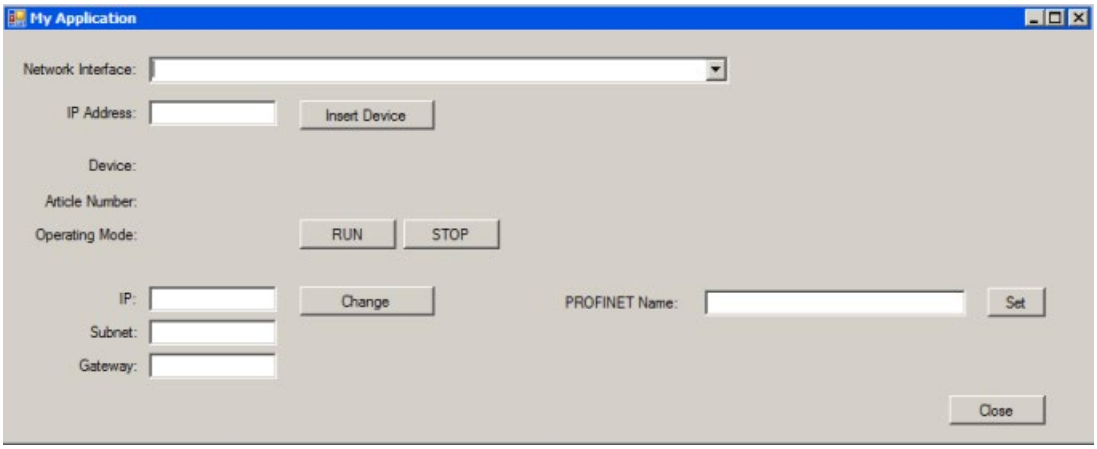
```
ProfinetName.Text = string.Empty;
```

5. 将以下代码添加到现有 `UpdateCPUStatus` (页 18) 辅助方法：

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

2.4.1 测试示例 3

编译并测试您的应用程序。如果您按照此示例中的步骤操作，则已创建如下所示的应用程序：



验证以下操作是否正确运行：

- 连接的 CPU 的 PROFINET 名称正确显示在 PROFINET 名称字段中。
- 您可以编辑和设置 CPU 的新 PROFINET 名称。

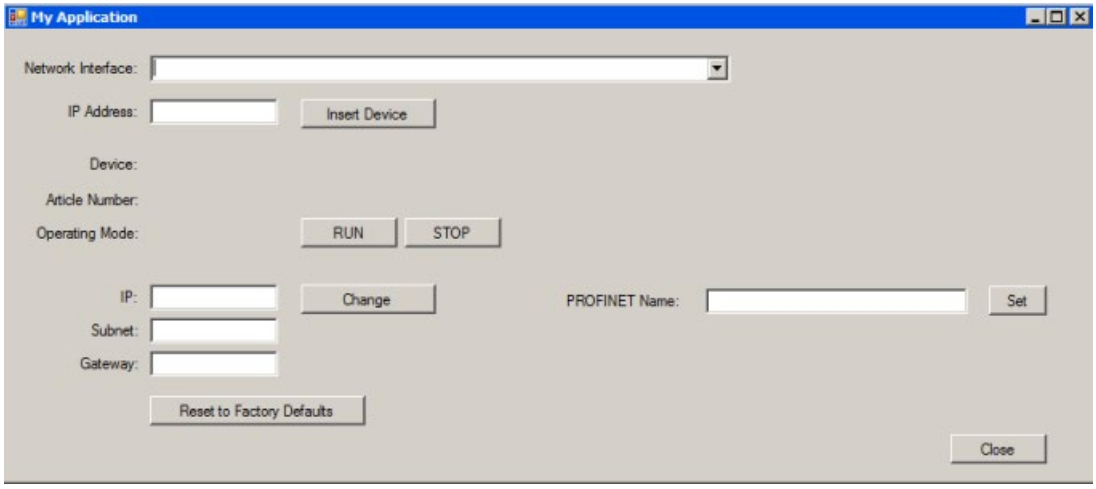
此外，测试各种错误情况，例如：

- 如果尝试将 PROFINET 名称设置为空，会导致出现一条消息。
- 如果尝试设置不遵循 PROFINET 命名规则的 PROFINET 名称，会导致出现一条消息。

您可以按照意愿执行其它测试。根据需要优化您的程序，以提供强大的错误处理功能。如果您对结果感到满意，可转到示例 4：将 CPU 复位为出厂设置 (页 32)

2.5 示例 4：将 CPU 复位为出厂设置

示例 4 继续扩展您在示例 3 (页 30) 中开发的应用程序。示例 4 将添加一个按钮，供用户将连接的 CPU 复位为出厂设置。由于将 CPU 复位为出厂设置会从 CPU 中删除用户程序和所有相关数据，因此应用程序会通知用户并在继续之前请求确认。



说明

在此示例中，请注意以下信息：

- 为了能够将 CPU 复位为出厂设置，CPU 不能受密码保护。
- “复位为出厂设置”操作会保留 CPU 的 IP 地址。

添加“复位为出厂设置”(Reset to Factory Defaults) 按钮

要添加“复位为出厂设置”(reset to factory defaults) 按钮并编写相应的逻辑，请按照下列步骤操作：

1. 向窗体添加一个按钮并将其命名为“ResetToFactoryDefaults”。
2. 为按钮文本添加“复位为出厂设置”(Reset to Factory Defaults) 文本。
3. 为按钮添加按钮单击事件并添加以下代码：

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

if (MessageBox.Show("复位为出厂设置是一项安全相关的操作，您是否希望继续？","Reset to
Factory Defaults", MessageBoxButtons.YesNo) == DialogResult.Yes)
{
   CurrentCPU.SelectedConfirmed = true;
}
else
{
   return;
}

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

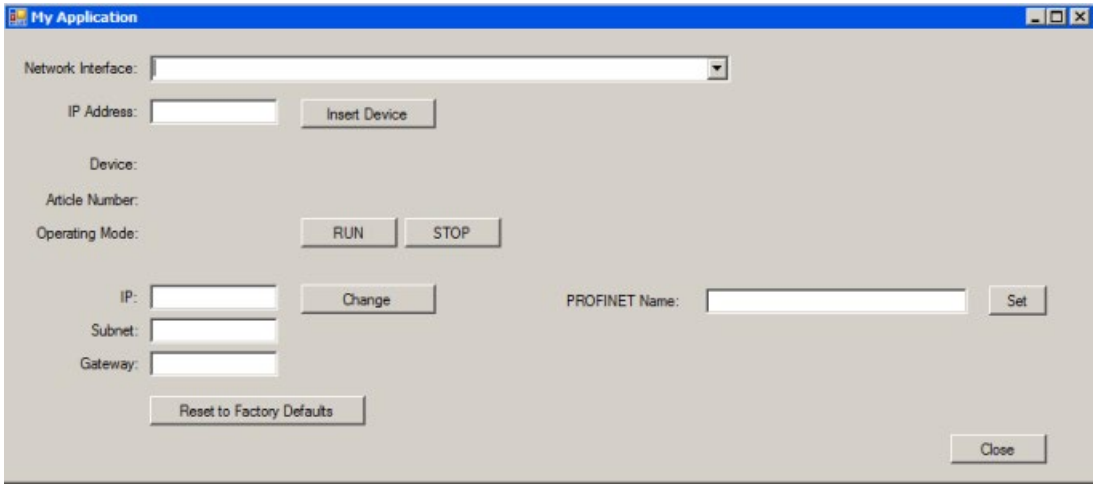
4. 根据需要添加结果检查。

说明

将故障安全 CPU 复位为出厂设置是一项安全相关操作。请参见安全相关操作的用户界面编程指南 (页 48)中的指南。

2.5.1 测试示例 4

编译并测试您的应用程序。如果您按照此示例中的步骤操作，则已创建如下所示的应用程序：



确认单击“复位为出厂设置”(Reset to Factory Defaults) 按钮具有以下效果：

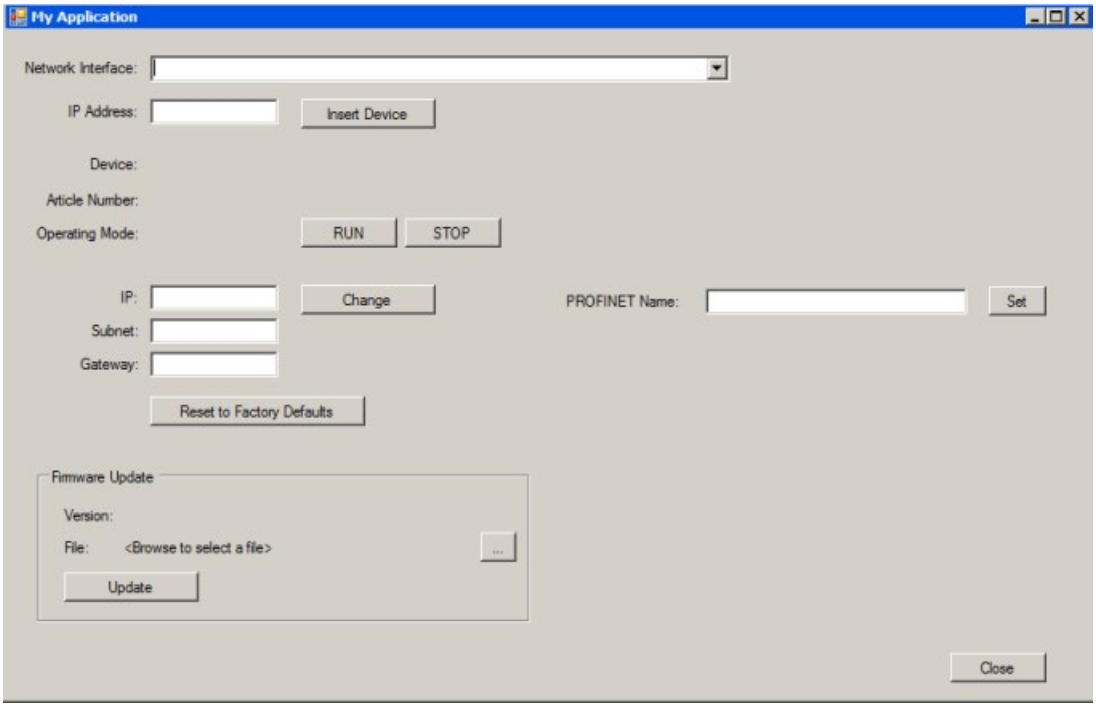
- 该操作将删除 CPU 中的用户程序和数据。例如，您可使用 STEP 7 显示用户程序。
- CPU 的 IP 地址保持不变。

您可以按照意愿执行其它测试。根据需要优化您的程序，以提供强大的错误处理功能。如果您对示例 1 的结果感到满意，可转到示例 5：更新 CPU 的固件 (页 35)。

2.6 示例 5：更新 CPU 的固件

示例 5 从示例 4 (页 32) 继续，将添加执行固件更新的功能。要完成此任务，您的应用程序需要包括以下部分：

- 用于显示当前固件版本的标签
- 用于显示选定固件更新文件的标签
- 用于选择固件更新文件的浏览按钮
- 用于执行固件更新的按钮



说明

此示例中的操作需要满足以下条件：

- CPU 没有密码保护
- CPU 中具有用户程序。（未复位为出厂设置。）

添加显示当前固件版本的字段

要添加用于显示所连接 CPU 的固件版本的字段，请按照下列步骤操作：

1. 添加一个名为“FirmwareVersion”的标签以显示当前固件版本。
2. 在标签前添加标识文本“版本：”(Version:)。
3. 将以下代码添加到现有 ClearCPUStatus (页 16) 辅助方法：

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

4. 将以下代码添加到现有 UpdateCPUStatus (页 18) 辅助方法：

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

添加显示选定文件的字段

您的应用程序需要一种选择固件更新文件的方法，以及一个显示当前选定文件的字段。最初，固件更新文件显示为空。

要添加文件名字段和文件浏览器控件，请按照下列步骤操作：

1. 添加一个名为“UpdateFileName”的标签以显示选定的文件名。
2. 在标签前添加标识文本“文件：”(File:)。
3. 将以下代码添加到现有 ClearCPUStatus (页 16) 辅助方法：

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

添加打开文件浏览器的按钮

您将提供一个用于打开文件浏览器的控件，供用户选择固件更新文件。在此示例中，文件浏览器最初为用户打开“我的文档”文件夹，并显示具有 .upd 扩展名的文件。如果用户浏览到其它文件夹，则文件浏览器会记住最后一个位置。

要添加按钮、浏览器控件和相应的程序逻辑，请按照下列步骤操作：

1. 在 `UpdateFileName` 标签旁边添加一个名为“`BrowseUpdateFile`”的按钮。
2. 创建一个全局字符串以包含从浏览器返回的路径和文件名：

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

3. 为 BrowseUpdateFile 按钮添加一个按钮单击方法并添加以下代码：

```
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;
   }
}
```

添加按钮以使用所选文件更新 CPU 固件

接下来，您必须添加一种方法，以便通过用户选择的固件更新连接的 CPU。

要添加用于执行固件更新的按钮，请按以下步骤操作：

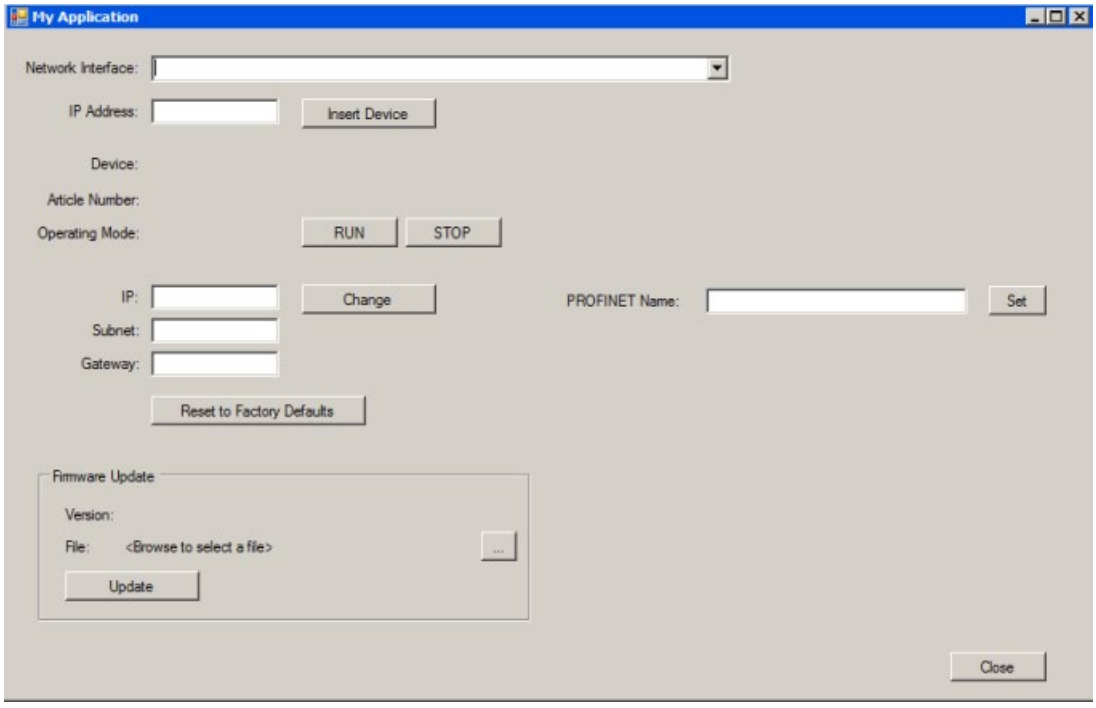
1. 添加名为“`FirmwareUpdate`”的按钮，并将“更新”(Update) 作为按钮文本。
2. 为按钮添加按钮单击事件并添加以下代码：

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

在实际应用程序中，添加检查功能以验证所选文件是否对连接的 CPU 有效。如果出现错误或选择了无效文件，请向用户提供反馈。

2.6.1 测试示例 5

编译并测试您的应用程序。如果您按照此示例中的步骤操作，则已创建如下所示的应用程序：



验证以下用户操作是否正确运行：

- 当您使用“插入设备”(Insert Device) 按钮连接到 CPU 时，“固件更新”(Firmware Update) 部分将显示正确的版本。
- 您可以使用“文件浏览”(File browse) 按钮浏览编程设备上的任何文件夹。
- 从浏览器中选择文件后，文件名会显示在“文件”(File) 字段中。
- 选择有效的固件更新文件后单击“更新”(Update)，CPU 会将固件更新为您选择的版本。

此外，测试各种错误情况，例如：

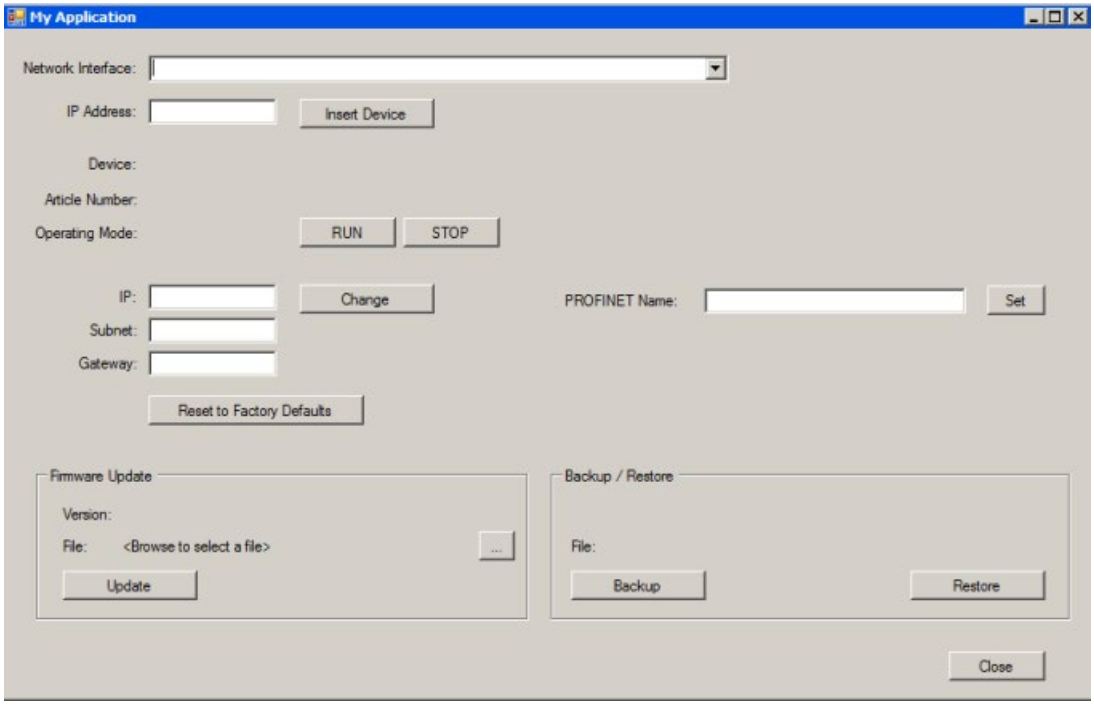
- 在未连接 CPU 的情况下单击“更新”(Update)，会显示错误消息。
- 选择无效的固件更新文件后单击“更新”(Update)，会显示错误消息。

您可以按照意愿执行其它测试。根据需要优化您的程序，以提供强大的错误处理功能。如果您对示例 5 的结果感到满意，可转到示例 6：备份 CPU 并通过备份文件恢复 CPU (页 40)。

2.7 示例 6：备份 CPU 并通过备份文件恢复 CPU

示例 6 从示例 5 (页 35) 继续，将添加备份当前 CPU 程序和恢复 CPU 程序的功能。要完成此任务，您的应用程序需要包括以下部分：

- 用于显示备份/恢复文件名称的标签
- 用于启动备份操作的按钮
- 用于启动恢复操作的按钮



说明

此示例中的操作需要以下条件：

- CPU 没有密码保护
- CPU 中具有用户程序。（未复位为出厂设置。）

添加备份/恢复文件标签

要为备份/恢复文件名添加标签字段，请按照下列步骤操作：

1. 向窗体添加一个标签并将其命名为 `BrFile`。
2. 向窗体添加一个全局字符串并将其命名为 `BrFileName`：

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

3. 在方法 `ClearCPUStatus` (页 16) 中，添加以下代码行：

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

编写按钮以将 CPU 备份到文件

要为此示例添加备份按钮并编写逻辑，请按照下列步骤操作：

1. 插入一个按钮控件并将其命名为“`Backup`”。
2. 为了能够执行文件操作，请将以下 `using` 语句添加到现有 using 语句中：

```
using System.IO;
```

3. 为 `Backup` 按钮添加单击事件并填充以下代码：

```
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;
```

编写按钮以从备份文件恢复 CPU

要为此示例添加恢复按钮并编写逻辑，请按照下列步骤操作：

1. 插入一个按钮控件并将其命名为“Restore”。
2. 为 Restore 按钮添加单击事件并填充以下代码：

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

if (MessageBox.Show("恢复是一项安全相关的操作，您是否希望继续？",
        "Backup / Restore Defaults", MessageBoxButtons.YesNo) == DialogResult.Yes)
{
    CurrentCPU.SelectedConfirmed = true;
}
else
{
    return;
}

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

对于这两个按钮操作，请检查操作结果并按照您的意愿进行处理。

说明

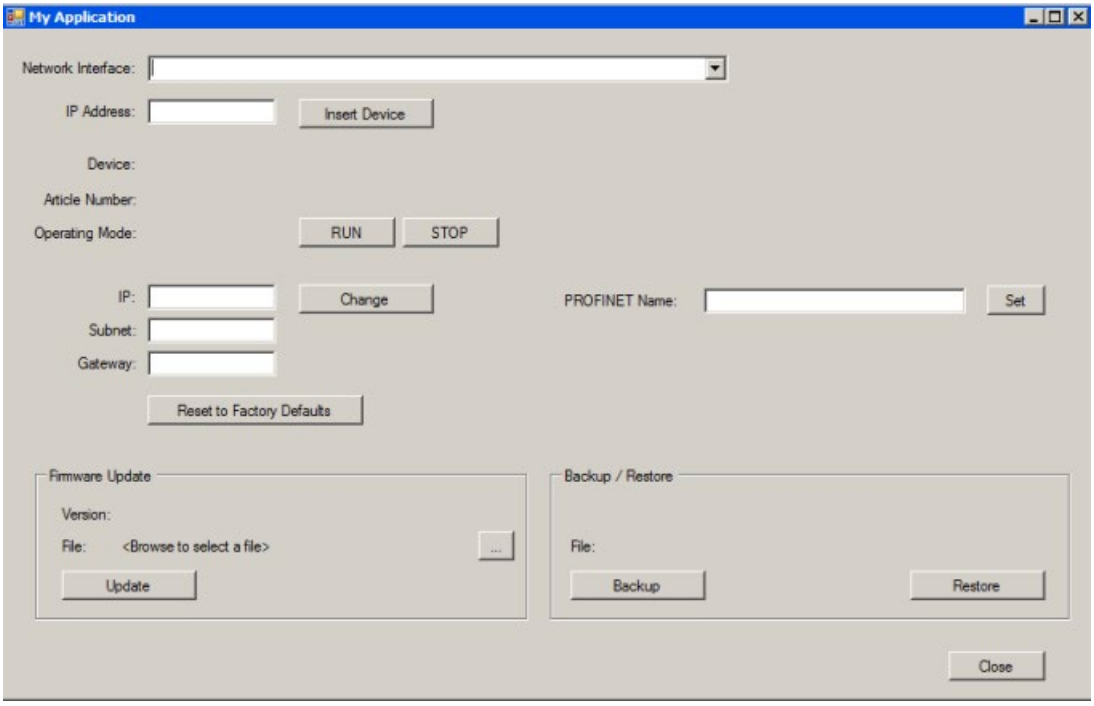
将程序恢复到故障安全 CPU 是一项安全相关的操作。请参见安全相关操作的用户界面编程指南 (页 48)中的指南。

可能的扩展名

此示例备份了 CPU 程序和组态，并将其恢复到同一 CPU。您可以添加其它控件并扩展此示例中的逻辑，从而备份一个 CPU 并将其恢复到另一 CPU。了解 API (页 46) 的功能以后，即可在网络上的多种设备上执行各种设备操作。

2.7.1 测试示例 6

编译并测试您的应用程序。如果您按照此教程中的步骤操作，则已创建如下所示的应用程序：



验证以下用户操作是否正确运行：

- 如果连接到具有用户程序且未受密码保护的 CPU，则单击“备份”(Backup) 按钮后，会在“文件”(File) 字段中显示备份文件名。
- 如果单击“复位为出厂设置”(Reset to Factory Defaults)，然后单击“恢复”(Restore)，则应用程序会将备份文件重新加载到复位 CPU 中。
- 如果使用“插入设备”(Insert Device) 按钮连接到其它 CPU，则备份/恢复文件为空。

此外，测试各种错误情况，例如：

- 如果尝试备份没有程序的 CPU，不会生成备份文件
- 如果尝试备份受密码保护的 CPU，不会生成备份文件
- 向 CPU 添加密码保护后，尝试将备份文件恢复到 CPU 会完全失败。

您可以按照意愿执行其它测试。根据需要优化您的程序，以提供强大的错误处理功能。

与已安装的西门子示例程序比较

运行 SAT SDK File Extractor (页 15) 时，已解压缩了 Examples 文件夹。完成本教程中的任务后，您可以将 `Form1.cs` 文件与解压缩的 Examples 文件夹中的 `Form1.cs` 文件进行比较。您可以编译并运行西门子示例并将其与您的应用程序进行比较。

2.8 将应用程序分发给您的用户

应用程序的最终用户需要以下内容：

- SAT SDK File Extractor (页 15) 创建的 Bin 文件夹
- SAT SDK File Extractor 创建的 Prerequisite 文件夹，或至少 Prerequisite 文件夹中的 “SIMATIC Automation Tool SDK PreReq Installer.exe”文件
- 自定义应用程序 .exe 文件以及属于您的应用程序的所有其它自定义文件。

将自定义文件与解压缩的 API 文件一起放在 Bin 文件夹中。

第三方软件许可条件与版权

SIMATIC Automation Tool SDK 安装列有第三方软件信息的文档，此类信息包括许可条件和版权以及开源软件信息。安装程序会在安装文件夹的“文件”(Documents) 文件夹中为每种语言安装此文件，具体如下：

语言文件夹	文件名
简体中文	SAT_SDK.ReadMe.OSS.V3.1.SP4.zh-CHS.rtf
德语	SAT_SDK.ReadMe.OSS.V3.1.SP4.de-DE.rtf
英语	SAT_SDK.ReadMe.OSS.V3.1.SP4.en-US.rtf
西班牙语	SAT_SDK.ReadMe.OSS.V3.1.SP4.es-ES.rtf
法语	SAT_SDK.ReadMe.OSS.V3.1.SP4.fr-FR.rtf
意大利语	SAT_SDK.ReadMe.OSS.V3.1.SP4.it-IT.rtf

西门子建议您将这些文档的内容以及任何其它第三方软件信息包含在您提供给用户的文件中。

为您的用户创建安装程序

使用选择的工具，为您的用户创建一个安装必要文件和文件夹的安装程序。实现调用以下命令的安装程序：

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

您可以在 Prerequisite 文件夹中的“Install Prerequisites Example.bat”文件中找到此命令行的示例。

用于 .NET 框架的 SIMATIC Automation Tool API 3

3.1 API 简介

SIMATIC Automation Tool API 允许用户使用各种 .NET 接口、类和方法创建自定义应用程序。通过 API 可与网络上的设备进行通信。用户可开发自定义应用程序来组合操作，支持特定于工业自动化网络的用户工作流程。

3.2 API 软件和版本兼容性

您必须购买 SIMATIC Automation Tool SDK 并同意许可条款和条件，这样才能使用 SDK API（应用程序编程接口）。该安装程序将提供您要接受的许可条款和条件。如果已购买 SDK，则不需要再次购买并许可 SIMATIC Automation Tool 使用该 API。有关购买和安装的说明，请参见 SIMATIC Automation Tool SDK 安装说明。

执行以下任务后，您可以使用 SDK API 编写自定义应用程序：

- 购买 SIMATIC Automation Tool SDK
- 安装 SIMATIC Automation Tool SDK
- 安装 SDK 时同意所有许可条款

兼容之前的版本

如果 SIMATIC Automation Tool V3.1 或 V3.1 SP1 API 的 .cs (C#) 文件中具有 .NET 框架代码，您可以在开发应用程序时使用此代码。您需要更新 Visual Studio 对 API 的引用 (页 60)。

您可能需要更改一些方法调用以使用某些方法的更新版本。编译项目时，会向您显示所需的更改。请参见本用户指南中的方法标题定义，进行必要的更改。

SDK API 与早于 V3.1 的 API 版本不兼容。您必须重写使用 V3.1 之前的 API 版本编写的程序。

3.3 为故障安全设备和安全相关操作设计用户界面应用程序

3.3.1 安全相关操作和故障安全设备的 API 支持

SIMATIC Automation Tool API 支持以下安全相关操作：

- 程序更新
- 从备份文件恢复设备
- 复位为出厂默认值
- 格式化存储卡

说明

*《SIMATIC Safety - 组态和编程》*手册包含一个标识为“S078”的警告。此警告声明如下：“若使用自动化或操作工具（TIA Portal 或 Web 服务器）绕过 F-CPU 的访问保护（如保存或自动输入“包括故障安全的完全访问（无保护）”保护级别的 CPU 密码或 Web 服务器密码），与安全相关的项目数据可能无法再防止受到无意更改的影响。”

此 S078 警告不适用于 SIMATIC Automation Tool API。API 支持与 F-CPU 通信并存储 F-CPU 的 CPU 密码。

API 提供的安全功能

SIMATIC Automation Tool 和 SIMATIC Automation Tool SDK API 已经过 TÜV SÜD 认证。

SIMATIC Automation Tool API 使用多种冗余技术。因此，API 能够保护用户应用程序代码，避免执行潜在危险操作。API 可以提供以下功能：

- 各安全相关操作的独立连接和合法化过程
- 故障安全设备与安全相关的操作的身份检查
- 安全程序的识别
- 要求使用安全 F-CPU 密码对任何受密码保护的 F-CPU 执行安全相关操作
- 使用 32 位 CRC 校验和来比较故障安全设备的在线和离线表示
- 使用汉明码 (页 57)表示 TRUE 和 FALSE 状态
- 程序更新与从备份操作恢复后进行 F 签名比较，以验证操作是否成功完成

3.3.2 安全相关操作的用户界面编程指南

⚠警告

尽可能保护与安全相关的操作

故障安全 CPU 与故障安全 I/O 和安全程序可共同提供操作的高度安全性。

使用 SIMATIC Automation Tool API 时，尽可能确保安全相关操作的安全。西门子对使用 SIMATIC Automation Tool API 开发的用户界面应用程序不承担任何责任。软件开发人员承担所有责任。

如果软件开发人员不遵循适当的编程习惯，则用户操作其开发的用户界面应用程序时，可能会导致死亡或人身伤害。

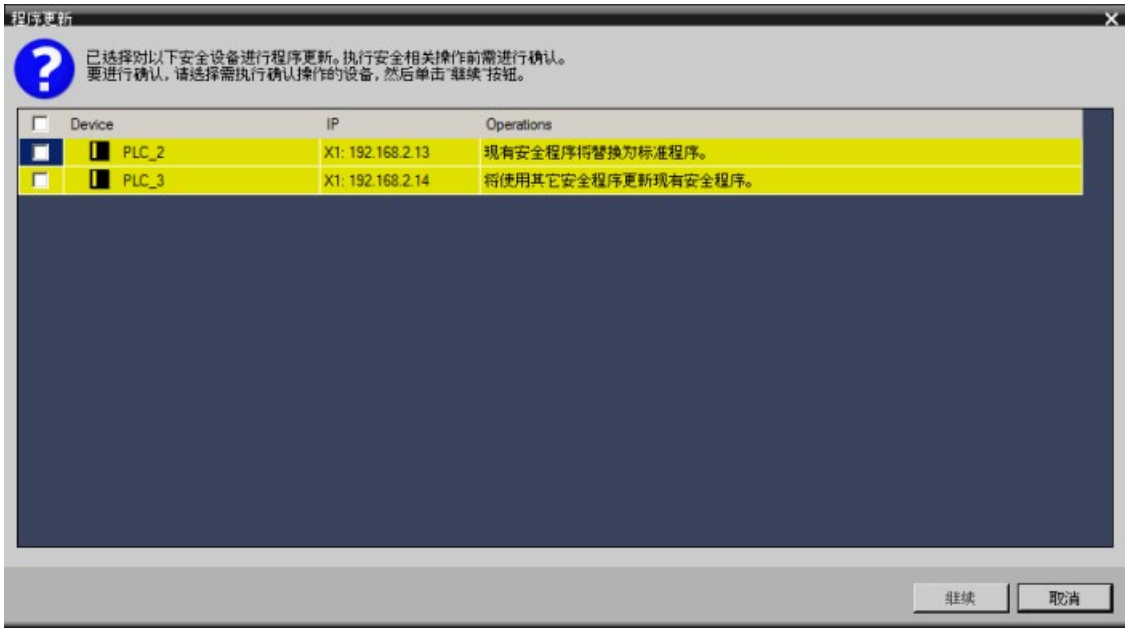
识别和保护与安全相关的操作

西门子建议提供确认安全相关操作的对话框。设计应用程序时，应要求用户选择每台故障安全设备进行操作，然后在执行操作前单击“继续”(Continue)。

使用 API 开发用户界面应用程序后，请确定操作是否属于以下安全相关操作之一：

- 程序更新
- 从备份文件恢复设备
- 复位为出厂默认值
- 格式化存储卡

对于安全相关操作，向用户提供确认对话框。使用 DetermineConfirmationMessage API 方法 (页 166)确定要显示的确认对话框类型。提供额外的确认对话框可防止用户不小心执行意外的安全相关操作。以下对话框为程序更新操作的确认对话框示例：



推荐编程习惯

使用以下编程习惯可确保保护安全相关操作并最大限度地减小不安全用户操作的可能性：

- 在单个线程上执行所有与安全相关的操作。
- 需要输入安全 F-CPU 密码才能进行安全相关操作。根据 CPU 密码验证输入的密码。用户输入密码时，使用星号隐藏画面上的密码。
- 检查所有方法的返回代码。确保程序逻辑仅在成功返回方法时继续。
- 在实现中包含适当的例外处理。API 会提供其检测到的关键内部错误等异常。请确保软件以适当的方式处理所有异常。
- 对于所有安全相关操作，请评估操作是否成功执行。程序更新成功后向用户显示一条消息。程序更新失败时显示一条错误消息。
- 在应用程序中使用汉明码 (页 57)实现布尔状态。
- 在应用程序中使用颜色 (页 51)指示故障安全设备、安全程序、安全 F-CPU 密码和其它用户输入的数据。
- 提示确认所有操作模式更改 (RUN/STOP)。
- 每次操作后刷新用户界面，以便应用程序显示正确的设备数据。

程序更新要求

对于所选 F-CPU 的程序更新，请为用户提供一个附加对话框，以重新选择故障安全设备并确认以下操作：

- 使用其它安全程序更新现有安全程序
- 使用标准程序更新现有安全程序
- 使用安全程序更新现有标准程序
- 使用安全程序更新无程序的 CPU
- 删除现有安全程序

更新安全程序后，验证 CPU 中更新的程序的 F 签名是否与程序更新文件的 F 签名匹配。

从备份恢复要求

在恢复备份文件之前，请根据程序更新的相同要求评估该文件是否为安全程序并提示用户确认。

认证

说明

获取用户界面应用程序的认证

西门子强烈建议通过 TÜV SÜD 等认证机构来证明设计和实施的安全性。

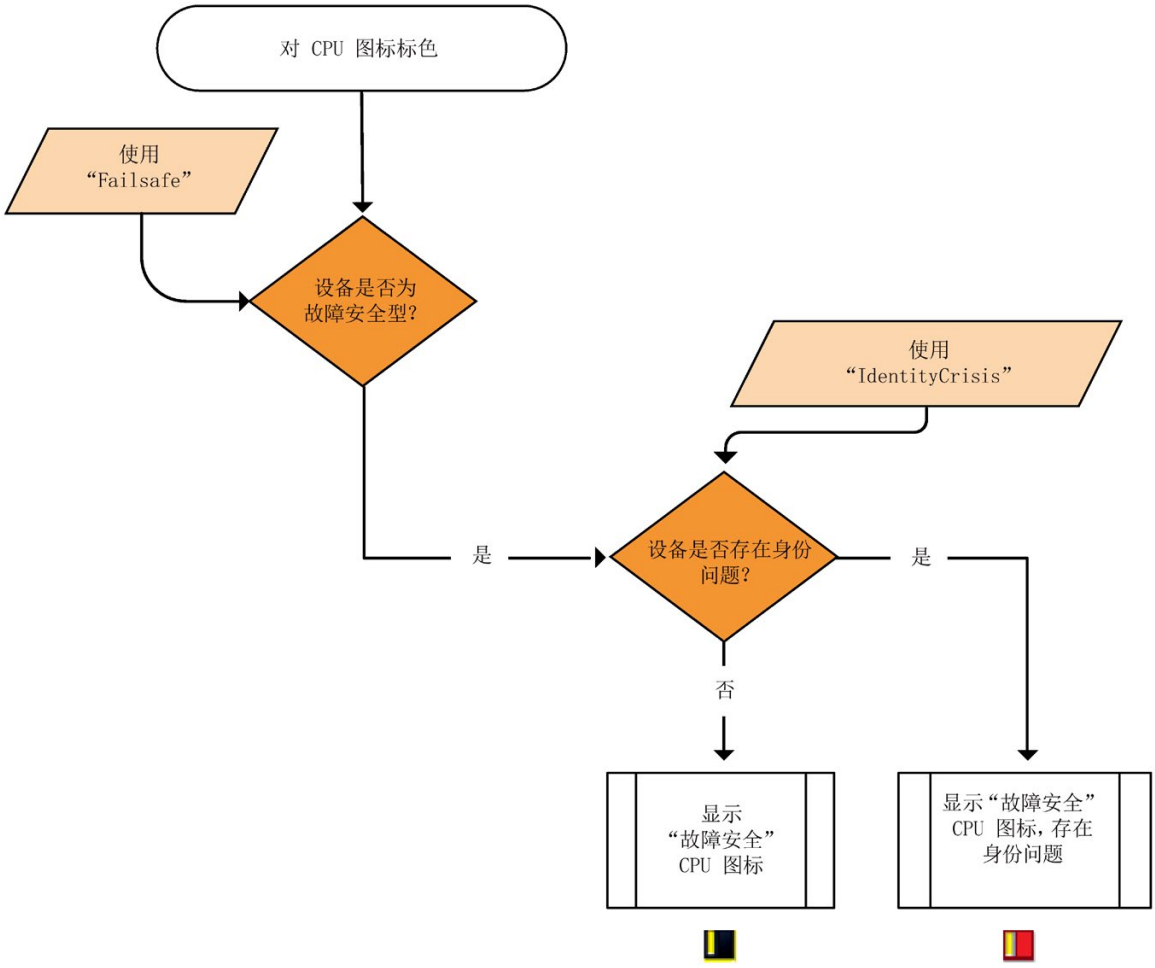
3.3.3 用户界面中的颜色编码安全域

如果使用 API 开发用户界面，西门子强烈建议使用颜色编码为用户提供与故障安全 CPU 和安全程序相关的任何内容的可视指示。决策树表示西门子推荐的典型安全相关领域的颜色编码逻辑。在设计应用程序时，请考虑采用相同或类似的方法。

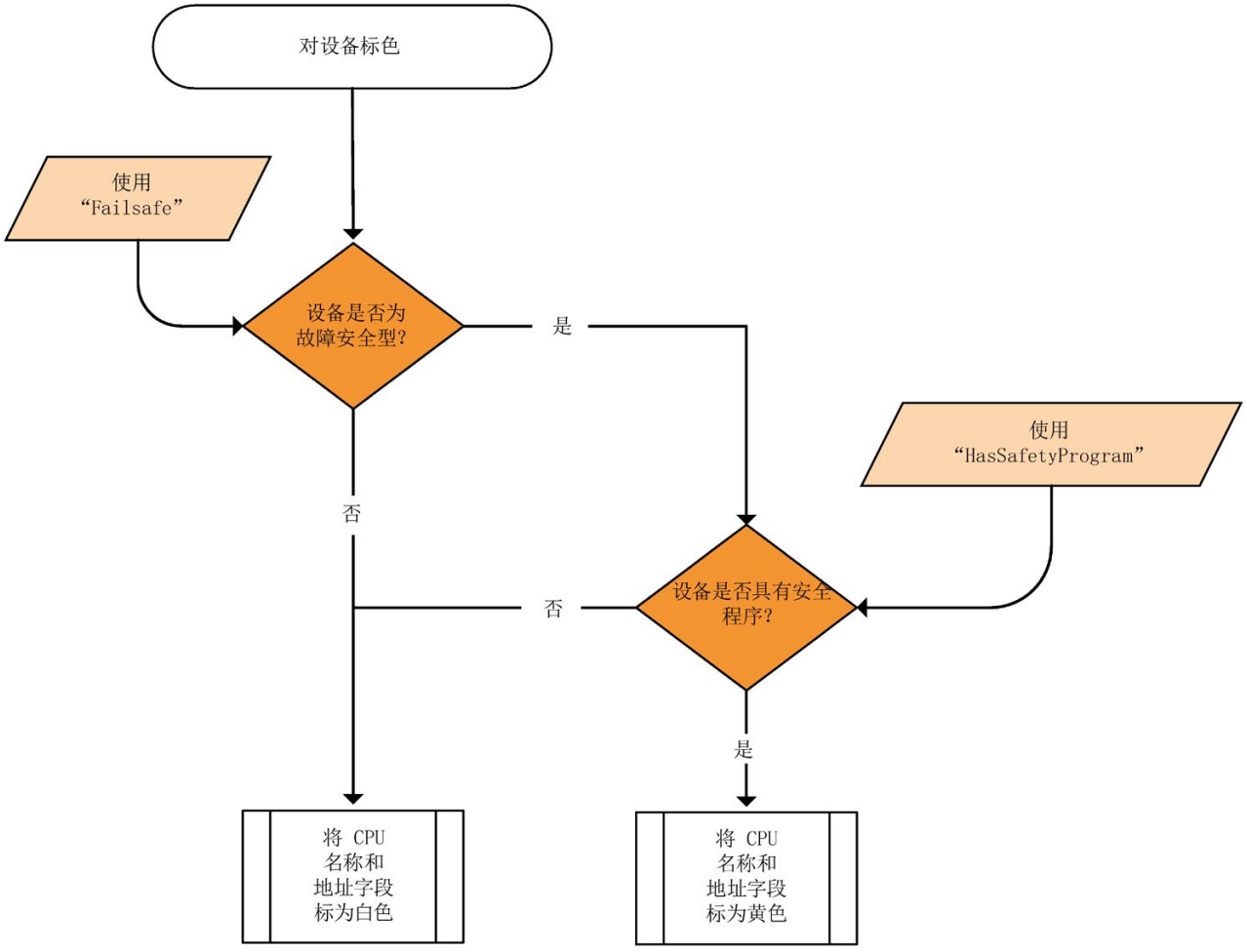
决策树约定

* 不会失败、未进行错误处理或表示应用程序故意忽略错误的情况的子进程。

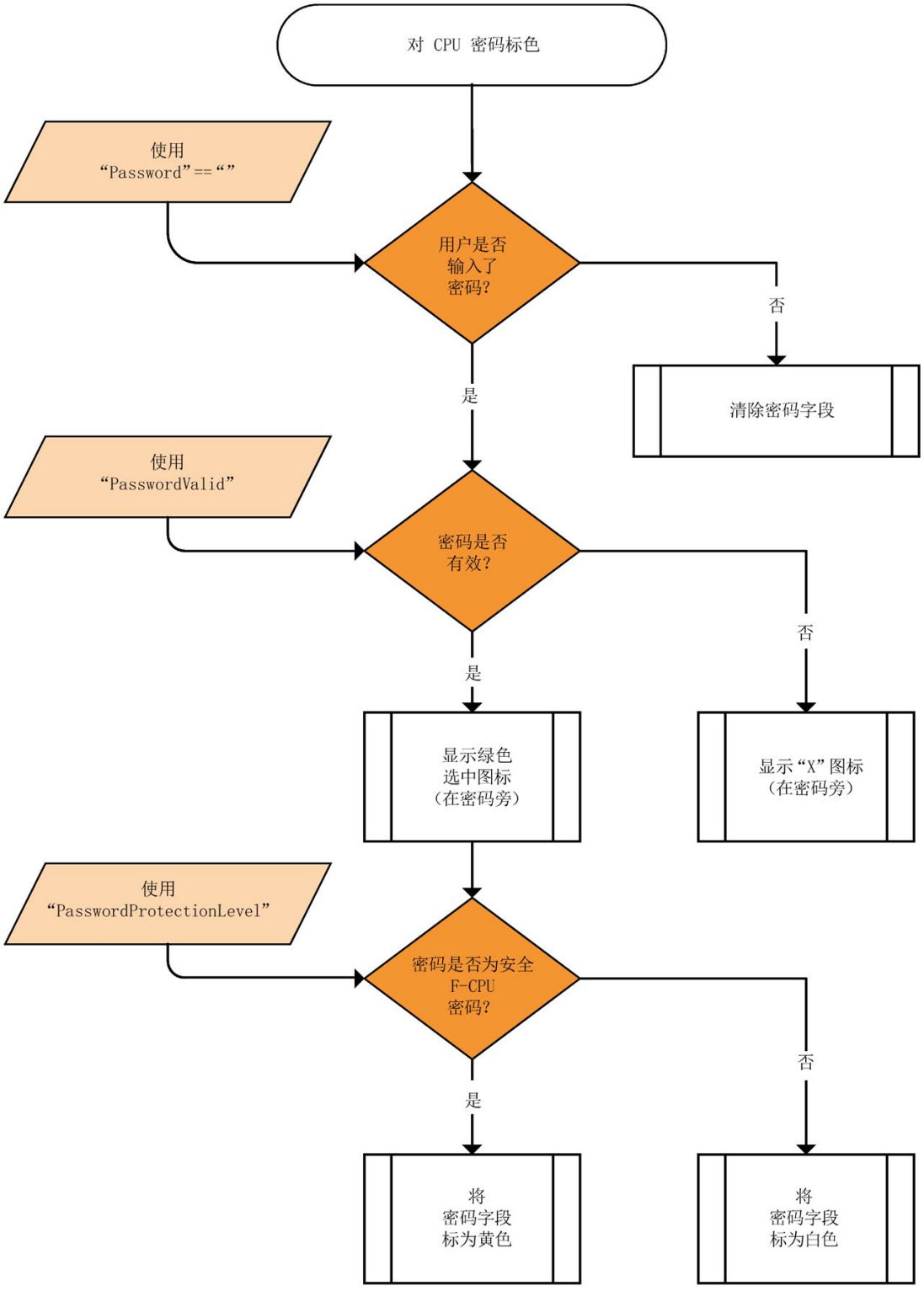
3.3.3.1 对 CPU 设备图标进行颜色编码



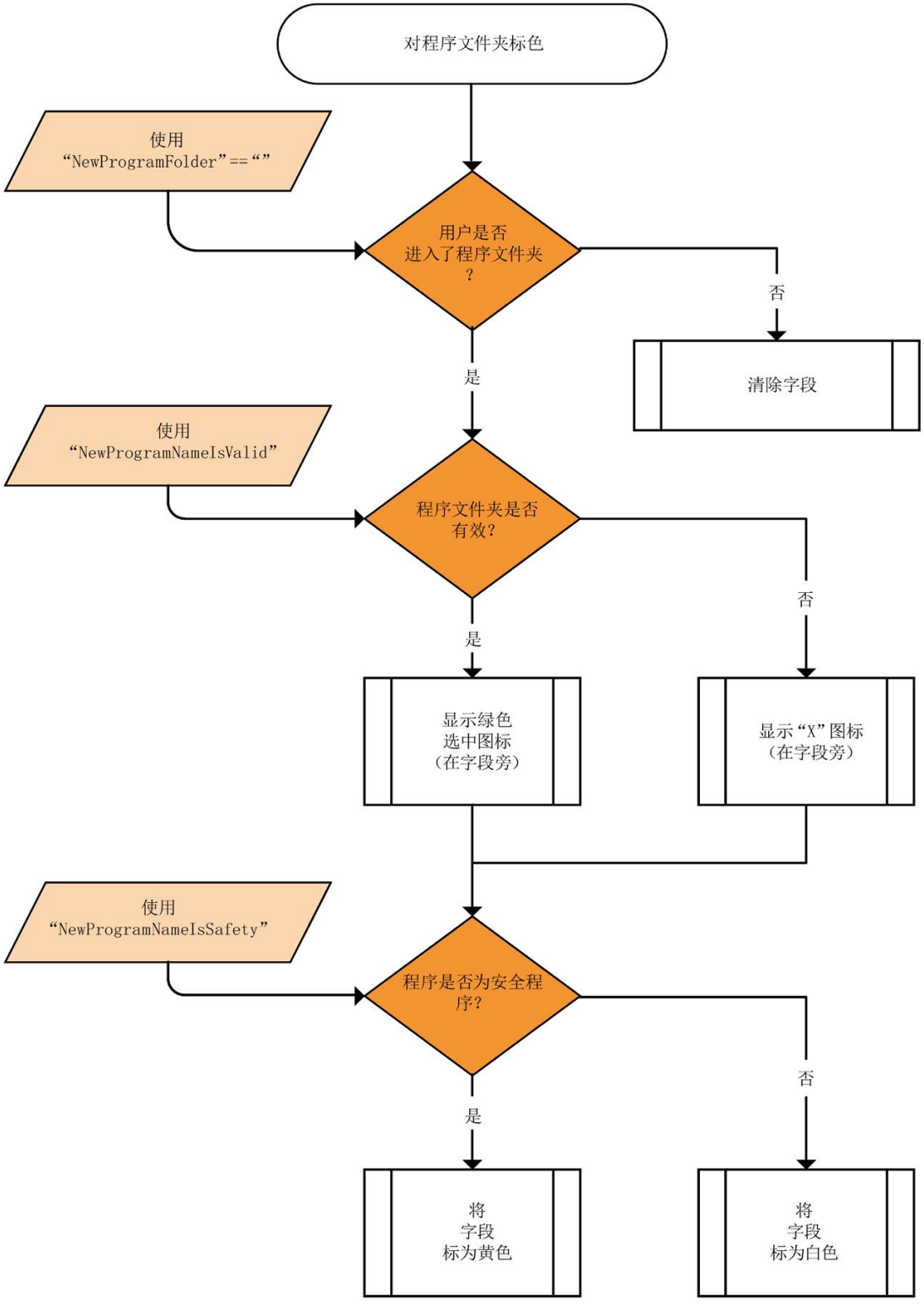
3.3.3.2 对设备数据进行颜色编码



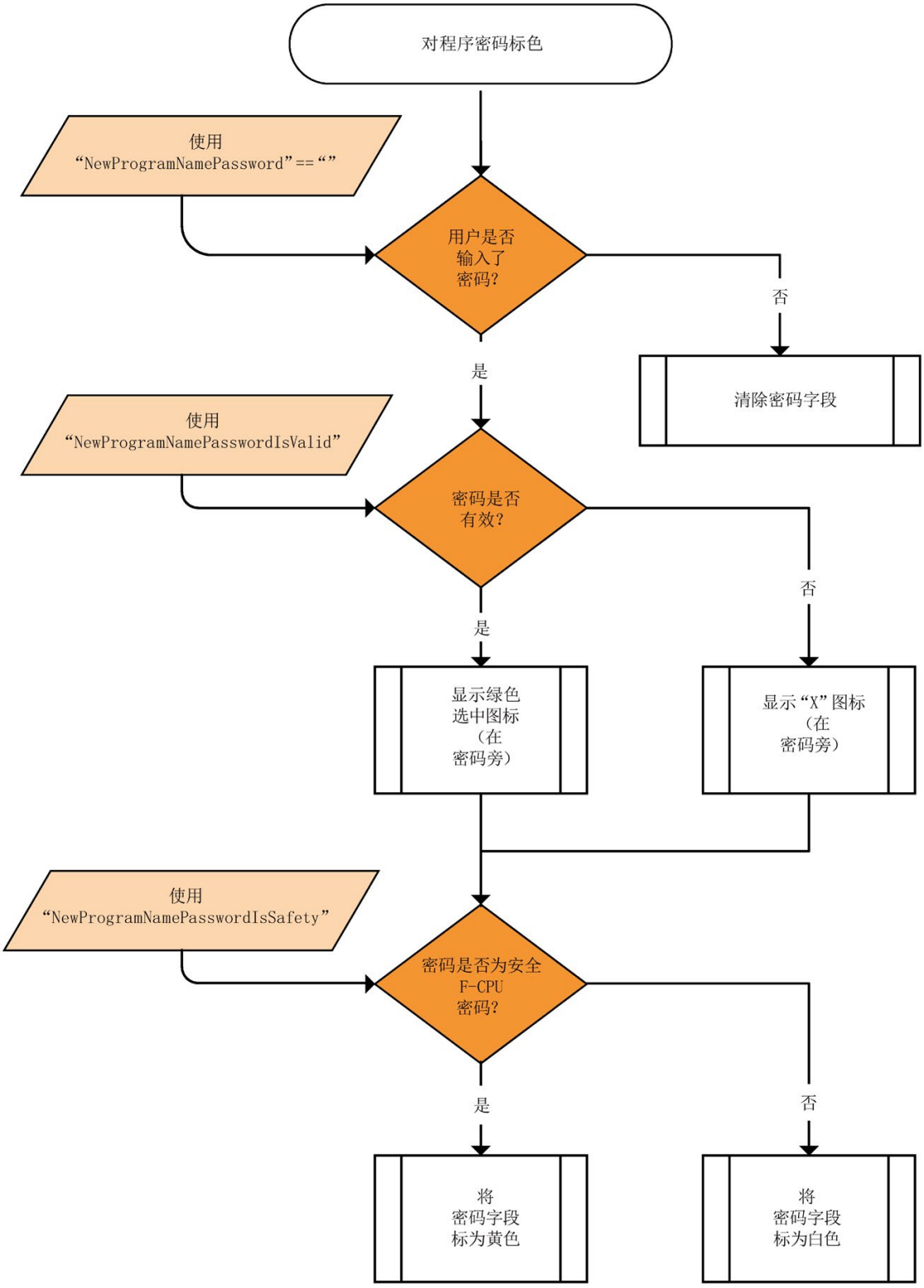
3.3.3.3 对 CPU 密码进行颜色编码



3.3.3.4 对程序文件夹进行颜色编码



3.3.3.5 对程序密码进行颜色编码



3.3.4 汉明码

汉明码为二进制码。可以检测偶然的位错误。SIMATIC Automation Tool API 使用 32 位汉明码，汉明距离为 8。API 使用汉明码来表示与安全相关操作有关的所有布尔值。可以对用户界面应用程序进行编程，以使用提供的布尔值状态进行安全相关操作。由于 API 使用汉明码实现这些状态，因此与安全相关的布尔状态的数据完整性具有较高的可信度。

3.4 架构概述

API 提供支持与 SIMATIC 设备网络通信的类别、接口和方法。

网络

.NET 类别 Network (页 75) 代表整个工业网络。此类别使用安装在编程设备上的网络接口卡 (NIC) 执行功能。网络类别用于扫描可用的接口卡并选择网络接口卡。

网络类别包括构造函数以及以下特性和方法：

- 网络构造函数 (页 75)
- QueryNetworkInterfaceCards 方法 (页 75)
- SetCurrentNetworkInterface 方法 (页 77)
- CurrentNetworkInterface 特性 (页 78)
- ScanNetworkDevices 方法 (页 78)
- SetCommunicationsTimeout 方法 (页 80)
- GetCommunicationsTimeout 方法 (页 81)
- GetEmptyCollection 方法 (页 82)

设备

网络上的各个设备由接口表示。每个接口类别都提供适用于所表示的网络设备的特性和方法。网络上的每个硬件设备最好由以下接口之一表示：

`IProfinetDevice` – 工业网络上可直接访问的任何设备都可以由此接口表示。

`ICPU` – 这代表直接连接到网络的 S7 CPU。CPU 支持特定功能。

`ICPUClassic` – 表示直接连接到网络的经典类型 S7-300 和 S7-400 CPU。

`IHMI` – 这代表直接连接到网络的 SIMATIC HMI。HMI 支持特定功能。

`IBaseDevice` – 此接口表示未直接连接到以太网，但可通过其它设备访问的设备。例如，将连接到网络上的 CPU 的 PROFIBUS 从站表示为 `IBaseDevice`。

`IModule` – 此接口用于表示插入 CPU、PROFINET 设备或 PROFIBUS 站的各个 I/O 模块。

`IHardware` – 这是所有其它接口的基本类别。此接口提供对网络上识别的所有硬件项通用的特性的访问权限。

接口会分组到表示设备组的集合中。提供集合以支持迭代、过滤和搜索。

`IProfinetDeviceCollection` – 网络上可直接访问的所有设备的集合。

`IModuleCollection` – 可代表插入 CPU 或 IM 的模块的集合。

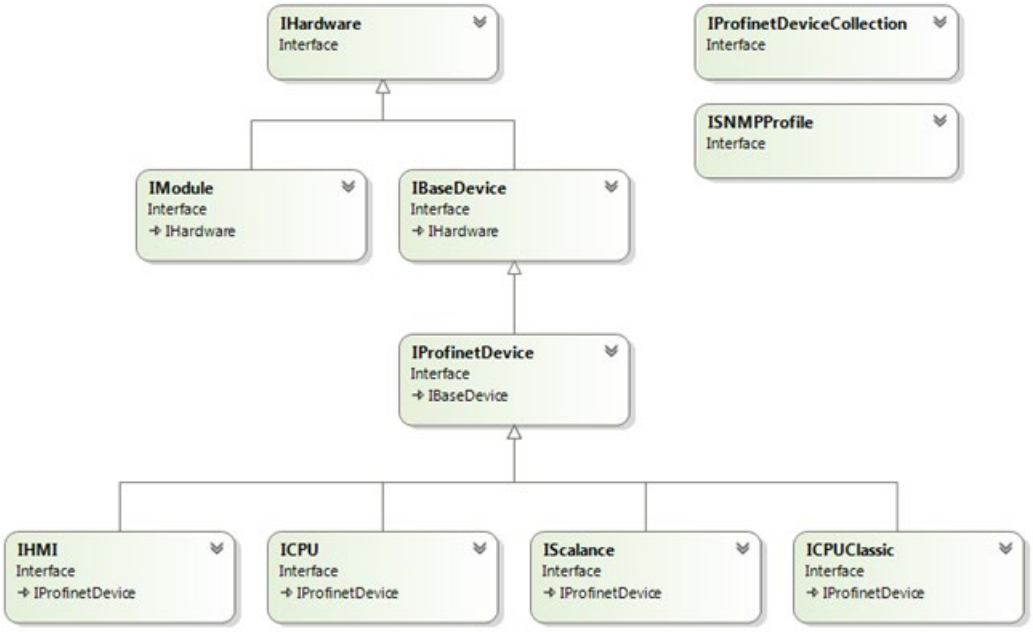
`IHardwareCollection` – 此集合代表 CPU 及其所有模块。

`IScanErrorCollection` – 此集合表示执行网络扫描操作后从所有设备返回的错误集。

设备类别、接口和方法：

- IProfinetDeviceCollection 类别 (页 85)
- IProfinetDevice 接口 (页 102)
- ICPU 接口 (页 123)
- IHMI 接口 (页 176)

以下类别图显示了这些接口类别之间的继承关系：



说明

请参见示例 (页 229)工业网络和用于表示每个网络组件的 SIMATIC Automation Tool API 类别。

3.5 在用户界面应用程序中引用 API

西门子为 API 提供了多个 DLL、可执行文件和源文件：

- `SIMATICAutomationToolAPI.dll`
- `DeviceManagerClient.dll` (HMI)
- `hmitr.dm.client.proxy.dll` (HMI)
- `hmitr.dm.client.stub.exe` (HMI)
- `hmitr.ipc.dll` (HMI)
- `AsModels` 文件夹和子文件夹（离线对象模型）

使用该 API 时，这些文件和 `AsModels` 文件夹必须位于开发自定义应用程序的同一文件夹中。

该 API 通过 Microsoft Visual Studio 2015 SP2 Update 3 使用 .NET Framework 4.6.2 创建。可以将使用该版本 Visual Studio 创建的应用程序与此 API 配合使用。本文件中的所有代码示例和屏幕截图均使用 Visual Studio 2015 SP2 Update 3 获取，采用 C# 编程语言。

要在应用程序中包含 API，必须在 Visual Studio 解决方案中添加 `SIMATICAutomationToolAPI.dll` 作为引用。

要能够将 PC 数据从应用程序中导出，应添加 `SimaticAutomationToolHealthCheck.dll` 作为引用。

在任何引用 API 类的应用程序中，必须添加以下引用 API 命名空间的语句：

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

如果应用程序导出 PC 数据，则必须添加以下语句：

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

要编译本文档中的任何代码示例，必须在与示例代码相同的源文件 (*.cs) 中使用正确的 `using` 语句。为简单起见，本文件中的各个代码示例未纳入使用语句。

3.6 S7 通信的要求

开发应用程序时的 S7 通信

要运行使用 API 开发的应用程序，必须在编程设备上安装 S7 通信组件。SIMATIC Automation Tool SDK 的安装程序将安装所需的 S7 通信组件。运行 SAT SDK File Extractor (页 15) 以将 API、HMI、S7 通信组件和其它所需组件保存到文件夹。将自定义应用程序的可执行文件放在同一文件夹中。从该文件夹运行应用程序并执行测试。

应用程序的最终用户的 S7 通信

您通常将为自定义应用程序的用户创建应用程序。例如，您可以使用 SIMATIC Automation Tool SDK 提供的 Windows Installer 创建此安装程序。当您的客户运行安装程序时，此安装程序会将 API、HMI、S7 通信组件、其它所需组件和您的自定义应用程序安装在一个文件夹中。然后，您的用户可以从该文件夹运行自定义应用程序。

3.7 公共支持类别

3.7.1 EncryptedString 类

很多 API 操作需要与受保护的 S7 CPU 进行合法连接。对于这些操作，需要密码作为方法的参数之一。S7 CPU 接受加密格式的密码。API 提供 `EncryptedString` 类来完成密码加密。

构造函数	说明
`EncryptedString()`	空加密字符串
`EncryptedString(string strText)`	加密字符串

属性名称	返回类型	说明
`IsEmpty`	`bool`	真（不存在密码时）
`IsEncrypted`	`bool`	真（存在加密密码时）

方法名称	返回类型	说明
`ToString()`	`string`	加密密码的十六进制字符串表示法
`Clear()`	`void`	清除加密密码
`GetHash()`	`byte[]`	密码的密码加密哈希数组表示法
`WriteToStream(Stream stream)`	`void`	序列化流密码
`ReadFromStream(Stream stream)`	`void`	反序列化流密码

此类提供一种加密纯文本密码的方法，从而可以使 CPU 连接合法化。许多代码示例体现了此类的典型用法。

要将密码加密以便在代码中多次使用，可将 `EncryptedString` 实例化。可将其作为参数传递给一次或多次调用，如下所示：

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

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

说明

如果 CPU 没有密码保护，则只需将空字符串传递给 `EncryptedString` 构造函数。例如，对于未组态保护的 CPU，可成功运行以下代码：

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

`EncryptedString` 对象不存储用户特定的纯文本密码。但是，如果应用程序将密码编码为文字串，则会造成安全风险。

例如，`new EncryptedString("myPassword")` 将纯文本“`myPassword`"编译到用户应用程序中。该密码可能对其它使用 .NET 反射的程序可见。

3.7.2 Result 类

`Result` 类对确定给定 API 操作是否成功执行的逻辑进行封装。大多数 API 操作都涉及网络通信。其中很多操作还涉及打开到网络设备的连接。这些操作能否成功并不确定。检查 API 操作返回的 `Result` 对象，以确定操作是否成功。

构造函数	说明
`Result()`	创建成功结果，无警告
`Result(ErrorCode nCode)`	创建特定错误，无警告
`Result(string strCode)`	设备定义的错误描述
`Result(Result result)`	创建包含结果参数内容的结果

属性名称	返回类型	说明
Warnings	`ErrorCode[]{get;}`	返回错误代码组中的所有警告
Error	`ErrorCode {get;}`	返回错误代码
HasWarnings	`Bool {get;}`	存在警告时为真
Failed	`Bool {get;}`	结果失败时为真
Succeeded	`Bool {get;}`	结果成功时为真 Succeeded 表示未失败

方法名称	返回类型	说明
`Clear()`	`void`	清除错误和所有警告
`SetError(ErrorCode error)`	`void`	设置当前错误
`AddWarning(ErrorCode warning)`	`void`	添加警告
`ChangeErrorToWarning()`	`void`	将当前错误更改为警告
`GetErrorDescription (Language language)`	`string`	获取以指定语言描述的当前错误字符串
`GetWarningDescription(Language language)`	`string[]`	获取以指定语言描述的警告字符串

在某些情况下，设备会返回一个错误字符串，具体描述该设备特定错误。在这种情况下，返回的 ErrorCode 是 DeviceDefinedError。方法 GetErrorDescription 提供设备特定的错误字符串。

在许多情况下，只需了解给定操作是否成功。在这种情况下，请检查 Succeeded 属性：

```
ICPU devAsCpu = dev as ICPU;
if (devAsCpu != null)
{
       //-----------------------------------------------------
       // 该设备为 CPU。
       // ICPU 接口可用于与之交互。
       //-----------------------------------------------------


    dev.Selected = true;
    Result retVal = dev.RefreshStatus();
    if (retVal.Succeeded)
    {
        //-------------------------------------
        // 继续操作....
        //-------------------------------------
    }
```

在其它情况下，获取有关故障的更多信息可能会有所帮助。要检查特定错误，请使用 ErrorCode 属性，如下所示：

```
    dev.Selected = true;
    Result retVal = dev.RefreshStatus();
    if (retVal.Succeeded)
    {
        //-------------------------------------
        // 继续操作....
        //-------------------------------------
    }
    else
    {
        //-------------------------------------
        // 发生了什么?
        //-------------------------------------
        switch (retVal.Error)
```

```
        {
            case ErrorCode.AccessDenied:
                break;
            case ErrorCode.TooManySessions:
                break;
        }
    }
```

有关可能错误值的列表，请参见 ErrorCode (页 216) 主题。

`Result` 类还提供特定于语言的错误描述。`GetErrorDescription` 方法使用 `Language` 值作为参数提供特定语言的错误描述。

例如，以下代码会返回德语的错误描述：

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

有关值列表，请参见语言 (页 223)枚举主题。

SIMATIC Automation Tool 具有警告功能，可用于了解已发生的问题。例如，程序更新结束时在设备上执行刷新可能会创建与主调用函数不直接相关的警告。可通过 `Result` 类访问这些警告，如下所示：

```
    if (retVal.HasWarnings)
    {
        foreach (ErrorCode warning in retVal.Warnings)
        {
            //--------------------------------------
            // 继续操作....
            //--------------------------------------
        }
    }
```

3.7.3 DiagnosticsItem 类

诊断项包含单个事件的诊断信息。可从 CPU 读取诊断缓冲区。有关详细信息，请参见 “ICPU 接口 (页 123)”一章。

构造函数	说明
`DiagnosticsItem()`	创建默认诊断项

属性名称	返回类型	说明
`TimeStamp`	`DateTime {get;}`	诊断事件的时间戳
`State`	`Byte {get;}`	0 = 离去事件；1 = 到达事件
`Description1`	`String {get;}`	基本描述
`Description2`	`String {get;}`	详细说明

3.7.4 DataChangedEventArgs 类

数据更改事件包含关于 API 内已更改的数据的信息。有关详细信息，请参见“IProfinet 接口 (页 102)”一章。

构造函数	说明
`DataChangedEventArgs(DataChangedType type)`	创建特定类型的事件
`DataChangedType type`	已更改的数据类型

属性名称	返回类型	说明
`Type`	`DataChangedType`	事件类型

与以下事件处理程序配合使用：

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

3.7.5 ProgressChangedEventArgs 类

进度更改事件包含关于 API 内已更改的数据的信息。有关详细信息，请参见“IProfinet 接口 (页 102)”一章。

构造函数	说明
`ProgressChangedEventArgs(ProgressAction action, int index, int count, uint hardwareID)`	用于创建和默认进度更改事件 args 类。
`ProgressAction action`	已发生的进度类型
`int index`	正在处理的当前项目的索引
`int count`	待处理的总项目
`uint hardwareID`	正在处理的项目的 ID

属性名称	返回类型	说明
`ID`	`uint {get;}`	项目 ID
`Cancel`	`bool {get;}`	设置为真以终止当前操作

属性名称	返回类型	说明
Count	int {get;}	最大值
Index	int {get;}	当前值
Action	ProgressAction {get;}	此事件的动作类型

与以下事件处理程序配合使用：

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

3.8 通用支持接口

3.8.1 IRemoteFile 接口

IRemoteFile 接口用于表示数据日志和配方中使用的文件。

属性名称	返回类型	说明
Selected	bool {get; set;}	已选状态
FileSize	ulong {get;}	CPU 上的文件大小
Name	string {get;set;}	设备上的文件名和扩展名

3.8.2 IRemoteFolder 接口

`IRemoteFolder` 表示数据日志和配方中使用的文件夹。

方法名称	返回类型	说明
`SetRemoteFile(string strFile)`	`Result`	设置远程文件名
`string strFile`		远程文件的完整文件名和路径

属性名称	返回类型	说明
`NewFileNameErrorCode`	`Result {get;}`	调用 `SetRemoteFile` 方法后保存的错误代码
`NewFileNameIsValid`	`bool {get;}`	文件名有效时为真
`FileUpdateAllowed`	`bool {get;}`	文件夹可以添加或替换列表中的文件时为真
`SelectedCount`	`int {get;}`	选择的文件数
`Files`	`List<IRemoteFile> {get;}`	此文件夹中的文件列表
`FolderType`	`RemoteFolderType {get;}`	文件夹类型（数据日志或配方）
`Exists`	`bool {get;}`	设备上存在此文件夹时为真
`Selected`	`bool {get;set;}`	已选中文件夹时为真
`NewFile`	`string {get;}`	要添加或替换的文件的完整文件路径
`NewFileName`	`string {get;}`	要添加或替换的文件的名称
`Name`	`string {get;}`	文件夹名称

3.8.3 IRemoteInterface 接口

IRemoteInterface 接口用于表示网络上的分布式 I/O。

属性名称	返回类型	说明
Devices	List<IBaseDevice>{get;}	用于表示分散式 I/O 的远程接口列表
InterfaceType	RemoteInterfaceType{get;}	远程接口的类型，如 PROFINET 或 PROFIBUS
Name	string {get;}	设备上的文件名和扩展名

3.8.4 IHardware 接口

IHardware 接口用于表示设备和模块的基本通用硬件接口。IModule 扩展 IHardware 接口。

方法名称	返回类型	说明
SetFirmwareFile(string strFile)	Result	设置固件文件以更新该设备或模块

属性名称	返回类型	说明
Comment	string {get;set;}	各设备和模块的注释
Selected	bool {get;set;}	用于所选状态的外部存储
NewFirmwareNameErrorCode	Result {get;}	SetFirmwareFile 方法的最后一个错误
FirmwareUpdateAllowed	bool {get;}	该设备或模块支持固件更新时为真
NewFirmwareNameIsValid	bool {get;}	固件文件对此设备或模块有效时为真
Failsafe	bool {get;}	设备或模块为故障安全型时为真
Supported	bool {get;}	支持此设备或模块时为真
NewFirmwareFile	string {get;}	固件文件的完整文件路径
NewFirmwareVersion	string {get;}	下拉列表中显示的固件文件的版本
Configured	bool {get;}	此设备或模块已组态时为真
HardwareNumber	short {get;}	设备或模块的硬件修订号
SlotName	string {get;}	设备或模块的插槽的名称
SubSlot	uint {get;}	设备或模块的子插槽编号

属性名称	返回类型	说明
Slot	uint {get;}	设备或模块的插槽编号
StationNumber	uint {get;}	设备或模块的站号
SerialNumber	string {get;}	设备或模块的序列号
FirmwareVersion	string {get;}	设备或模块的固件版本
ArticleNumber	string {get;}	设备或模块的订货号
Description	string {get;}	设备或模块的订货号说明
Name	string {get;}	设备或模块的名称
ID	uint {get;}	设备或模块的 ID

3.8.5 IBaseDevice 接口

IBaseDevice 接口用于扩展表示基础设备类型的 IHardware 接口。

方法名称	返回类型	说明
GetHardwareFromID(uint hardwareID)	IHardware	使用 ID 查找设备或模块

属性名称	返回类型	说明
HardwareInDisplayOrder	IHardwareCollection	按显示顺序排列的硬件项集合
HardwareInFirmwareOrder	IHardwareCollection	按固件更新顺序排列的硬件项集合
Modules	IModuleCollection	模块集合
ThreadNumber	int	当前线程的操作数
Family	DeviceFamily	系列类型枚举

事件	返回类型	说明
ProgressChanged	ProgressChangedEventHandler	被调用监视进度
DataChanged	DataChangedEventHandler	API 中的数据发生变化时进行调用

3.8.6 IHardwareCollection 接口

`IHardwareCollection` 接口用于表示 `IHardware` 接口列表。此接口扩展了 .NET List 类。

属性名称	类型	说明
-		

3.8.7 IModuleCollection 接口

`IModuleCollection` 接口用于表示 `IModule` 接口数组。此接口扩展了 .NET List 类。用于表示硬件机架中的本地模块和远程模块。

属性名称	类型	说明
-		

3.8.8 IScanErrorCollection 类

此接口类对通过 ScanNetworkDevices 调用确定设备网络扫描是否成功所需的逻辑进行封装。此外，此类可以通过返回 ScanErrorEvent，为网络上扫描未成功的设备提供错误信息。应始终检查 ScanNetworkDevices 调用返回的 IScanErrorCollection 对象是成功还是失败。增加此接口是为了使用户能够获取更多设备特定的扫描错误信息，从而为诊断网络扫描问题提供帮助。

构造函数	说明
`ScanErrorCollection()`	创建成功的网络扫描结果

属性名称	返回类型	说明
`Count`	`Int`	扫描错误事件的数量
`ScanErrorEvent[int]`	`ScanErrorEvent`	返回扫描事件集合中的指定扫描事件 如果是无效元素，则返回空值
`Failed`	`Bool {get;}`	结果失败时为真
`Succeeded`	`Bool {get;}`	结果成功时为真 `Succeeded = Failed`

以下情况会导致扫描状态为失败：

- 没有 SAT 许可证或未安装 SDK
- 网络接口无效
- 扫描时未发现设备
- 用户取消了操作。

方法名称	返回类型	说明
GetEnumerator()	IEnumerator<IScanErrorEvent>	获取扫描错误事件集合的枚举器

3.8.9 IScanErrorEvent 类

该类提供调用 ScanNetworkDevices 方法尝试扫描设备失败的相关信息。ScanErrorEvent 的内容提供设备的相关信息，并将错误指定为信息、警告、错误或无效。

构造函数	说明
ScanErrorEvent()	创建 ScanErrorEvent

属性名称	返回类型	说明
Code	ErrorCode {get;}	扫描错误
IP	Uint {get;}	设备的 IP 地址
MAC	ulong {get;}	设备的 MAC 地址
Name	string {get;}	设备的名称
TimeStamp	DateTime {get;}	错误的关联时间戳
Type	ScanErrorType {get};	发生错误的扫描的类型

3.9 网络类

3.9.1 网络构造函数

.NET 类 Network 使用安装在编程设备上的网络接口卡 (NIC) 执行功能。Network 类用于扫描可用的接口卡以及选择与工业网络通信的接口卡。

要与工业网络交互，程序会如下声明 Network 类型的变量：

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

可以使用此对象查找可用的网络接口，以及选择网络接口。

3.9.2 QueryNetworkInterfaceCards 方法

返回类型	方法名称
Result	QueryNetworkInterfaceCards

参数			
名称	数据类型	参数类型	说明
aInterfaces	List<string>	Out	编程设备上按名称列出的所有网络接口卡的集合

要识别可用的网络接口卡，可使用 `QueryNetworkInterfaceCards` 方法，如以下示例所示：

```
Network myNetwork = new Network();

List<String> interfaces = new List<String>();
Result retVal = myNetwork.QueryNetworkInterfaceCards(out interfaces);
if (retVal.Succeeded)
{
  //-----------------------------------------------------
  // 该方法返回字符串列表。
  // 列表中的每个字符串代表一个可用的 NIC。
  // 可以使用数组表示法迭代列表。
  //-----------------------------------------------------
  for (Int32 index = 0; index < interfaces.Count; index++)
  {
    String strInterfaceName = interfaces[index];
  }
}
```

如示例所示，该方法将输出字符串列表。列表中的每一项代表一个可用的网络接口卡，由名称标识。

`QueryNetworkInterfaceCards` 方法会返回包含操作状态的 `Result` 对象 (页 63)。Result 对象将指示操作是成功（`Succeeded` 属性为真）还是失败（`Succeeded` 属性为假）。导致操作失败的原因可能有多种。可以通过 Result 类包含的属性和方法获取失败的详细信息。

3.9.3 SetCurrentNetworkInterface 方法

返回类型	方法名称
Result	SetCurrentNetworkInterface

参数			
名称	数据类型	参数类型	说明
strInterface	string	In	要使用的网络接口的名称。通常，这是从 QueryNetworkInterfaceCards 方法返回的名称之一。

要使用其中一个已识别的网络接口卡访问工业网络，必须“设置”此接口。以下代码显示如何为 API 操作分配一个已标识的网络接口。在此示例中，代码选择 QueryNetworkInterfaceCards 示例 (页 75)中标识的第一个网络接口卡：

```
Network myNetwork = new Network();

List<String> interfaces = new List<String>();
Result retVal = myNetwork.QueryNetworkInterfaceCards(out interfaces);
if (retVal.Succeeded)
{
    retVal = myNetwork.SetCurrentNetworkInterface(interfaces[0]);
    if (retVal.Succeeded)
    {
            //------------------------------------------------------
            // 动作已成功执行。继续操作。
            //------------------------------------------------------
    }
}
```

3.9.4 CurrentNetworkInterface 属性

此只读属性查询当前选定的网络接口。以下示例说明了如何使用该属性：

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

说明

如果先前调用 SetCurrentNetworkInterface 方法时未选择任何网络接口，则此属性将返回空字符串。

3.9.5 ScanNetworkDevices 方法

返回类型	方法名称
IScanErrorCollection	ScanNetworkDevices

参数			
名称	数据类型	参数类型	说明
strFile	IProfinetDeviceCollection	Out	包含工业网络上每台可访问设备的 IProfinetDevice 元素的集合

确定网络接口后，可以查询工业网络上的设备。ScanNetworkDevices 方法输出项目集合，其中每一项代表直接连接到工业以太网的设备。这些设备可包括 CPU、本地模块、分散式 IO 站、HMI 和其它设备。

扫描包含多台设备的网络可能需要几分钟。返回的 IScanErrorCollection 中的“成功”(Succeeded) 属性会提示扫描成功或失败。

以下示例将创建所选网络接口上所有可访问设备的集合：

```
Network myNetwork = new Network();
//-----------------------------------------------------------------------
// 注：此示例假定对 SetCurrentNetworkInterface
```

```
// 的调用已完成网络 NIC 分配
//------------------------------------------------------------------------

IProfinetDeviceCollection scannedDevices;

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
    //-----------------------------------------------------
    // 操作成功。继续操作。
    //-----------------------------------------------------
}
else
{
    foreach (ScanErrorEvent scanErrorEvent in scanResult)
    {
       // 处理扫描错误事件
    }
}
```

此方法输出 `IProfinetDeviceCollection` (页 85)。

说明

`ScanNetworkDevices` 方法使用要求。

您必须拥有有效且未过期的 SAT 许可证或已安装 SDK 才能成功使用 ScanNetworkDevices 方法。如果在运行时尚未安装 SDK 或没有有效且未过期的 SAT 许可证，则 ScanNetworkDevices 方法返回空集合。ScanNetworkDevices 不向调用应用程序返回设备信息。

3.9.6 SetCommunicationsTimeout 方法

返回类型	方法名称
Result	SetCommunicationsTimeout

参数			
名称	数据类型	参数类型	说明
nTimeout	uint	In	操作超时的指定时间

可为使用 API 调用的操作设置时间限制。SetCommunicationsTimeout 允许指定 180 到 999 秒之间的时间限制（以秒为单位）。超出此范围的任何值都会导致操作失败。

以下示例说明了如何在 ScanNetworkDevices 操作中使用上述方法设置时间限制：

```
Network myNetwork = new Network();

IProfinetDeviceCollection scannedDevices;

Result retVal = Network.SetCommunicationsTimeout(180); //超时 3 分钟
IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
    //------------------------------------------------------
    // 动作已成功执行。继续操作。
    //------------------------------------------------------
}
```

3.9.7 GetCommunicationsTimeout 方法

返回类型	方法名称
uint	GetCommunicationsTimeout

设置通信超时值后，可以通过 `GetCommunicationsTimeout` 调用检索超时值。此方法返回当前超时值。

以下示例显示当前超时值大于 180 秒时，如何检索当前超时值以及将新的超时值设置为 180 秒：

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

3.9.8 GetEmptyCollection 方法

返回类型	方法名称
IProfinetDeviceCollection	GetEmptyCollection

GetEmptyCollection 方法也可以获取 Profinet 设备集合，而无需执行网络扫描。该方法十分便捷，例如，用户可以使用该方法在省去先执行完全网络扫描开销的情况下将设备插入到 IProfinetDeviceCollection 中。

IProfinetDeviceCollection 是一个无法实例化的接口。网络扫描返回集合。若要在不执行扫描的情况下打开项目，请调用 GetEmptyCollection 以返回空集合。然后，您可以将设备插入到空的 Profinet 设备集合中，而无需执行网络扫描。

以下示例说明了如何使用 GetEmptyCollection 方法：

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

Result result = collection.WriteToStream(stream);
if (result.Succeeded)
{
    //-----------------------------------------------------
    // 集合已成功序列化
    //-----------------------------------------------------
}
```

3.10 HealthCheck 类

3.10.1 HealthCheck 构造函数

.NET 类 `HealthCheck` 定义在 `SimaticAutomationToolHealthCheck.dll` 中，支持创建 PC 数据文件。

要与工业网络交互，程序会实例化 `HealthCheck` 类型的对象，如下所示：

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

3.10.2 ExportPCData 方法

返回类型	方法名称
`HealthCheckResultType`	`ExportPCData`

参数			
名称	**数据类型**	**参数类型**	**说明**
`filePath`	`string`	`In`	存储包含已导出 PC 数据的 zip 文件的完整文件路径

`ExportPCData` 方法创建一个 HealhCheck.zip 文件，该文件包含有关编程设备的数据。返回的 `HealthCheckResultType` 对象指示操作结果。HealthCheck.zip 文件中的 HealthCheck.log 文件包含关于已导出数据或已出现错误的详细信息。

```
HealthCheck myHealthCheck = new HealthCheck ();
String healthCheckFilePath = @"c:\export\healthcheck.zip";
HealthCheckResultType hcResult = myHealthCheck.ExportPCData(healthCheckFilePath);
if (hcResult == HealthCheckResultType.Success)
{
  //-------------------------------------------------
  // 动作已成功执行。继续执行操作。
  // HealthCheck.zip 文件中的 HealthCheck.log 文件
  // 包含关于已导出数据的详细信息。
  //-------------------------------------------------
}
```

```
else
{
  //-----------------------------------------------------
  // 动作执行失败，包含警告或由用户取消。
  //
  // 如果操作未取消，则 HealthCheck.zip 文件中的
  // HealthCheck.log 文件 包含关于
  //导出操作的信息。
  //-----------------------------------------------------
}
```

使用 ProgressChanged 事件监控 PC 数据的导出进程

API 提供 `ProgressChanged 事件` (页 119)以监控耗时可能过长的方法进程。`ExportPCData` 是一种耗时可能过长的方法。

要使用 `ProgressChanged` 事件来监控 `ExportPCData` 的进程，请将事件处理程序附加到该事件。然后，操作进程变化时会自动调用事件处理程序。

以下示例显示如何使用 `ProgressChanged` 事件监控 PC 数据的导出进程。示例代码定义了一个事件处理程序，并将其附加到 `ProgressChanged` 事件。然后，代码调用 `ExportPCData` 方法，这可能需要较长的时间。当 `ExportPCData` 完成时，示例代码将事件处理程序与事件分离。

```
HealthCheck myHealthCheck = new HealthCheck ();

// 登入进程事件
myHealthCheck.ProgressChanged += healthCheck_ProgressChanged;

HealthCheckResultType hcResult = myHealthCheck.ExportPCData(healthCheckFilePath);

// 删除进程事件登入
myHealthCheck.ProgressChanged -= healthCheck_ProgressChanged;

private static void healthCheck_ProgressChanged(object sender,ExportProgressEventArgs
e)
{
   String strProgress = String.Format("Processing {0} of {1}", e.WorkItem,
e.MaxEntries);
   // 设置为真，以取消终止进程
```

```
    e.Cancel = false;
}
```

3.11 IProfinetDeviceCollection 类

3.11.1 迭代集合中的项

3.11.1.1 迭代集合中的项

ScanNetworkDevices 方法输出 IProfinetDeviceCollection 类型的对象。此类别提供以多种方式迭代集合中的项目的功能。此外，还提供了根据特定条件过滤集合中的项目的方法。

扫描包含多台设备的网络可能需要几分钟。返回的 IScanErrorCollection 中的“成功”(Succeeded) 属性会提示扫描成功或失败。

以下示例将创建所选网络接口上所有可访问设备的集合：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
```

对于倾向使用类数组语法的编程人员，您可以按如下方式访问 scansDevices 中的项目：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);

if (scanResult.Succeeded)
{

    for (int deviceIdx = 0; deviceIdx < scannedDevices.Count; deviceIdx++)
    {
        //------------------------------------------------------------
        // 集合中的每个项目都是 IProfinetDevice。
        // 此接口将在下一部分中详细介绍
```

```
        //------------------------------------------------------
         IProfinetDevice dev = scannedDevices[deviceIdx];
    }
}
```

该集合还支持使用 foreach 语法进行迭代。以下示例所示为使用该语法迭代的相同集合：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
    foreach (IProfinetDevice dev in scannedDevices)
    {
        //---------------------------------------------
        // 变量"dev"现在代表集合中的下一个项目
        //---------------------------------------------
    }
}
```

3.11.1.2 GetEnumerator 方法

返回类型	方法名称
IEnumerator<IProfinetDevice>	GetEnumerator

参数			
名称	数据类型	参数类型	说明
-			

此方法用于枚举 IProfinetDeviceCollection 中的所有 IProfinetDevices。

3.11.1.3 Count 属性

返回类型	属性名称
int	Count

此属性会返回 IProfinetDeviceCollection 中的 IProfinetDevices 数目计数。

3.11.1.4 [ ] 特性

返回类型	属性名称
IProfinetDevice	this[int index]

此特性返回特定索引处的 IProfinetDevice。请参见以下示例：

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

Result result = collection.WriteToStream(stream);
if (retVal.Succeeded)
{
    //-----------------------------------------------------
    // 集合已成功序列化
    //-----------------------------------------------------

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

3.11.2 过滤集合中的项目

3.11.2.1 集合项

该集合包含工业以太网上各设备的一个项。并且可以包含来自多个产品系列的设备（例如 S7-1200、S7-1500 和 ET 200S）。

该集合还可以包含不同“类别”的设备（例如 CPU 或 IO 站）。对于不同类别的设备，可以使用特定操作。因此，在某些情况下，可能有助于过滤集合以使其仅包含某些设备的情况。

3.11.2.2 FilterByDeviceFamily 方法

返回类型	方法名称
`List<IProfinetDevice>`	`FilterByDeviceFamily`

参数			
名称	数据类型	参数类型	说明
`familiesToInclude`	`List<DeviceFamily>`	In	列表中要返回的设备系列类型

此方法返回仅包含指定产品系列的设备的集合。该过滤器首先构建为一个或多个设备系列的列表。例如，此声明仅针对 S7-1200 和 S7-1500 系列设备创建过滤器。

将此过滤器传递给 FilterByDeviceFamily 方法。结果是 IProfinetDeviceCollection 只包含指定产品系列的设备：

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

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

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

然后可以迭代生成的集合，从而只在包含的设备上执行操作。

说明

传递空的 List<DeviceFamily> 会导致返回空集合。

3.11.2.3 FilterOnlyCPUs 方法

返回类型	方法名称
List<ICPU>	FilterOnlyCPUs

SIMATIC Automation Tool API 支持多种只能在 CPU 上执行的操作。因此，过滤集合从而仅包含在网络上发现的 CPU 非常有用。

```
Network myNetwork = new Network();
IProfinetDeviceCollection scannedDevices;
IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);

if (scanResult.Succeeded)
{

   List<ICPU> cpus = scannedDevices.FilterOnlyCpus();
   foreach (ICPU cpu in cpus)
   {
```

```
        //------------------------------------------------------------
        // 循环访问仅包含 CPU 设备的列表
        //------------------------------------------------------------
    }
}
```

此方法返回 ICPU 列表。CPU 设备支持其它 API 操作。ICPU 接口提供这些操作。关于 ICPU 接口的详细描述，请参见“ICPU 接口 (页 123)”一章。

3.11.3 在集合中查找特定设备

3.11.3.1 FindDeviceByIP 方法

可在集合中搜索特定设备。

返回类型	方法名称
IProfinetDevice	FindDeviceByIP

参数			
名称	数据类型	参数类型	说明
ip	uint	In	要搜索的 IP 地址

以下示例所示为搜索指定 IP 地址的设备。如果在集合中找不到该设备，则返回 NULL 引用：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (!scanResult.Succeeded)
    return;

IProfinetDevice dev = scannedDevices.FindDeviceByIP(targetIPAddress);
if (dev != null)
```

```
{
    // 已找到
}
```

3.11.3.2 FindDeviceByMAC 方法

FindDeviceByMAC 方法可以搜索具有特定 MAC 地址的设备。

返回类型	方法名称
IProfinetDevice	FindDeviceByMAC

参数			
名称	数据类型	参数类型	说明
mac	ulong	In	要搜索的 MAC 地址

以下示例将搜索指定 MAC 地址的设备。如果在集合中找不到该设备，则返回 NULL 引用：

```
ulong targetMAC = 0x112233445566; // 相当于字符串 11:22:33:44:55:66
Network myNetwork = new Network();
IProfinetDeviceCollection scannedDevices;

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (!scanResult.Succeeded)
    return;

IProfinetDevice dev = scannedDevices.FindDeviceByMAC(targetMAC);
if (dev != null)
{
    // 已找到
}
```

3.11.4 序列化

3.11.4.1 将集合传入或传出外部数据文件

序列化方法可实现集合内容序列化以及将集合内容传入或传出外部数据文件。

3.11.4.2 WriteToStream 方法

返回类型	方法名称
Result	WriteToStream

参数			
名称	数据类型	参数类型	说明
stream	Stream	In	集合的序列化输出的目标

此方法用于在外部存储集合的内容。以下示例为此方法的使用方式：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (!scanResult.Succeeded)
    return;

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

Result retVal = scannedDevices.WriteToStream(f);

f.Close();
```

此方法在内部序列化版本信息，支持已保存数据的向前兼容性。

3.11.4.3 ReadFromStream 方法

ReadFromStream 方法会通过之前创建的序列化文件创建集合。以下示例说明了如何使用该方法：

返回类型	方法名称
Result	ReadFromStream

参数			
名称	数据类型	参数类型	说明
stream	Stream	In	反序列化集合的源

此方法用于通过之前创建的序列化文件创建集合。以下示例显示此方法的使用：

```
IProfinetDeviceCollection devices = Network.GetEmptyCollection();

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

Result retVal = devices.ReadFromStream(f);

f.Close();
```

3.11.4.4 ExportDeviceInformation 方法

ExportDeviceInformation 方法创建并导出一个包含当前设备集合数据的 .csv 文件：

返回类型	方法名称
Result	ExportDeviceInformation

参数			
名称	数据类型	参数类型	说明
ExportFilePath	string	In	生成的导出文件的目标文件路径

以下示例显示此方法的用法：

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

//------------------------------------------------------------------------
// 注：此示例假定网络 NIC 分配已经完成
// 如 SetCurrentNetworkInterface 中所示
//------------------------------------------------------------------------

//------------------------------------------------------------------------
// 获取网络上的当前设备内容
//------------------------------------------------------------------------

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (!scanResult.Succeeded)
  return;

//------------------------------------------------------------------------
// 为所有需要密码才能访问设备信息的 CPU 设置密码。
//------------------------------------------------------------------------
List<ICPU> aCPUs = scannedDevices.FilterOnlyCpus();
foreach (ICPU cpu in aCPUs)
{
   if (cpu.Protected == true)
   {
       cpu.SetPassword(new EncryptedString("Password"));
   }
}


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

3.11.4.5 ExportDeviceDiagnostics 方法

ExportDeviceDiagnostics 方法创建并导出一个包含当前设备集合中各 CPU 诊断数据的 .csv 文件：.csv 文件中的列标题为英语。

返回类型	方法名称
Result	ExportDeviceDiagnostics

参数			
名称	数据类型	参数类型	说明
ExportFilePath	string	In	生成的导出文件的目标文件路径
Language	Language	In	用于导出的诊断缓冲区条目的语言
Format	TimeFormat	In（可选）	诊断条目日期和时间的显示格式

以下示例显示此方法的用法：

```
Network myNetwork = new Network();
IProfinetDeviceCollection scannedDevices;
//-----------------------------------------------------------------------
// Note: this example assumes Network NIC assignment has already been made
// 参见 SetCurrentNetworkInterface
//-----------------------------------------------------------------------
IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (!scanResult.Succeeded)
  return;

//-------------------------------------------------------------------------------
// 选择设备列表中的所有 CPU 以包含诊断信息
//-------------------------------------------------------------------------------
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;
}
//-----------------------------------------------------------------------
// 为所有需要密码才能访问设备信息的 CPU 设置密码。
//-----------------------------------------------------------------------

List<ICPU> aCPUs = deviceCollection.FilterOnlyCpus();
foreach (ICPU cpu in aCPUs)
{
   cpu.SetPassword(new EncryptedString("Password"));
}
// 借助 API，客户端应用程序可确定文件路径和名称
String exportFilePath = @"c:\export\Device\DeviceDiagnostics.csv";
IScanErrorCollection exportResult =
   scannedDevices.ExportDeviceDiagnostics(exportFilePath,
Language.English,TimeFormat.Local);
```

对于集合中的每个 CPU，返回的错误集合指示是否已成功获取诊断数据。如果未提供所需的密码或发生网络错误，则该错误会在 CPU 中列出。错误导致 CPU 中将存储一个空的数据条目。

使用 ProgressChanged 事件监控设备诊断的导出进程

API 提供 ProgressChanged 事件以监控耗时可能过长的方法进程。ExportDeviceDiagnostics 是一种耗时可能过长的方法。

要使用 ProgressChanged 事件来监控 ExportDeviceDiagnostics 的进程，请将事件处理程序附加到该事件。然后，操作进程变化时会自动调用事件处理程序。

以下示例显示如何使用 ProgressChanged 事件监控设备诊断的导出进程。示例代码定义了一个事件处理程序，并将其附加到 ProgressChanged 事件。然后，代码调用 ExportDeviceDiagnostics 方法，这可能需要较长的时间。当 ExportDeviceDiagnostics 完成时，示例代码将事件处理程序与事件分离。

```
Network myNetwork = new Network();

IProfinetDeviceCollection deviceCollection = null;
```

```
// 调用序列以导出设备诊断
// 扫描网络设备
IScanErrorCollection scanErrors = myNetwork.ScanNetworkDevices(out deviceCollection);

// 登入进程事件
deviceCollection.ProgressChanged += Export_ProgressChanged;

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

// 删除进程事件登入

deviceCollection.ProgressChanged -= Export_ProgressChanged;

private static void Export_ProgressChanged(object sender,ExportProgressEventArgs e)
{
   String strProgress = String.Format("Processing {0} of {1}", e.WorkItem,
e.MaxEntries);
   // 设置为真，以取消终止进程
   e.Cancel = false;
}
```

3.11.5 手动将项目添加到集合中

根据工业网络的物理拓扑结构，网络上可能存在无法响应 DCP 命令的设备（例如 `ScanNetworkDevices` 方法使用的设备），但可以通过 IP 地址访问。用于插入设备的方法允许用户根据设备地址手动将设备添加到集合中。

3.11.5.1 InsertDeviceByIP 方法

返回类型	方法名称
`Result`	`InsertDeviceByIP`

参数			
名称	数据类型	参数类型	说明
index	int	In	集合中要插入值的位置
`ip`	`uint`	In	要添加到集合的设备的 IP 地址

使用以下代码扫描网络，然后在指定的索引处手动添加特定 IP 地址的设备：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (!scanResult.Succeeded)
    return;

UInt32 missingDeviceIPAddress = 0xC0A80001; // 192.168.0.1
Int32 index = 0;

Result retVal = scannedDevices.InsertDeviceByIP(index, missingDeviceIPAddress);
```

3.11.5.2 InsertDeviceByMAC 方法

返回类型	方法名称
Result	InsertDeviceByMAC

参数			
名称	数据类型	参数类型	说明
index	int	In	集合中要插入值的位置
mac	ulong	In	要添加到集合的设备的 MAC 地址

使用以下代码扫描网络，然后在指定的索引处手动添加特定 MAC 地址的设备：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (!scanResult.Succeeded)
    return;

UInt64 targetMAC = 0x112233445566; // 相当于字符串 11:22:33:44:55:66
Int32 index = 0;

retVal = scannedDevices.InsertDeviceByMAC(index, targetMAC);
```

3.11.6 从集合中复制数据

3.11.6.1 CopyUserData 方法

可能需要将用户输入的数据从一个 IProfinetDeviceCollection 复制到另一个中。API 不提示用户重新输入此信息，而是提供 CopyUserData 方法。

返回类型	方法名称
Result	CopyUserData

参数			
名称	数据类型	参数类型	说明
oldNetwork	IProfinetDeviceCollection	In	应用程序中之前使用的列表

以下代码将用户输入的数据从一网络扫描复制到另一扫描中。

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (!scanResult.Succeeded)
    return;

IProfinetDeviceCollection rescannedDevices;
IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out rescannedDevices);
if (!scanResult.Succeeded)
    return;

Result retVal = rescannedDevices.CopyUserData(scannedDevices);
if (!retVal.Succeeded)
    return;
```

3.11.7 从集合中移除设备

3.11.7.1 Clear 方法

返回类型	方法名称
void	Clear

参数			
名称	数据类型	参数类型	说明
-			

此方法用于清除扫描到的设备中的内容。

3.11.7.2 Remove 方法

返回类型	方法名称
void	Remove

参数			
名称	数据类型	参数类型	说明
device	IProfinetDevice	In	列表中待移除的设备。

此方法用于从集合中删除特定项。

3.12 IProfinetDevice 接口

3.12.1 IProfinetDevice 属性

IProfinetDeviceCollection 集合中的每一项都由 IProfinetDevice 接口表示。可通过此接口访问数据并对直接连接至工业网络的所有设备提供操作。

IProfinetDevice 接口支持以下特性，这些特性提供了有关网络设备的信息。这些属性都是只读的。为确保其能够返回当前信息，读取特性前应首先对设备调用 RefreshStatus 方法。

属性名称	返回类型	说明
ArticleNumber	string {get;}	模块的订单号。也称为 MLFB 或“订货号”。
Comment	string {get;set;}	用户可通过此属性指定设备注释，并在 SIMATIC Automation Tool 用户界面中使用。此注释与 API 操作无关。
Configured	bool {get;}	当设备具有有效组态时为真
DefaultGateway	uint {get;}	设备的默认网关地址，表示为一个无符号整数。编码的网关地址使用一个字节表示地址中的每个十进制值。例如，编码值 0xC0A80001 等同于更常见的字符串表示形式 192.168.0.1
DefaultGatewayString	string {get;}	设备的默认网关地址，表示为“xx.xx.xx.xx”形式的字符串。 例如：192.168.0.1

属性名称	返回类型	说明
Description	string {get;}	硬件项描述，基于订货号。此描述与用户可在 TIA Portal 中看到的描述相同。 示例：CPU 1215 DC/DC/DC
DeviceFound	bool {get;}	在网络扫描中是否发现了此设备？
DuplicateIP	bool {get;}	该设备的 IP 地址是否重复？
DuplicateProfinetName	bool {get;}	该设备的 PROFINET 名称是否重复？
Failsafe	bool {get;}	根据其 ArticleNumber，此设备是否为故障安全设备？
Family	DeviceFamily {get;}	此设备属于哪个产品系列？更多信息，请参见 DeviceFamily 枚举的描述。
FirmwareUpdateAllowed	bool {get;}	该设备是否支持固件更新？
FirmwareVersion	string {get;}	设备的当前固件版本
ID	uint {get;}	工作站中每个设备和模块的唯一标识符。该标识符用作执行 FirmwareUpdate 时的唯一标识符。
HardwareNumber	short {get;}	设备的硬件版本或“F-Stand”。（函数状态）
IP	uint {get;}	设备的 IP 地址，表示为一个无符号整数。编码的 IP 地址使用一个字节表示 IP 地址中的每个十进制值。例如，编码值 0xC0A80001 等同于更常见的字符串表示形式“192.168.0.1” 注：SIMATIC Automation Tool 仅支持 IPv4 地址。不支持 Ipv6 地址。

属性名称	返回类型	说明
IPString	string {get;}	设备的 IP 地址，表示为“xx.xx.xx.xx”形式的字符串（例如 192.168.0.1）
MAC	ulong {get;}	分配给设备的唯一 MAC 地址。编码的 MAC 地址使用一个字节对为该地址定义的 6 个八位字节进行编码。例如，编码的 MAC 地址 0x112233445566 等同于更常见的字符串表示形式 11:22:33:44:55:66。
MACString	string {get;}	分配给设备的唯一 MAC 地址，表示为 11:22:33:44:55:66 形式的字符串
Modules	IModuleCollection {get;}	插入工作站的模块集合。此处详细描述了该属性。
Name	string {get;}	设备名称
NewFirmwareFile	string {get;}	将用于固件更新的固件文件的位置
NewFirmwareNameErrorCode	Result {get;}	附加至新固件名称的 ErrorCode
NewFirmwareNameIsValid	bool {get;}	设置的固件文件是否有效?
NewFirmwareVersion	string {get;}	此属性用于 SIMATIC Automation Tool 用户界面中。此属性与 API 操作无关。
NewDefaultGateway	String {get;}	此属性用于 SIMATIC Automation Tool 用户界面中。此属性与 API 操作无关。

属性名称	返回类型	说明
NewIP	String {get;set;}	此属性用于 SIMATIC Automation Tool 用户界面中。此属性与 API 操作无关。
NewProfinetName	String{get;set;}	此属性用于 SIMATIC Automation Tool 用户界面中。此属性与 API 操作无关。
NewProgramName	String {get;set;}	此属性用于 SIMATIC Automation Tool 用户界面中。此属性与 API 操作无关。
NewRestoreName	String{get;set;}	此属性用于 SIMATIC Automation Tool 用户界面中。此属性与 API 操作无关。
ProfinetName	String {get;}	设备的 Profinet 名称
ResetToFactoryAllowed	bool {get;}	设备是否支持 ResetToFactory？
Selected	bool {get;set;}	将设备标记为选定设备，以便能够执行操作
SerialNumber	string {get;}	设备的唯一序列号
Slot	uint {get;}	硬件项的插槽号
SlotName	string {get;}	此属性用于 SIMATIC Automation Tool 用户界面中。此属性与 API 操作无关。
StationNumber	uint {get;}	设备的站编号
SubSlot	uint {get;}	设备的子插槽。这与 S7-1200 SB 模块等可插拔子模块相关。

属性名称	返回类型	说明
`Supported`	`bool {get;}`	如果数据库中存在订货号，并且当前 SIMATIC Automation Tool API 支持该设备，则为真。
`SubnetMask`	`uint {get;}`	设备的子网掩码，表示为一个无符号整数。编码的子网掩码使用一个字节表示地址中的每个十进制值。例如，编码值 0xFFFFFF00 等同于更常见的字符串表示形式 255.255.255.0。
`SubnetMaskString`	`string {get;}`	设备的子网掩码，表示为“xx.xx.xx.xx”形式的字符串（即 192.168.0.1）

参见

DeviceFamily (页 215)

模块属性和 IModuleCollection 类 (页 121)

3.12.2 IProfinetDevice 方法

3.12.2.1 RefreshStatus 方法

返回类型	方法名称
Result	RefreshStatus

当 ScanNetworkDevices 方法创建 IProfinetDeviceCollection 时，扫描只会返回关于各设备的少量信息。要获取设备的所有可用信息，应调用 RefreshStatus 方法。此方法可以建立与设备的连接，查询各种信息，然后断开与设备的连接。

以下代码会为网络中的每个设备调用 RefreshStatus：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded))
{
    foreach (IProfinetDevice dev in scannedDevices)
    {

        retVal = dev.RefreshStatus();
        if (retVal.Succeeded)
        {
                //-------------------------------------------------
                // 操作成功。数据可信任。
                //-------------------------------------------------
        }
    }
}
```

RefreshStatus 方法连接到设备以读取信息。如果设备为 CPU，CPU 可能受密码保护。RefreshStatus 方法以及与 CPU 进行内部连接的所有方法都需要一个密码参数。
示例中显示了 IProfinetDevice 类别。ICPU 在调用 RefreshStatus 方法前需要在设备中设置密码。使用 SetPassword (页 130)(EncryptedString) (页 62) 方法设置密码。

3.12.2.2 FirmwareUpdate 方法

返回类型	方法名称
Result	FirmwareUpdate

参数			
名称	数据类型	参数类型	说明
hardwareID	uint	In	模块的硬件标识符
bUpdateSameVersion	Bool	In	如果为真，该方法将继续进行更新。如果更新文件与模块的当前固件的版本相同，更新将继续进行。

此方法将更新设备上指定硬件项 (hardwareID) 的固件版本。hardwareID 可指定设备本身或同一机架中的模块。

某些设备不支持固件更新功能。检查特性 FirmwareUpdateAllowed 以确保当前设备支持此功能。

以下示例代码用于搜索特定 IP 地址处的设备以及更新设备中的固件：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (!scanResult.Succeeded)
    return;

uint targetIPAddress = 0xC0A80001; // 192.168.0.1
string updateFile = @"c:\myUpdates\6ES7 221-1BF32-0XB0 V02.00.00.upd";


IProfinetDevice dev = scannedDevices.FindDeviceByIP(targetIPAddress);
if (dev != null)
{
    Result retVal = dev.RefreshStatus();
```

```
 if (!retVal.Succeeded)
        return;

 dev.Selected = true;
    dev.SetFirmwareFile(updateFile);
    retVal = dev.FirmwareUpdate(dev.ID, true);
}
```

也可以使用 FirmwareUpdate 方法更新中心站中的模块的固件。以下代码显示了如何搜索特定地址的 CPU 以及如何搜索该 CPU 上具有特定订货号的模块。随后更新符合搜索条件的模块中的固件：

```
Network myNetwork = new Network();
IProfinetDeviceCollection scannedDevices;
IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
uint targetIPAddress = 0xC0A80001; // 192.168.0.1
string targetModule = @"6ES7 221-1BF32-0XB0";
string updateFile = @"c:\myUpdates\6ES7 221-1BF32-0XB0 V02.00.00.upd";

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);

if (!scanResult.Succeeded)
    return;

IProfinetDevice dev = scannedDevices.FindDeviceByIP(targetIPAddress);
if (dev != null)
{

    Result retVal = dev.RefreshStatus();
    if (!retVal.Succeeded)
        return;

    //---------------------------------
    // 搜索 CPU 上的模块
    //---------------------------------
    IModuleCollection mods = dev.Modules;
    foreach (IModule mod in mods)
    {
```

```
            if (mod.ArticleNumber == targetModule)
            {
                mod.Selected = true;
                mod.SetFirmwareFile(updateFile);

                //----------------------------------------
                // 更新符合条件的模块的固件
                //----------------------------------------
                dev.FirmwareUpdate(mod.ID, true);
            }
        }
    }
```

请注意，`FirmwareUpdate` 方法在 CPU 上调用。传递给方法的 `hardwareID` 指示要更新的模块。

说明

Classic 和 Plus 固件更新文件

有两种不同类型的固件更新文件：

- Classic 固件更新文件夹中包含组成固件更新的多个文件。此文件夹中的 header.upd 或 cpu_hd.upd 是传递给 `FirmwareUpdate` 方法的文件。
- Plus 固件更新文件是单个更新文件。此文件是传递给 `FirmwareUpdate` 方法的文件。

3.12.2.3 Identify 方法

返回类型	方法名称
Result	Identify

此方法会使特定网络设备的设备 LED 或 HMI 屏幕闪烁。闪烁光有助于识别设备的物理位置。

以下示例用于使 IP 地址为 192.168.0.1 的设备的 LED 或画面闪烁：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
    //------------------------------------------------------------
    // 搜索该 IP 的设备，使其 LED/HMI 屏幕闪烁
    //------------------------------------------------------------
    IProfinetDevice dev = scannedDevices.FindDeviceByIP(targetIPAddress);
    if (dev != null)
    {
        retVal = dev.Identify();
    }
}
```

3.12.2.4 ResetCommunicationParameters 方法

返回类型	方法名称
Result	ResetCommunicationParameters

使用此方法将 PROFINET 设备的通信参数复位为出厂设置。这会设置下列参数：

- 将 NameOfStation 设置为 ""（空字符串）
- 将 IP 套件参数设置为 0.0.0.0
- 将 DHCP 参数（如果可用）设置为出厂值
- 将所有 P Dev 参数（PD IR 数据、PD 端口数据调整、PD 接口 MRP 数据调整 ...）均设置为出厂值
- 将 SMNP 调整的参数，如 MIB-II 的 sysContact、sysName 和 sysLocation 均设置为出厂值

以下示例将针对特定 IP 地址的设备调用 ResetCommunicationParameters 方法：

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


IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
    //------------------------------------------------------------------------
    // 搜索该 IP 的设备，并将其复位
    //------------------------------------------------------------------------
    IProfinetDevice dev = scannedDevices.FindDeviceByIP(targetIPAddress);
    if (dev != null)
    {
         Result retVal = dev.ResetCommunicationParameters();
    }
}
```

说明

除非将 CPU 配置为智能设备，否则无法使用此方法复位 CPU。ICPU 接口 (页 123)提供了 ResetToFactoryDefaults 方法 (页 154)来复位 CPU。

3.12.2.5 SetIP 方法

返回类型	方法名称
Result	SetIP

参数			
名称	数据类型	参数类型	说明
nIP	uint	In	新编码的 IP 地址
nSubnet	uint	In	新编码的子网地址
nGateway	uint	In	新编码的网关地址

此方法用于设置或修改设备的 IP 地址。

要使此操作成功，必须为设备端口组态设置“直接在设备上设置 IP 地址”选项。

以下示例用于搜索指定 MAC 地址处的设备并设置其 IP 地址：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
    //-----------------------------------------------
    // 搜索此 MAC 地址处的设备并设置 IP
    //-----------------------------------------------
    IProfinetDevice dev = scannedDevices.FindDeviceByMAC(targetMACAddress);
    if (dev != null)
    {
        Result retVal = dev.SetIP(0xC0A80001, 0xFFFFFF00, 0x0);
    }
}
```

将地址从字符串格式转换为编码的 uint 格式

SetIP 方法要求地址为编码格式（如上所示）。可以使用以下 C# 代码将地址从字符串格式转换为编码的 uint 格式：

```
string userEnteredAddress = @"192.168.0.1"; // 示例

//--------------------------------
// 将字符串地址转换为 uint
//--------------------------------

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

uint encodedIp = BitConverter.ToUInt32(bytes, 0); // 可用的编码 IP 地址
```

3.12.2.6 SetProfinetName 方法

返回类型	方法名称
Result	SetProfinetName

参数			
名称	数据类型	参数类型	说明
strName	String	In	PROFINET 站的新名称

此方法用于设置或修改设备的 PROFINET 站名称。为确保此操作可成功执行，必须在 STEP 7 项目中将设备端口配置为“直接在设备上设置 PROFINET 名称”(PROFINET name is set directly on the device) 选项：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
    //------------------------------------------------
    // 搜索此 MAC 地址处的设备并设置 PROFINET 名称
    //------------------------------------------------
    IProfinetDevice dev = scannedDevices.FindDeviceByMAC(targetMACAddress);
    if (dev != null)
    {
        Result retVal = dev.SetProfinetName("new name");
    }
}
```

3.12.2.7 ValidateIPAddressSubnet 方法

返回类型	方法名称
Result	ValidateIPAddressSubnet

参数			
名称	数据类型	参数类型	说明
nIP	uint	In	IP 地址
nSubnetMask	uint	In	子网掩码

使用此方法验证 IP 地址和子网掩码组合是否兼容。

以下示例用于搜索指定 MAC 地址处的设备，并验证设备的 IP 地址和子网掩码是否兼容：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
    //------------------------------------------------
    // 搜索此 MAC 地址处的设备并设置 IP
    //------------------------------------------------
    IProfinetDevice dev = scannedDevices.FindDeviceByMAC(targetMACAddress);

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

3.12.2.8 ValidatePROFINETName 方法

返回类型	方法名称
Result	ValidatePROFINETName

参数			
名称	数据类型	参数类型	说明
strName	string	In	要验证的 PROFINET 名称

此方法用于验证提供的 PROFINET 名称。

以下示例用于搜索指定 MAC 地址处的设备，并验证给定 PROFINET 名称的有效性，然后将其分配至设备：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
    //------------------------------------------------
    // 搜索此 MAC 地址处的设备并设置 IP
    //------------------------------------------------
    IProfinetDevice dev = scannedDevices.FindDeviceByMAC(targetMACAddress);

    if (dev != null)
    {
        string name = "ValidName";
        Result retVal = dev.ValidatePROFINETName(name);
        if (retVal.Succeeded)
        {
            retVal = dev.SetProfinetName(name);
        }
    }
}
```

3.12.3 IProfinetDevice 事件

3.12.3.1 DataChanged 事件

IProfinetDevice 接口支持 DataChanged 事件。

此事件允许程序监视由 API 执行的其他操作是否引起了网络中给定设备的变更。例如，当程序保持对特定 IProfinetDevice 的引用时，可以检测此设备的变更。

对于网络中的每个设备，DataChanged 事件都附加了如下示例的代码：

```
private void AttachEvents(IProfinetDeviceCollection devices)
{
  foreach (IProfinetDevice dev in devices)
  {
    dev.DataChanged += new DataChangedEventHandler(Dev_DataChanged);
  }
}

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

private void Dev_DataChanged(object sender, DataChangedEventArgs e)
{
  if (e.Type == DataChangedType.OperatingState)
  {
   // 设备操作状态已更改。在此响应更改。
  }
}
```

此时，当 API 的任何操作引起设备的操作模式发生变化时，将调用 Dev_DataChanged 方法。

说明

DataChanged 事件不会主动监视实时网络，而会监视 IProfinetDevice 的属性。仅当此对象的状态发生变化时才会触发此事件。

DataChangedEventArgs 类

DataChanged 事件处理程序接收 DataChangedEventArgs 对象。如以上示例中所示，此类具有一个 DataChangedType (页 215) 数据类型的单一属性“Type”。

3.12.3.2 ProgressChanged 事件

IProfinetDevice 接口支持 ProgressChanged 事件。

此事件允许程序监视用时较长的方法的进程。例如 FirmwareUpdate 方法。

事件中附加了一个事件处理程序以便应用此事件。当操作进程发生变化时，将调用此事件处理程序。

以下示例说明了如何监视执行进程。该示例介绍更新网络设备的固件的方法。此操作可能需要相当长的时间。方法在 ProgressChanged 事件中定义并附加了一个事件处理程序以便监视操作进度。固件更新完成后，事件处理程序将从事件中分离：

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

        ProgressChangedEventHandler(Dev_ProgressChanged);
        dev.SetFirmwareFile(updateFile);
        dev.FirmwareUpdate(dev.ID, true);

        dev.ProgressChanged -= new

        ProgressChangedEventHandler(Dev_ProgressChanged);
```

```
    }
}

private void Dev_ProgressChanged(object sender, ProgressChangedEventArgs e)
{
    IProfinetDevice device = sender as IProfinetDevice;
    double percent = 0;
    if (device != null)
    {
        if (e.Count != 0)
        {
            string sPercent = e.Index.ToString() + " %";
        }
    }
}
```

ProgressChangedEventArgs 类

`ProgressChanged` 事件处理程序接收 `ProgressChangedEventArgs` 对象。此对象具有以下属性：

属性名称	返回类型	说明
`Action`	`ProgressAction` (页 224)	当前操作说明
`Cancel`	`bool`	是否已取消操作？
`Count`	`int`	要传送的总数据量
`ID`	`uint`	硬件 ID
`Index`	`int`	当前已传送的数据量

3.13 IModuleCollection 类和模块属性

3.13.1 模块属性和 IModuleCollection 类

IProfinetDevice 接口提供有关站中的任何模块（如信号模块、信号板、CM 和 CP）的信息。模块特性返回这些模块的集合 (页 73)。

以下代码显示了如何访问已存在的给定 IProfinetDevice 的信息：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
    //-----------------------------------------------------
    // 确保信息为当前的且为完整的信息，
    // 首先调用 RefreshStatus()
    //-----------------------------------------------------
    Result retVal = scannedDevices[0].RefreshStatus();
    if (retVal.Succeeded)
    {
        //-------------------------------------------------------
        // 模块特性返回集合 IModule
        //-------------------------------------------------------
        IModuleCollection modules = scannedDevices[0].Modules;
        foreach (IModule mod in modules)
        {
            //-----------------------------------------------------
            // 获取站中每个模块的订货号
            //-----------------------------------------------------
            string displayArticleNum = mod.ArticleNumber;
        }
    }
}
```

3.13.2 IModule 接口

站中的每个模块都表示为一个 `IModule` 接口。该接口提供了设备可用属性的子集。

`IModule` 接口不提供方法。必须在设备上启动对模块的所有操作。
`IModule` 接口支持以下特性：

属性名称	返回类型	说明
`ArticleNumber`	`string`	模块的订单号。 也称为 MLFB 或“订货号”。
`Comment`	`string`	用户可通过此属性指定设备注释，并在 SIMATIC Automation Tool 用户界面中使用。此注释与 API 操作无关。
`Configured`	`bool`	当设备具有有效组态时为真
`Description`	`string`	硬件项描述，基于订货号。此描述与用户可在 TIA Portal 中看到的描述相同。 （即“CPU-1215 DC/DC/DC”）
`Failsafe`	`bool`	根据其 ArticleNumber，此设备是否为故障安全设备？
`FirmwareUpdateAllowed`	`bool`	该设备是否支持固件更新？
`FirmwareVersion`	`string`	设备的当前固件版本
`ID`	`uint`	工作站中每个设备和模块的唯一标识符。该标识符用作执行 FirmwareUpdate 时的唯一标识符。
`Name`	`string`	设备名称
`NewFirmwareNameErrorCode`	`Result`	附加至新固件名称的 `ErrorCode`
`NewFirmwareNameIsValid`	`bool`	固件文件对此设备或模块有效时为真
`FirmwareVersion`	`string`	此属性用于 SIMATIC Automation Tool 用户界面中。此属性与 API 操作无关。
`Selected`	`bool`	当前是否已选中设备？即 GUI 中的复选框状态。
`SerialNumber`	`string`	设备的唯一序列号
`Slot`	`uint`	硬件项的插槽号

属性名称	返回类型	说明
SlotName	string	此属性用于 SIMATIC Automation Tool 用户界面中。此属性与 API 操作无关。
StationNumber	uint	设备的站编号
SubSlot	uint	设备的子插槽。这与 S7-1200 SB 模块等可插拔子模块相关。
Supported	FeatureSupport	当前的 SIMATIC Automation Tool 操作是否支持检测到的网络设备？

3.14 ICPU 接口

3.14.1 识别 IProfinetDeviceCollection 中的 CPU 设备

该 ScanNetworkDevices 方法 (页 78) 会生成一个 IProfinetDeviceCollection。该集合包含工业网络上每台可访问设备的一个项。这些设备可能包括 CPU 和分散式 I/O 站。

IProfinetDevice 接口提供适用于各类设备的属性和方法。但是，其中有一些属性和方法是特定于 CPU 设备的。可使用 ICPU 接口访问这些属性和方法。

要确定给定的 IProfinetDevice 接口是否确实表示一个 CPU 设备，只需将其转换为 ICPU。若转换成功，则表明该网络设备是 CPU 设备，并且可以使用 ICPU 接口上的属性/方法。以下示例说明了此过程：

```
Network myNetwork = new Network();
IProfinetDeviceCollection scannedDevices;
IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
    foreach (IProfinetDevice dev in scannedDevices)
    {
        ICPU devAsCpu = dev as ICPU;
        if (devAsCpu != null)
        {
            //-----------------------------------------------------
            // 该设备为 CPU。
            // ICPU 接口可用于与之交互。
```

```
            //------------------------------------------------
        }
    }
}
```

说明

ICPU 接口继承自 IProfinetDevice。因此，IProfinetDevice 支持的所有属性和方法也都受 ICPU 支持。

说明

必须设置 Selected 和 SelectedConfirmed 标志，然后才能够通过 API 在设备中执行操作。来自 ICPU 接口的所有函数都需要 Selected。如果 ICPU 代表安全操作，则必须设置 SelectedConfirmed。安全相关操作包括程序更新、格式化存储卡、复位为出厂设置和恢复。

3.14.2 ICPU 属性

ICPU 接口对 IProfinetDevice 进行了扩展，增加了以下属性。这些属性是只读的。为确保其能够返回当前信息，程序代码应首先调用 RefreshStatus 方法。

属性名称	返回类型	说明
RemoteInterfaces	List<IRemoteInterface>	为 CPU 组态的所有远程 I/O 接口的列表。此属性的用法将在本文档的后续部分中介绍。
DataLogFolder	IRemoteFolder	CPU 的 SIMATIC 存储卡中有关所有数据日志的信息
RecipeFolder	IRemoteFolder	CPU 的 SIMATIC 存储卡中有关所有配方的信息
OperatingMode	OperatingState	表示 CPU 的当前模式。该值是只读的。
IdentityCrisis	bool	当无法确定设备的身份时为真
LastRefreshSuccessful	bool	当对 RefreshStatus 的上一次调用成功完成时为真

属性名称	返回类型	说明
SelectedConfirmed	bool	当用户从操作的确认对话框中重新选择一个或多个设备并确认操作时，要执行与安全相关的操作的方法必须将 SelectedConfirmed 标志设置为真。SelectedConfirmed 表示已选择且已确认操作。
Initialized	bool	当设备或模块具有有效组态时为真
InterfaceNumber	int	用于连接设备的端口
Password	EncryptedString	用于在设备上执行的函数中的 CPU 密码
PasswordProtectionLevel	ProtectionLevel	合法 CPU 密码的保护等级
Protected	bool	CPU 当前是否受密码保护。这表示需要使用密码才能访问部分或全部功能，具体取决于访问级别。
PasswordValid	bool	对 SetPassword() 的调用是否有效？

另请参见 RemoteInterfaces 属性 (页 168)

3.14.3 ICPU 标志

3.14.3.1 程序更新标志

要在设备上成功执行与安全相关的函数，需要从设备中获取更多信息。下列标志确保可在安全设备中正确且安全地执行“程序更新”(Program Update) 函数。

属性名称	返回类型	说明
NewProgramNamePassword	EncryptedString	用于在程序更新完成后尝试建立连接的 CPU 密码。 通过使用 SetProgramPassword(EncryptedString) 来设置值。
HasSafetyProgram	bool	设备中具有安全程序时布尔值置 1。这可在连接到 CPU 时确定。
NewProgramNameIsValid	bool	当使用有效程序调用 SetProgramFolder 时为真。
NewProgramNameIsSafety	bool	当使用有效的安全程序调用 SetProgramFolder 方法时为真。
NewProgramNameHasSafetyPassword	bool	当使用有效的安全程序调用 SetProgramFolder 方法时为真。 打开标准程序时为假。
NewProgramNamePasswordIsValid	bool	当使用有效密码调用 SetProgramPassword 方法时为真。
NewProgramNamePasswordIsSafety	bool	当使用有效密码调用 SetProgramPassword 方法且新程序的密码具有安全的 F-CPU 密码时为真。
ProgramUpdateSucceeded	bool	当 ProgramUpdate 方法成功时为真。 程序更新仍可返回错误。
NewProgramNamePasswordPresent	bool	当调用 SetProgramFolder 方法且程序受密码保护时为真
NewProgramNamePasswordLevel	ProtectionLevel	用于确定新程序的 CPU 密码的保护等级。
NewProgramName	string	用于确定新程序的名称。

属性名称	返回类型	说明
NewProgramFolder	string	用于确定新程序的文件夹位置。 该值通过 SetProgramFolder 方法设置。
NewProgramNameFSignature	uint	用于确定新项目的 FSignature 。 在比较流程中用于确定 ProgramUpdate 是否已成功完成
NewProgramNameIP	uint	存储在新程序中的 IP 地址
NewProgramNameSubnetMask	uint	新程序中设备的子网掩码
NewProgramNameGateway	uint	新程序中设备的网关
NewProgramNameErrorCode	Result	验证新程序时用于发现可能存在的问题（例如程序是否对设备无效或者程序中的 IP 是否已经存在于网络上）的可行方式
NewProgramNamePasswordErrorCode	Result	存储上一次调用 SetProgramPassword 时的错误代码

3.14.3.2 恢复标志

API 中包括以下标志以便能够在安全设备中正确且安全地执行“从备份中恢复”(Restore from Backup) 函数。备份文件不包含所有程序更新信息，因为备份文件的内容与程序文件不同。

属性名称	返回类型	说明
`NewRestoreNamePassword`	`EncryptedString`	用于在恢复完成后尝试建立连接的 CPU 密码。 通过使用 `SetBackupFilePassword` (EncryptedString) 来设置值。
NewRestoreNameIsValid	bool	当调用 `SetBackupFile` 方法且恢复文件有效时为真
NewRestoreNameIsSafety	bool	当调用 `SetBackupFile` 方法且恢复文件为安全的恢复文件时为真
NewRestoreNamePasswordIsValid	bool	当调用 `SetBackupFilePassword` 包含有效密码时为真
NewRestoreNamePasswordIsSafety	bool	当调用 `SetBackupFilePassword` 包含有效的安全密码时为真
RestoreSucceeded	bool	`Restore` 操作是否成功？
NewRestoreName	string	用于确定新程序的名称。
NewRestoreFile	string	用于确定新程序的文件位置。 该值通过 `SetBackupFile` 方法设置
NewRestoreNameFSignature	uint	用于确定新项目的 FSignature。 在比较流程中用于确定 `Restore` 是否已成功完成
NewRestoreNameErrorCode	Result	验证新程序时用于发现可能存在的问题（例如程序是否对设备无效或与设备不兼容）的可行方式

3.14.3.3 功能标志

SIMATIC Automation Tool 的 ICPU 接口和 IHMI 接口中包括功能标志。这些标志的返回类型是 bool。

属性名称	返回类型	说明
ChangeModeAllowed	bool	CPU 支持模式更改操作时为真。（RUN 和 STOP）
BackupAllowed	bool	CPU 支持备份操作时为真
MemoryResetAllowed	bool	CPU 支持复位为出厂操作时为真
ProgramUpdateAllowed	bool	CPU 支持程序更新操作时为真
RestoreAllowed	bool	CPU 支持恢复操作时为真
FormatMCAllowed	bool	CPU 支持格式化存储卡操作时为真
PasswordAllowed	bool	CPU 支持使用密码时为真
RemoteRecipesAllowed	bool	CPU 支持配方操作时为真
RemoteDataLogsAllowed	bool	CPU 支持数据日志操作时为真
ServiceDataAllowed	bool	CPU 支持服务数据上传操作时为真
SetTimeAllowed	bool	CPU 支持设置及读取时间操作时为真
DiagBufferAllowed	bool	CPU 支持诊断缓冲区操作时为真

3.14.4 ICPU 方法

3.14.4.1 受保护的 CPU 和密码

ICPU 接口上的多数操作都需要与 CPU 进行合法连接，并且可能需要输入密码。因此，ICPU 接口上的大多数方法都需要密码参数。

3.14.4.2 SetPassword 方法

返回类型	方法名称
Result	SetPassword

参数			
名称	**数据类型**	**参数类型**	**说明**
password	EncryptedString	In	为用于执行操作的对象设置 CPU 密码

SetPassword 方法设置 CPU 密码。调用 SetPassword 方法时，请在尝试设置密码之前检查 dev.Protected 特性。如果设备是受保护的，则 SetPassword 方法返回一个错误。以下示例显示 SetPassword 方法的使用：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
    //------------------------------------------------
    // 搜索此 MAC 地址处的设备和 SetPassword
    //------------------------------------------------
    ICPU dev = scannedDevices.FindDeviceByMAC(targetMACAddress) as ICPU;
    if (dev != null)
    {
        if (dev.Protected)
        {
```

```
            Result retVal = dev.SetPassword(new EncryptedString("Password"));
        }
    }
}
```

3.14.4.3 SetProgramFolder 方法

返回类型	方法名称
Result	SetProgramFolder

参数			
名称	数据类型	参数类型	说明
strFolder	string	In	设置下载程序的文件夹位置

该方法会在 ICPU 对象中设置以下标志：

- NewProgramFolder
- NewProgramName
- NewProgramNameIP
- NewProgramNameSubnetMask
- NewProgramNameGateway
- NewProgramNameIsValid

当对安全对象执行上述操作时，可以设置以下属性：

- NewProgramNameIsSafety
- NewProgramNameHasSafetyPassword

以下示例说明了如何在设备中设置程序文件夹：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
    //-----------------------------------------------
    // 搜索此 MAC 地址处的设备和 SetProgramFolder
    //-----------------------------------------------
    ICPU dev = scannedDevices.FindDeviceByMAC(targetMACAddress) as ICPU;
    if (dev != null)
    {
        Result retVal = dev.SetProgramFolder(@"C:\MyFolder");
        if (retVal.Error == ErrorCode.ProgramPasswordNeeded)
        {
            retVal = dev.SetProgramPassword(new EncryptedString("Password"));
        }
    }
}
```

说明

如果选择的程序具有密码，`SetProgramFolder` 会返回一个错误。如果 `SetProgramFolder` 返回错误，则调用 `SetProgramPassword`。

3.14.4.4 SetProgramPassword 方法

返回类型	方法名称
Result	SetProgramPassword

参数			
名称	数据类型	参数类型	说明
password	EncryptedString	In	设置要在 ProgramUpdate 期间传递给 CPU 的项目的 CPU 密码

执行 ProgramUpdate 后，应用程序会尝试重新连接至设备。如果位于 CPU 中的程序受密码保护，则设置更新的 CPU 密码可重新访问设备。

该方法会在 ICPU 对象上设置以下标志：

- NewProgramNamePasswordIsValid
- NewProgramNamePasswordIsSafety
- NewProgramNamePasswordLevel

以下示例说明了如何在设备中设置更新的 CPU 密码：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
    //-----------------------------------------------
    // 搜索此 MAC 地址处的设备和 SetProgramFolder
    //-----------------------------------------------
    ICPU dev = scannedDevices.FindDeviceByMAC(targetMACAddress) as ICPU;
    if (dev != null)
    {
        Result retVal = dev.SetProgramFolder(@"C:\MyFolder");
        if (retVal.Error == ErrorCode.ProgramPasswordNeeded)
        {
            retVal = dev.SetProgramPassword(new EncryptedString("Password"));
        }
    }
}
```

3.14.4.5 ProgramUpdate 方法

返回类型	方法名称
Result	ProgramUpdate

此方法用于在 CPU 上执行程序更新。

必须满足以下前提条件才能更新程序：

- 必须选中设备。
- 如果设备为故障安全设备，则 SelectedConfirmed 必须为真。
- 必须设置对象的 NewProgramFolder。
- 如果新程序包含 CPU 密码，则 NewProgramPasswordPresent 必须为真并已通过 SetProgramPassword. 设置值
- 设备必须支持程序更新 (ProgramUpdateAllowed)。

以下示例用于搜索特定 IP 地址处 CPU 的 IProfinetDeviceCollection 以及更新该 CPU 中的程序：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
    ICPU cpu = scannedDevices.FindDeviceByMAC(targetMACAddress) as ICPU;
    if (cpu != null && cpu.ProgramUpdateAllowed == true)
    {
        // 选中 CPU 以进行更新
        cpu.Selected = true;

        retVal = cpu.SetProgramFolder(@"C:\MyFolder");
        if (retVal.Error == ErrorCode.ProgramPasswordNeeded)
        {
            retVal = cpu.SetProgramPassword(new EncryptedString("Password"));
            if (retVal.Failed == true)
                return;
```

```
        }
        if (retVal.Failed == true)
            return;

        // IP 地址是否唯一?
        if (cpu.DuplicateIP == true)
            return;

        // 设备是否受支持?
        if (cpu.Supported == false)
            return;

        // 设备是否已初始化?
        if (cpu.Initialized == false)
            return;

        // 是否为故障安全设备?
        if (cpu.Failsafe == true)
        {
            ConfirmationType type =
cpu.DetermineConfirmationMessage(FailsafeOperation.ProgramUpdateOperation);

            // 验证类型并确认
            cpu.SelectedConfirmed = true;

            // 检查以确保可以更新
            if (cpu.HasSafetyProgram == true || cpu.Protected == true
||cpu.NewProgramNameIsSafety == true)
            {
                // 设备是否受密码保护?
                if (cpu.Protected == true)
                {
                  // 是否提供了有效密码?
                  if (cpu.PasswordValid == false)
                  return;

                  // 是否具备合法化安全等级?
                  bool bSufficientAccess = cpu.PasswordProtectionLevel ==
ProtectionLevel.Failsafe;
```

```
            if (bSufficientAccess == false)
              return;


        }
    }
}
else
{
    // 是否要加载安全程序?
    if (cpu.NewProgramNameHasSafetyPassword == true)
        return;
}

// 设备是否受密码保护?
if (cpu.Protected == true)
{
    // 是否提供了有效密码?
    if (cpu.PasswordValid == false)
        return;
```

```
            // 是否具有足够的合法访问等级以成功执行操作?
            bool bSufficientAccess = cpu.PasswordProtectionLevel ==
ProtectionLevel.Failsafe || cpu.PasswordProtectionLevel == ProtectionLevel.Full;
            if (bSufficientAccess == false)
                return;
        }
        // 执行程序更新
        retVal = cpu.ProgramUpdate();

        // 复位
        cpu.SelectedConfirmed = false;
    }
}
```

说明

传递至 `ProgramUpdate` 方法的文件夹名称应包含一个名为 SIMATIC.S7S 的文件夹。SIMATIC.S7S 文件夹包含要下载的程序。

3.14.4.6 SetBackupFile 方法

返回类型	方法名称
Result	SetBackupFile

参数			
名称	**数据类型**	**参数类型**	**说明**
`strFile`	`string`	In	设置备份文件的位置

该方法会在 ICPU 对象上设置以下标志：

- NewRestoreName
- NewRestoreFile
- NewRestoreNameIsValid
- NewRestoreNameIsSafety
- NewRestorenameFSignature

以下示例显示了设置备份文件路径的方式：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
    ICPU cpu = scannedDevices.FindDeviceByMAC(targetMACAddress) as ICPU;
    if (cpu != null && cpu.RestoreAllowed == true)
    {
        // 选中 CPU 以进行更新
        cpu.Selected = true;

        Result retVal = cpu.SetBackupFile(@"C:\MyFolder\Backup.s7pbkp");

        if (retVal.Failed == true)
            return;

        // IP 地址是否唯一？
        if (cpu.DuplicateIP == true)
            return;

        // 设备是否受支持？
        if (cpu.Supported == false)
            return;

        // 设备是否已初始化？
        if (cpu.Initialized == false)
            return;

        //是否为故障安全设备？
        if (cpu.Failsafe == true)
        {
            ConfirmationType type =
            cpu.DetermineConfirmationMessage(FailsafeOperation.RestoreOperation);
```

```
            // 验证类型并确认
            cpu.SelectedConfirmed = true;

            // 检查以确保可以更新
            if (cpu.HasSafetyProgram == true || cpu.Protected == true ||
cpu.NewRestoreNameIsSafety == true)
            {
                // 设备是否受密码保护?
                if (cpu.Protected == true)
                {
                    // 是否提供了有效密码?
                    if (cpu.PasswordValid == false)
                        return;

                    //是否具备相应的安全等级?
                    bool bSufficientAccess = cpu.PasswordProtectionLevel ==
ProtectionLevel.Failsafe;
                    if (bSufficientAccess == false)
                        return;
                }
            }
        }
        else
        {
            // 是否要加载安全程序?
            if (cpu.NewRestoreNameIsSafety == true)
                return;
        }

        // 设备是否受密码保护?
        if (cpu.Protected == true)
        {
            // 是否提供了有效密码?
            if (cpu.PasswordValid == false)
                return;

            // 是否具有足够的合法访问等级以成功执行操作?
            bool bSufficientAccess = cpu.PasswordProtectionLevel ==
```

```
ProtectionLevel.Failsafe || cpu.PasswordProtectionLevel == ProtectionLevel.Full;
            if (bSufficientAccess == false)
                return;
        }
        // 执行恢复
        retVal = cpu.Restore();

        // 复位
        cpu.SelectedConfirmed = false;
    }
}
```

说明

如果所选备份文件具有 CPU 密码，则 `SetBackupFile` 返回异常。必须调用 `SetBackupFilePassword` 方法并成功返回，之后才能调用 `Restore` 方法。

3.14.4.7 SetBackupFilePassword 方法

返回类型	方法名称
`Result`	`SetBackupFilePassword`

参数			
名称	**数据类型**	**参数类型**	**说明**
`password`	`EncryptedString`	In	设置要在恢复期间传递给 CPU 的项目的密码

执行恢复后，应用程序会尝试重新连接至设备。如果 CPU 中加载的程序受密码保护，则设置更新的 CPU 密码可重新访问设备。

该方法会在 `ICPU` 对象中设置以下标志：

- NewRestoreNamePassword

以下示例说明了如何在设备中设置更新的 CPU 密码：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
    //-----------------------------------------------
    // 搜索此 MAC 地址处的设备并设置备份文件密码
    //-----------------------------------------------
    ICPU dev = scannedDevices.FindDeviceByMAC(targetMACAddress) as ICPU;
    if (dev != null)
    {
        if (dev.Protected)
        {
            Result retVal = dev.SetBackupFile(@"C:\MyFolder\MyCPUBackupFile.s7pbkup");
            retVal = dev.SetBackupFilePassword(new EncryptedString("Password"));
        }
    }
}
```

3.14.4.8 Restore 方法（ICPU 接口）

返回类型	方法名称
Result	Restore

此方法用于从之前的 CPU 备份中恢复信息。某些 CPU 不支持备份/恢复功能。

要执行此操作，必须满足以下前提条件：

- 必须选中设备。
- 如果设备为故障安全设备，则 SelectedConfirmed 必须为真。
- 必须设置对象的 NewProgramFolder。
- 如果新程序包含 CPU 密码，则 NewProgramPasswordPresent 必须为真并已通过 SetProgramPassword 设置了值
- 必须支持 Restore 以便执行操作 (RestoreAllowed)。

以下示例用于搜索特定 IP 地址处 CPU 的 IProfinetDeviceCollection。成功搜索到后，检查受密码保护的 CPU 是否支持恢复功能，然后对不含 CPU 密码的备份文件调用 Restore 方法：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out devices);
if (scanResult.Succeeded)
{
    IProfinetDevice dev = devices.FindDeviceByIP(targetIPAddress);
    if (dev != null)
    {
        ICPU devAsCpu = dev as ICPU;
        if (devAsCpu != null && devAsCpu.RestoreAllowed)
        {
            retVal = devAsCpu.SetPassword(new EncryptedString("Password"));
            retVal = devAsCpu.SetBackupFile(bkFile);
            devAsCpu.Selected = true;
            if (devAsCpu.Failsafe)
```

```
                devAsCpu.SelectedConfirmed = true;

            retVal = devAsCpu.Restore();
        }
    }
}
```

3.14.4.9 Backup 方法（ICPU 接口）

返回类型	方法名称
Result	Backup

参数			
名称	数据类型	参数类型	说明
strFile	string	In	要存储备份文件的完全限定路径和文件名

此方法用于备份 CPU 中的数据。某些 CPU 不支持备份/恢复功能。可以检查 BackupAllowed 属性，以确保当前 CPU 支持此功能。

以下示例用于搜索特定 IP 地址处 CPU 的 IProfinetDeviceCollection。成功搜索到后，检查 CPU 是否支持备份功能，然后调用 Backup 方法：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out devices);
if (scanResult.Succeeded)
{
    foreach (IProfinetDevice dev in devices)
    {
        ICPU devAsCpu = dev as ICPU;
```

```
        if (devAsCpu != null && devAsCpu.IP == targetIPAddress &&
devAsCpu.BackupAllowed)
        {
            devAsCpu.Selected = true;
            retVal = devAsCpu.Backup(bkFile);
        }
    }
}
```

3.14.4.10 DownloadRecipe 方法

返回类型	方法名称
Result	DownloadRecipe

参数			
名称	数据类型	参数类型	说明
strFile	string	In	要从编程设备下载至 CPU 存储卡的配方文件的完整路径和文件名

此方法用于向 CPU 存储卡中添加或替换其中的配方 .CSV 文件。某些 CPU 不支持远程访问配方。可以检查 RemoteRecipesAllowed 属性，以确保当前 CPU 支持此功能。以下代码示例显示了如何向 CPU 存储卡写入配方：

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

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

```
        if ((devAsCpu != null) &&
            (devAsCpu.IP == targetIPAddress) &&
            (devAsCpu.RemoteRecipesAllowed))
        {
            retVal = devAsCpu.SetPassword(new EncryptedString("Password"));
            IRemoteFolder recipes = devAsCpu.RecipeFolder;
            recipes.Selected = true;
            recipes.SetRemoteFile(rcpFile);
            retVal = devAsCpu.DownloadRecipe(rcpFile);
        }
    }
}
```

说明

如果 CPU 存储卡上已存在具有相同名称的配方，则会替换已有配方。

3.14.4.11 DeleteDataLog 方法

返回类型	方法名称
Result	DeleteDataLog

参数			
名称	数据类型	参数类型	说明
strFileName	string	In	待从 CPU 存储卡中删除的数据日志文件的文件名

此方法用于从 CPU 存储卡中删除数据日志文件。

某些 CPU 不支持远程访问数据日志。检查 RemoteDataLogsAllowed 属性以确保当前 CPU 支持此功能。

以下代码示例实现了使用 DataLogFolder 属性迭代 CPU 存储卡中的所有数据日志。删除所有数据日志：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
    foreach (IProfinetDevice dev in devices)
    {
        ICPU devAsCpu = dev as ICPU;
        if (devAsCpu != null)
        {
            if (devAsCpu.RemoteDataLogsAllowed)
            {
                //----------------------------------------
                // 首先检查存储卡中的数据日志是否可用
                //----------------------------------------
                if (devAsCpu.DataLogFolder.Exists)
                {
                    devAsCpu.SetPassword(new EncryptedString("Password"));
                    //----------------------------------------
                    // 搜索所有数据日志文件
                    //----------------------------------------
                    foreach (IRemoteFile datalog in
                                    devAsCpu.DataLogFolder.Files)
                    {
                        datalog.Selected = true;
                        //----------------------------------------
                        // 删除数据日志
                        //----------------------------------------
                        devAsCpu.DeleteDataLog(datalog.Name);
                    }
                }
            }
        }
    }
}
```

3.14.4.12 DeleteRecipe 方法

返回类型	方法名称
Result	DeleteRecipe

参数			
名称	数据类型	参数类型	说明
strFileName	string	In	待从 CPU 存储卡中删除的配方文件的文件名

此方法用于从 CPU 存储卡中删除配方文件。

某些 CPU 不支持远程访问配方。检查 RemoteRecipesAllowed 属性以确保当前 CPU 支持此功能。

以下代码示例实现了使用 RecipeFolder 属性迭代 CPU 存储卡中的所有配方。删除所有配方：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
    foreach (IProfinetDevice dev in devices)
    {
        ICPU devAsCpu = dev as ICPU;
        if (devAsCpu != null)
        {
            if (devAsCpu.RemoteDataLogsAllowed)
            {
                //----------------------------------------
                // 首先检查存储卡中的配方是否可用
                //----------------------------------------
                if (devAsCpu.RecipeFolder.Exists)
                {
                    devAsCpu.SetPassword(new EncryptedString("Password"));
                    //----------------------------------------
                    // 搜索所有配方
```

```
                    //----------------------------------------
                    foreach (IRemoteFile recipe in devAsCpu.RecipeFolder.Files)
                    {
                        recipe.Selected = true;
                        //----------------------------------------
                        // 删除配方.
                        //----------------------------------------
                        devAsCpu.DeleteRecipe(recipe.Name);
                    }
                }
            }
        }
    }
}
```

3.14.4.13 GetCurrentDateTime 方法

返回类型	方法名称
Result	GetCurrentDateTime

参数			
名称	数据类型	参数类型	说明
DateTime	System.DateTime	Out	从 CPU 返回的当前日期和时间

此方法用于获取 CPU 的当前时间戳。

以下示例用于搜索特定 IP 地址处 CPU 的 `IProfinetDeviceCollection` 以及获取其当前日期和时间：

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

uint targetIPAddress = 0xC0A80001; // 192.168.0.1

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out devices);
if (!scanResult.Succeeded)
    return;

foreach (IProfinetDevice dev in devices)
{
    ICPU devAsCpu = dev as ICPU;
    if ((devAsCpu != null) && (devAsCpu.IP == targetIPAddress))
    {
        devAsCpu.SetPassword(new EncryptedString("Password"));
        devAsCpu.Selected = true;

        DateTime curTime = new DateTime();

        Result retVal = devAsCpu.GetCurrentDateTime(out curTime);
    }
}
```

3.14.4.14 GetDiagnosticsBuffer 方法

返回类型	方法名称
Result	GetDiagnosticsBuffer

参数			
名称	数据类型	参数类型	说明
DiagnosticsItems	List<DiagnosticsItem>	Out	诊断项的集合：集合中的每一项代表诊断缓冲区中的一个条目。
Language	Language	In	请求用于诊断缓冲区条目的语言。

此方法用于读取 CPU 的当前诊断条目。GetDiagnosticsBuffer 方法会返回 DiagnosticsItem (页 67) 对象的集合。以下示例用于搜索特定 IP 地址处 CPU 的 IProfinetDeviceCollection。成功搜索到后，将从 CPU 读取诊断信息。使用 Language enum (页 223) 参数获取采用特定语言的诊断条目：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
    foreach (IProfinetDevice dev in devices)
    {
        ICPU devAsCpu = dev as ICPU;
        if ((devAsCpu != null) && (devAsCpu.IP == targetIPAddress))
        {
            devAsCpu.SetPassword(new EncryptedString("Password"));
            devAsCpu.Selected = true;
            retVal = devAsCpu.GetDiagnosticsBuffer(out aLogs, Language.English);
            if (retVal.Succeeded)
            {
```

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

                    }
                }
            }
        }
    }
```

3.14.4.15 MemoryReset 方法

返回类型	方法名称
Result	MemoryReset

此方法用于在 CPU 上执行存储器复位。

以下示例用于搜索特定 IP 地址处 CPU 的 IProfinetDeviceCollection，并对此 CPU 调用 MemoryReset 方法：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
    IProfinetDevice dev = devices.FindDeviceByIP(targetIPAddress);
    if (dev != null)
    {
        ICPU devAsCpu = dev as ICPU;
        if (devAsCpu != null)
        {
            devAsCpu.SetPassword(new EncryptedString("Password"));
            devAsCpu.Selected = true;
            Result retVal = devAsCpu.MemoryReset();
        }
    }
}
```

3.14.4.16 ResetToFactoryDefaults 方法

返回类型	方法名称
Result	ResetToFactoryDefaults

参数			
名称	数据类型	参数类型	说明
password	EncryptedString	In	此方法用于与设备建立合法连接。因此，可能需要密码。

该方法用于将 CPU 复位为其出厂默认值。

以下示例用于搜索特定 IP 地址处 CPU 的 IProfinetDeviceCollection 并调用 ResetToFactoryDefaults 方法。对于故障安全设备，必须将 SelectedConfirmed 标志设置为真：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
    ICPU cpu = scannedDevices.FindDeviceByMAC(targetMACAddress) as ICPU;
    if (cpu != null && cpu.ResetToFactoryAllowed == true)
    {
        // 选中 CPU 以进行更新
        cpu.Selected = true;

        // IP 地址是否唯一?
        if (cpu.DuplicateIP == true)
            return;

        // 设备是否受支持?
        if (cpu.Supported == false)
            return;
```

```
        // 设备是否已初始化?
        if (cpu.Initialized == false)
            return;

        // 是否为故障安全设备?
        if (cpu.Failsafe == true) { ConfirmationType type =
cpu.DetermineConfirmationMessage(FailsafeOperation.ResetToFactoryOperation);
        // 验证类型并确认
        cpu.SelectedConfirmed = true;

        // 检查以确保可以更新
        if (cpu.HasSafetyProgram == true || cpu.Protected == true ||
cpu.NewRestoreNameIsSafety == true)
        {
            // 设备是否受密码保护?
            if (cpu.Protected == true)
            {
                // 是否提供了有效密码?
                if (cpu.PasswordValid == false)
                    return;

                // 是否具备相应的安全等级?
                bool bSufficientAccess = cpu.PasswordProtectionLevel ==
ProtectionLevel.Failsafe;
                if (bSufficientAccess == false)
                    return;
            }
        }
    }
    else
    {
        // 是否为要加载的安全程序?
        if (cpu.NewRestoreNameIsSafety == true)
            return;
    }
    // 设备是否受密码保护?
    if (cpu.Protected == true)
    {
        // 是否提供了有效密码?
```

```
            if (cpu.PasswordValid == false)
                return;

            // 是否具有足够的合法访问等级以成功执行操作？
            bool bSufficientAccess = cpu.PasswordProtectionLevel ==
ProtectionLevel.Failsafe || cpu.PasswordProtectionLevel == ProtectionLevel.Full;
            if (bSufficientAccess == false)
                return;
        }
            // 执行复位为出厂默认值
            Result retVal = cpu.ResetToFactoryDefaults();

            // 复位
            cpu.SelectedConfirmed = false;
        }
}
```

3.14.4.17 SetOperatingState 方法

返回类型	方法名称
Result	SetOperatingState

参数			
名称	数据类型	参数类型	说明
nRequestState	OperatingStateREQ	In	新操作状态

此方法用于切换 CPU 的操作状态。

某些 CPU 不支持此功能。检查 C`hangeModeAllowed` 属性以确保当前 CPU 支持此功能。

以下示例用于搜索特定 IP 地址处 CPU 的 `IProfinetDeviceCollection`。成功搜索到后，检查 CPU 是否支持切换模式功能，然后将 CPU 置于 RUN 模式：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
    IProfinetDevice dev = devices.FindDeviceByIP(targetIPAddress);
    if (dev != null)
    {
        ICPU devAsCpu = dev as ICPU;
        if ((devAsCpu != null) && (devAsCpu.ChangeModeAllowed))
        {
            devAsCpu.SetPassword(new EncryptedString("Password"));
            devAsCpu.Selected = true;
            Result retVal = devAsCpu.SetOperatingState(OperatingStateREQ.Run);
        }
    }
}
```

3.14.4.18 SetCurrentDateTime 方法

返回类型	方法名称
Result	SetCurrentDateTime

参数			
名称	数据类型	参数类型	说明
password	EncryptedString	In	此方法用于与设备建立合法连接。因此，可能需要密码
time	System.DateTime	In	新的 CPU 当前时间值

此方法用于设置 CPU 的当前时间。已组态的时间转换规则不受此操作影响。因此，指定的 DateTime 值将基于 UTC 时间非本地时间。

以下示例遍历整个工业网络，并将每个 CPU 设备的当前时间设置为编程设备的当前时间：

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

Result retVal = new Result();


IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
    foreach (IProfinetDevice dev in devices)
    {
        ICPU devAsCpu = dev as ICPU;
        if (devAsCpu != null)
        {
            devAsCpu.SetPassword(new EncryptedString("Password"));
            devAsCpu.Selected = true;
            retVal = devAsCpu.SetCurrentDateTime(DateTime.UtcNow);
        }
    }
}
```

3.14.4.19 UploadDataLog 方法

返回类型	方法名称
Result	UploadDataLog

参数			
名称	数据类型	参数类型	说明
strFileName	string	In	要从 CPU 的可移动 SIMATIC 存储卡上传的数据日志的文件名
strDestinationFolder	string	In	存储上传的数据日志文件的完全限定路径

此方法用于将指定数据日志文件的副本从 CPU 存储卡上传至编程设备。某些 CPU 不支持远程访问数据日志。检查 RemoteDataLogsAllowed 属性以确保当前 CPU 支持此功能。以下代码示例实现了使用 DataLogFolder 属性迭代 CPU 存储卡中的所有数据日志。将所有数据日志的副本上传至 C:\MyDataLogs 文件夹中：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
    foreach (IProfinetDevice dev in devices)
    {
        ICPU devAsCpu = dev as ICPU;
        if (devAsCpu != null)
        {
            devAsCpu.SetPassword(new EncryptedString("Password"));

            if (devAsCpu.RemoteDataLogsAllowed)
            {
                //----------------------------------------
                // 首先检查存储卡中的数据日志是否可用
                //----------------------------------------
                if (devAsCpu.DataLogFolder.Exists)
```

```
                    {
                        //----------------------------------------
                        // 搜索所有数据日志文件
                        //----------------------------------------
                        foreach (IRemoteFile datalog in devAsCpu.DataLogFolder.Files)
                        {
                            datalog.Selected = true;
                            //----------------------------------------
                            // 上传每个数据日志的副本
                            //----------------------------------------
                            devAsCpu.UploadDataLog(datalog.Name, @"C:\MyDataLogs");
                        }
                    }
                }
            }
        }
    }
```

3.14.4.20 UploadRecipe 方法

返回类型	方法名称
Result	UploadRecipe

参数			
名称	数据类型	参数类型	说明
strFileName	string	In	要从 CPU 存储卡中上传的配方的文件名
strDestinationFolder	string	In	写入上传的配方文件的完全限定路径

此方法用于从 CPU 存储卡上传配方文件副本。某些 CPU 不支持远程访问配方。检查 RemoteRecipesAllowed 属性以确保当前 CPU 支持此功能。

以下代码示例实现了使用 RecipeFolder 属性迭代 CPU 存储卡中的所有配方。将所有配方文件的副本上传至 C:\MyRecipes 文件夹中：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out devices);
if (scanResult.Succeeded)
{
    foreach (IProfinetDevice dev in devices)
    {
        ICPU devAsCpu = dev as ICPU;
        if (devAsCpu != null)
        {
            devAsCpu.SetPassword(new EncryptedString("Password"));
            if (devAsCpu.RemoteRecipesAllowed)
            {
                //----------------------------------------
                // 首先检查存储卡中的配方是否可用。
                //----------------------------------------
                if (devAsCpu.RecipeFolder.Exists)
                {
```

```
                    //----------------------------------------
                    // 搜索所有配方文件
                    //----------------------------------------
                    foreach (IRemoteFile recipe in devAsCpu.RecipeFolder.Files)
                    {
                        recipe.Selected = true;
                        //----------------------------------------
                        // 上传每个配方的副本。
                        //----------------------------------------
                        devAsCpu.UploadRecipe(recipe.Name, @"C:\MyRecipes");
                    }
                }
            }
        }
    }
}
```

3.14.4.21 UploadServiceData 方法

返回类型	方法名称
Result	UploadServiceData

参数			
名称	数据类型	参数类型	说明
strPath	string	In	包含程序卡内容的文件夹的完全限定路径
format	TimeFormat	In（可选）	日期和时间的显示格式。可能值为 UTC 和“本地”。若未提供，则格式为“本地”。

此方法可以从存在故障的 CPU 中上传服务数据。

以下示例用于搜索特定 IP 地址处 CPU 的 IProfinetDeviceCollection。然后检查 CPU 的当前 OperatingState。如果 CPU 存在故障，则将上传服务数据：

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

uint targetIPAddress = 0xC0A80001; // 192.168.0.1
string strDiagFolder = @"c:\Diagnostics";

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
    IProfinetDevice dev = devices.FindDeviceByIP(targetIPAddress);
    if (dev != null)
    {
        ICPU devAsCpu = dev as ICPU;

        if (devAsCpu != null)
        {
            devAsCpu.SetPassword(new EncryptedString("Password"));
            devAsCpu.Selected = true;
            if (devAsCpu.OperatingMode == OperatingState.Defective)
            {
                // 获取采用默认本地时间戳格式的服务数据
                Result retVal = devAsCpu.UploadServiceData(strDiagFolder);

                // 获取采用 UTC 时间戳格式的服务数据
                Result retVal = devAsCpu.UploadServiceData(strDiagFolder,
TimeFormat.UTC);
            }
        }
    }
}
```

3.14.4.22 FormatMemoryCard 方法

返回类型	方法名称
Result	FormatMemoryCard

此方法用于格式化插入 CPU 的可移动 SIMATIC 存储卡。

以下示例用于搜索特定 IP 地址处 CPU 的 IProfinetDeviceCollection。然后格式化设备的存储卡。对于故障安全设备，必须将 SelectedConfirmed 标志设置为真：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
    ICPU cpu = scannedDevices.FindDeviceByMAC(targetMACAddress) as ICPU;
    if (cpu != null && cpu.FormatMCAllowed == true)
    {
        // 选中 CPU 以进行更新
        cpu.Selected = true;

        // IP 地址是否唯一?
        if (cpu.DuplicateIP == true)
            return;

        // 设备是否受支持? ?
        if (cpu.Supported == false)
            return;

        // 设备是否已初始化?
        if (cpu.Initialized == false)
            return;

        // 是否为故障安全设备?
        if (cpu.Failsafe == true)
        {
            ConfirmationType type =
```

```
cpu.DetermineConfirmationMessage(FailsafeOperation.ResetToFactoryOperation);

        // 验证类型并确认
        cpu.SelectedConfirmed = true;

        // 检查以确保可以更新
        if (cpu.HasSafetyProgram == true || cpu.Protected == true ||
cpu.NewRestoreNameIsSafety == true)
        {
            // Is the device password protected?if (cpu.Protected == true)
            {
                // 是否提供了有效密码?
                if (cpu.PasswordValid == false)
                    return;

                // 是否具备相应的安全等级?
                bool bSufficientAccess = cpu.PasswordProtectionLevel ==
ProtectionLevel.Failsafe;
                if (bSufficientAccess == false)
                    return;
                }
        }
    }
    else
    {
        // 是否为要加载的安全程序?
        if (cpu.NewRestoreNameIsSafety == true)
            return;
    }

    // 设备是否受密码保护?
    if (cpu.Protected == true)
    {
        // 是否提供了有效密码?
        if (cpu.PasswordValid == false)
            return;

        // 是否具有足够的合法访问等级以成功执行操作?
        bool bSufficientAccess = cpu.PasswordProtectionLevel ==
```

```
ProtectionLevel.Failsafe || cpu.PasswordProtectionLevel == ProtectionLevel.Full;
        if (bSufficientAccess == false)
            return;
    }
    // 格式化存储卡
    Result retVal = cpu.FormatMemoryCard();

    // 复位
    cpu.SelectedConfirmed = false;
    }
}
```

3.14.4.23 DetermineConfirmationMessage

返回类型	方法名称
ConfirmationType	DetermineConfirmationMessage

参数			
名称	数据类型	参数类型	说明
operation	FailsafeOperation	In	评估的操作

当用户尝试执行安全相关操作时，此方法用于确定确认消息中包含的内容。

ConfirmationType 类包含一系列汉明码，其中每个汉明码都代表一条应向用户显示的错误消息，如下所示：

ConfirmationType	要显示的消息
SafetyPasswordIsBeingUsed	将使用安全 CPU 密码启动对标准程序的操作。
DeletingExistingSafetyProgram	将删除现有安全程序。
ReplacingExistingSafetyProgram	将使用其它安全程序更新现有安全程序。
ReplacingExistingSafetyProgramWithNonSafetyProgram	现有安全程序将替换为标准程序。
LoadingSafetyProgram	安全程序将首次加载。

以下示例用于搜索特定 IP 地址处 CPU 的 IProfinetDeviceCollection。此方法用于在执行安全相关操作前，在用户确认对话框中显示有关安全相关状态的消息：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out devices);
if (scanResult.Succeeded)
{
    IProfinetDevice dev = devices.FindDeviceByIP(targetIPAddress);
    if (dev != null)
    {
        ICPU devAsCpu = dev as ICPU;
        if (devAsCpu != null)
        {
            devAsCpu.SetPassword(new EncryptedString("Password"));
            devAsCpu.Selected = true;
            if (devAsCpu.Failsafe)
            {
                devAsCpu.SelectedConfirmed = true;
            }
            ConfirmationType confirm =
devAsCpu.DetermineConfirmationMessage(FailsafeOperation.FormatMCOperation);

            if (confirm == ConfirmationType.DeletingExistingSafetyProgram)
                devAsCpu.FormatMemoryCard();
        }
    }
}
```

3.14.5 RemoteInterfaces 属性

3.14.5.1 分散式 I/O 模块

每个 CPU 可支持多个分散式 I/O 接口。可通过 ICPU 接口 (页 124)的 `RemoteInterfaces` 属性获取有关附加在这些远程接口中的设备的信息。

以下示例说明了如何访问网络中所有 CPU 的此类信息：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
    foreach (IProfinetDevice dev in scannedDevices)
    {
        ICPU devAsCpu = dev as ICPU;

        List<IRemoteInterface> decentalNets = devAsCpu.RemoteInterfaces;
        foreach (IRemoteInterface net in decentalNets)
        {
            //------------------------------------------------------------
            // 检查远程接口
            //------------------------------------------------------------
        }
    }
}
```

3.14.5.2 IRemoteInterface 属性

`IRemoteInterface` 接口支持以下属性。这些属性是只读的。

属性名称	返回类型	说明
`Devices`	`List<IBaseDevice>`	所有连接至此远程接口的分散式 I/O 站的列表
`InterfaceType`	`RemoteInterfaceType`	此远程接口的通信协议 另请参见 RemoteInterfaceType 枚举 (页 224)
`Name`	`string`	远程接口的已组态名称

可使用 `Devices` 属性遍历分散式网络。分散式网络中的每个设备都由一个 `IBaseDevice` 接口表示。该接口具有一个可用于 `IProfinetDevice` 的子属性集，并提供这些设备可用的有限 SIMATIC Automation Tool API 功能。

`IBaseDevice` 接口上提供以下属性：

属性名称	返回类型	说明
`ArticleNumber`	`string`	模块的订单号。也称为 MLFB 或订货号。
`Comment`	`string`	用户可通过此属性指定设备注释。并在 SIMATIC Automation Tool 用户界面中使用。此属性与 API 操作无关。
`Configured`	`bool`	设备是否具有有效组态？
`Description`	`string`	基于订货号的硬件项描述。此描述与用户在 TIA Portal 中看到的描述相同。 （即“CPU-1215 DC/DC/DC”）
`Failsafe`	`FeatureSupport`	根据其订货号，此设备是否为故障安全设备？

属性名称	返回类型	说明
Family	DeviceFamily	此设备属于哪个产品系列？更多信息，请参见 DeviceFamily (页 215) 枚举的描述。
FirmwareUpdateAllowed	FeatureSupport	该设备是否支持固件更新？
FirmwareVersion	string	设备的当前固件版本
HardwareInFirmwareOrder	IHardwareCollection	固件顺序中的硬件集
HardwareInDisplayOrder	IHardwareCollection	显示顺序中的硬件
HardwareNumber	short	编号标识符
ID	uint	工作站中每个设备和模块的唯一标识符。该标识符用作执行 FirmwareUpdate 时的唯一标识符。
Modules	IModuleCollection	连接至工作站的本地模块集合。此处 (页 121)详细描述了该属性。
Name	string	设备名称
NewFirmwareFile	string	新固件文件的文件路径
NewFirmwareVersion	string	此属性用于 SIMATIC Automation Tool 用户界面中。此属性与 API 操作无关。
NewFirmwareNameIsValid	bool	新固件文件是否有效？
Selected	bool	是否已选中设备？
SerialNumber	string	设备的唯一序列号
Slot	uint	硬件项的插槽号
SlotName	string	此属性用于 SIMATIC Automation Tool 用户界面中。此属性与 API 操作无关。
StationNumber	uint	设备的站编号

属性名称	返回类型	说明
SubSlot	uint	设备的子插槽。这与 SB-1200 等可插拔子模块相关。
Supported	FeatureSupport	当前的 SIMATIC Automation Tool API 操作是否支持检测到的网络设备？

使用 IRemoteInterface 设备属性，可以检查分散式网络中的所有站。

以下示例扩展了 分散式 I/O 模块 (页 168) 主题中的示例：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (!scanResult.Succeeded)
    return;

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

    List<IRemoteInterface> decentalNets = devAsCpu.RemoteInterfaces;
    List<string> orderNumbers = new List<string>();

    foreach (IRemoteInterface net in decentalNets)
    {
        //------------------------------------------------------------
        // 检查远程接口
        //------------------------------------------------------------
        if (net.InterfaceType == RemoteInterfaceType.Profinet)
        {
            //------------------------------------------------------
            // 查看每个分散式站
            //------------------------------------------------------
```

```
            List<IBaseDevice> stations = net.Devices;

            foreach (IBaseDevice station in stations)
            {
                orderNumbers.Add(station.ArticleNumber);
            }
        }
    }
}
```

此示例遍历所有远程 PROFINET 接口，并创建一个包含工业网络中的所有分散式站的订货号的列表。

同时 `IBaseDevice` 还支持 `Modules` 属性，可以很容易地进一步扩展示例，从而可以查看分散式工作站以及每个站中的所有本地模块。

3.15 ICPUClassic 接口

3.15.1 识别 IProfinetDeviceCollection 中的经典 CPU 设备

该 `ScanNetworkDevices` 方法 (页 78) 会生成一个 `IProfinetDeviceCollection` (页 85)。该集合包含工业网络上每台可访问设备的一个项。这些设备可以包括 S7-300 和 S7-400（经典）CPU。

`IProfinetDevice 接口` (页 102)提供适用于各类设备的属性和方法。但是，有一个方法是特定于经典 CPU 设备的。可使用 `ICPUClassic` 接口访问此方法 `GetDiagnosticsBuffer` (页 174)。

要确定给定的 IProfinetDevice 接口是否表示一个经典的 CPU 设备，只需将其转换为 ICPUClassic。若转换成功，则表明该网络设备是经典的 CPU，并且可以使用 ICPUClassic 接口的方法。以下示例说明了此转换过程：

```
Network myNetwork = new Network();
IProfinetDeviceCollection scannedDevices;
IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
    foreach (IProfinetDevice dev in scannedDevices)
    {
        ICPUClassic devAsClassicCpu = dev as ICPUClassic;
        if (devAsClassicCpu != null)
        {
            //-----------------------------------------------------
            // 该设备为经典 CPU。
            // ICPUClassic 接口可用于与之交互。
            //-----------------------------------------------------
        }
    }
}
```

说明

ICPUClassic 接口继承自 IProfinetDevice。因此，IProfinetDevice 支持的所有属性和方法也都受 ICPUClassic 支持。

3.15.2 GetDiagnosticsBuffer 方法

返回类型	方法名称
Result	GetDiagnosticsBuffer

参数			
名称	数据类型	参数类型	说明
DiagnosticsItems	List<DiagnosticsItem>	Out	诊断项的集合：集合中的每一项代表诊断缓冲区中的一个条目。
Language	Language	In	请求用于诊断缓冲区条目的语言。

此方法用于读取 CPU 的当前诊断条目。GetDiagnosticsBuffer 方法会返回 DiagnosticsItem (页 67) 对象的集合。以下示例用于搜索特定 IP 地址处经典 CPU 的 IProfinetDeviceCollection。成功搜索到后，将从 CPU 读取诊断信息。Language enum (页 223) 用于获取特定语言中的诊断条目：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
    foreach (IProfinetDevice dev in devices)
    {
        ICPUClassic devAsCpu = dev as ICPUClassic;
        if ((devAsCpu != null) && (devAsCpu.IP == targetIPAddress))
        {
            devAsCpu.SetPassword(new EncryptedString("Password"));
            devAsCpu.Selected = true;
            retVal = devAsCpu.GetDiagnosticsBuffer(out aLogs, Language.English);
            if (retVal.Succeeded)
```

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

`GetDiagnosticsBuffer` 方法会返回 `DiagnosticsItem` (页 67) 对象的集合。

3.16 IHMI 接口

3.16.1 IHMI 接口

通过调用 ScanNetworkDevices 方法生成 IProfinetDeviceCollection。该集合包含工业网络上每台可访问设备的一个项。这些设备可能包括 CPU、HMI、分散式 I/O 站和其它西门子设备。IProfinetDevice 接口提供适用于各类设备的属性和方法。

但是，其中有一些方法仅适用于 HMI 设备。可使用 IHMI 接口访问这些属性和方法。

要确定给定的 IProfinetDevice 接口是否确实表示一个 HMI 设备，只需将其转换为 IHMI。若转换成功，则表明该网络设备是 HMI 设备，并且可以使用 IHMI 接口上的方法。以下示例说明了此过程：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
    foreach (IProfinetDevice dev in scannedDevices)
    {
        IHMI devAsHmi = dev as IHMI;
        if (devAsHmi != null)
        {
            //-----------------------------------------------------
            // 该设备为 HMI。
            // IHMI 接口可用于与之交互。
            //-----------------------------------------------------
        }
    }
}
```

说明

IHMI 接口继承自 IProfinetDevice。因此，IProfinetDevice 接口支持的所有属性和方法也都受 IHMI 支持。以下 IHMI 属性主题仅描述 IHMI 接口特有的属性/方法。

说明

IHMI 接口支持 FirmwareUpdate 方法。但是，此方法始终会返回错误 FirmwareUpdateNotSupported。要更新 HMI 设备的固件，必须执行程序更新方法 (页 180)。

3.16.2 IHMI 属性和标志

3.16.2.1 IHMI 属性

IHMI 接口包含下列属性：

属性名称	返回类型	说明
DeviceType	string	返回对象代表的 HMI 类型
FirmwareDeviceVersion	string	返回 HMI 上存在的固件版本
RuntimeDeviceVersion	string	返回 HMI 上存在的运行系统版本

3.16.2.2 程序更新标志

可以对 IHMI 接口使用这些标志：

属性名称	返回类型	说明
NewProgramNameIsValid	bool	当对有效程序文件夹调用 SetProgramFolder 方法时为真。程序无效时为假。
ProgramUpdateSucceeded	bool	当程序更新成功时为真，即使会返回内部刷新状态错误
NewProgramName	string	新程序的名称
NewProgramFolder	string	新程序的文件夹位置： 该值是通过 SetProgramFolder 方法设置的
NewProgramNameErrorCode	Result	验证新程序时用于发现可能存在的问题（例如程序是否对设备无效或者程序中的 IP 分配是否已经存在于网络上）的代码

3.16.2.3 恢复标志

可以对 IHMI 接口使用这些标志：

属性名称	返回类型	说明
NewRestoreNameIsValid	bool	当使用有效备份文件调用 SetBackupFolder 方法时为真。备份文件无效时为假。
RestoreSucceeded	bool	当恢复成功时为真，即使会返回内部刷新状态错误
NewRestoreName	string	用于确定新程序的名称。
NewRestoreFile	string	用于确定新程序的文件位置。 通过 SetbackupFile 方法设置值。
NewRestoreNameErrorCode	Result	验证新程序时用于发现可能存在的问题（例如程序是否对设备无效或与设备不兼容）的可行方式

3.16.2.4 功能标志

可以对 IHMI 接口使用这些标志：

属性名称	返回类型	说明
BackupAllowed	bool	设备允许备份时为真
ProgramUpdateAllowed	bool	设备允许程序更新时为真
RestoreAllowed	bool	设备允许恢复时为真

3.16.3 IHMI 方法

3.16.3.1 Backup 方法（IHMI 接口）

返回类型	方法名称
Result	Backup

参数			
名称	数据类型	参数类型	说明
strFile	string	In	存储备份文件的完全限定路径和文件名
type	BackupType	In（可选）	若存在，则执行要备份的数据： • 完整备份 • 配方 • 用户管理数据

此方法用于备份 HMI 的数据。某些 HMI 不支持备份/恢复功能。检查 BackupAllowed 属性以确保当前 HMI 支持此功能。以下示例用于搜索特定 IP 地址处 HMI 的 IProfinetDeviceCollection。成功搜索到后，检查 HMI 是否支持备份功能，然后调用 Backup 方法：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
    foreach (IProfinetDevice dev in devices)
    {
    IHMI devAsHmi = dev as IHMI;

        if ((devAsHmi != null) &&
            (devAsHmi.IP == targetIPAddress) &&
```

```
            (devAsHmi.BackupAllowed)
           )
        {
            devAsHmi.Selected = true;
            // back up only recipes
            retVal = devAsHmi.Backup(bkFile, BackupType.Recipes);

            // or do full HMI backup
            Result retVal = devAsHmi.Backup(bkFile);
        }
    }
}
```

3.16.3.2 ProgramUpdate 方法（IHMI 接口）

返回类型	方法名称
Result	ProgramUpdate

此方法用于更新 HMI 设备的操作系统和运行系统软件。参数 strPath 指定包含要加载的程序的文件夹。

要在 IHMI 接口上成功完成 ProgramUpdate 方法，必须验证以下事项：

- 已选择设备
- 已通过 SetProgramFolder 设置 NewProgramFolder

以下示例用于搜索特定 IP 地址处 HMI 的 IProfinetDeviceCollection 以及更新此 HMI 中的程序：

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

Result retVal - new Result();

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out devices);
if (scanResult.Succeeded)
{
    IProfinetDevice dev = devices.FindDeviceByIP(targetIPAddress);
    if (dev != null)
```

```
    {
        IHMI devAsHMI = dev as IHMI;
        if (devAsHMI != null)
        {
            devAsHMI.Selected = true;
            devAsHMI.SetProgramFolder( @"c:\myFolder\ProgramUpdate\Simatic.HMI\RT_Pro
jects\Project1");
            retVal = devAsHMI.ProgramUpdate();
        }
    }
}
```

为成功完成更新，新的程序文件夹必须包含以下文件：

```
DownloadTask.xml
ProjectCharacteristics.rdf
```

这些文件通常位于以下列格式创建的（使用 TIA Portal）文件夹中：

```
{DeviceName)\Simatic.HMI\RT_Projects\{ProjectName}.{DeviceName}
```

例如：

```
"C:\Desktop\hmim14000100a\Simatic.HMI\RT_Projects\DasBasicUndMobilePanelen.hmim140001
00a[KTP700 Mobile]"
```

说明

HMI 操作系统和运行系统软件更新

HMI 设备的 `ProgramUpdate` 与 CPU 不同。此方法可更新 HMI 设备的操作系统和运行系统软件。不能选择部分更新。必要时，SIMATIC Automation Tool 将更新所有数据组分以保持下载一致。HMI 程序更新卡中可包含多个项目，需要在 `\Simatic.HMI\RT_Projects\` 下创建文件夹以进行下载。

3.16.3.3 Restore 方法（IHMI 接口）

返回类型	方法名称
Result	Restore

此方法用于从之前的 HMI 设备备份中恢复数据。某些 HMI 设备不支持备份/恢复功能。检查 RestoreAllowed 属性以确保当前 HMI 设备支持此功能。

要在 IHMI 接口上成功完成 Restore 方法，必须验证以下事项：

- 已选择设备
- 已通过 SetBackupFile 设置 BackupFile

以下示例用于搜索特定 IP 地址处 HMI 的 IProfinetDeviceCollection。成功搜索到后，检查 HMI 是否支持恢复功能，然后调用 Restore 方法：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
    IHMI hmi = scannedDevices.FindDeviceByMAC(targetMACAddress) as IHMI;
    if (hmi != null && hmi.RestoreAllowed == true)
    {
        // 选中要更新的 HMI 设备
        hmi.Selected = true;

        retVal = hmi.SetBackupFile(@"C:\MyFolder\Backup.s7pbkp");
        if (retVal.Failed == true)
            return;

        // IP 地址是否唯一?
        if (hmi.DuplicateIP == true)
            return;

        // 设备是否受支持?
        if (hmi.Supported == false)
            return;
```

```
        // 执行恢复
        retVal = hmi.Restore();
    }
}
```

3.16.3.4 SetProgramFolder 方法

返回类型	方法名称
Result	SetProgramFolder

参数			
名称	数据类型	参数类型	说明
strFolder	string	in	设置存储程序下载源的文件夹位置

以下示例说明了如何在 HMI 设备中设置程序文件夹：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
    IHMI hmi = scannedDevices.FindDeviceByMAC(targetMACAddress) as IHMI;
    if (hmi != null && hmi.ProgramUpdateAllowed == true)
    {
        // 选中要更新的 HMI 设备
        hmi.Selected = true;

        retVal = hmi.SetProgramFolder(@"C:\MyFolder");
        if (retVal.Failed == true)
            return;
```

```
        // IP 地址是否唯一?
        if (hmi.DuplicateIP == true)
            return;

        // 设备是否受支持?
        if (hmi.Supported == false)
            return;

        // 执行程序更新
        retVal = hmi.ProgramUpdate();
    }
}
```

为“strPath”指定的文件夹中必须包含以下文件才能成功完成操作：

- DownloadTask.xml
- ProjectCharacteristics.rdf

这些文件通常位于以下列格式创建的（使用 TIA Portal）文件夹中：

{DeviceName)\Simatic.HMI\RT_Projects\{ProjectName}.{DeviceName}

例如：

"C:\Desktop\hmim14000100a\Simatic.HMI\RT_Projects\DasBasicUndMobilePanelen.hmim14000100a[KTP700 Mobile]"

说明

HMI 设备的 ProgramUpdate 与 CPU 不同。对于 HMI 设备，此方法可用于更新操作系统和运行系统软件。不能选择部分更新。必要时，SIMATIC Automation Tool 将更新组分以保持下载一致。HMI 程序更新卡中可包含多个项目，需要在 \Simatic.HMI\RT_Projects\ 下创建文件夹。

3.16.3.5 SetBackupFile 方法

返回类型	方法名称
Result	SetBackupFile

参数			
名称	数据类型	参数类型	说明
strFile	string	in	设置存储备份文件源的文件夹位置

该方法会在 IHMI 对象上设置以下标志：

- NewRestoreName
- NewRestoreFile
- NewRestoreNameIsValid

以下示例说明了如何在 HMI 设备中设置备份文件路径：

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

IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
    IHMI hmi = scannedDevices.FindDeviceByMAC(targetMACAddress) as IHMI;
    if (hmi != null && hmi.RestoreAllowed == true)
    {
        // 选中要更新的 HMI 设备
        hmi.Selected = true;

        retVal = hmi.SetBackupFile(@"C:\MyFolder\Backup.s7pbkp");
        if (retVal.Failed == true)
            return;

        // IP 地址是否唯一?
        if (hmi.DuplicateIP == true)
            return;
```

```
            // 设备是否受支持?
            if (hmi.Supported == false)
                return;

            // 执行恢复
            retVal = hmi.Restore();
        }
    }
```

3.16.3.6 SetTransferChannel 方法

返回类型	方法名称
Result	SetTransferChannel

参数			
名称	数据类型	参数类型	说明
transferChannel	HMITransferChannel	in	设置用于与 HMI 设备通信的通信类型（传输通道）

HMI 设备支持使用多种协议与 SIMATIC Automation Tool 交换数据。协议即为传输通道。设置 HMI 传输通道的方法如下：

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

```
IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
    IHMI hmi = scannedDevices.FindDeviceByMAC(targetMACAddress) as IHMI;
    if (hmi != null)
   {
      // 将 PN/IE 分配给此 HMI 设备作为通信协议。
      Result retVal = hmi.SetTransferChannel(HMITransferChannel.PN_IE);
   }
}
```

说明

默认传输通道为 `HMITransferChannel.PN_IE`。

3.17 IScalance 接口

3.17.1 IScalance 接口

SCALANCE 设备是 ScanNetworkDevices 方法 (页 78)可搜索到的一种设备类型。ScanNetworkDevices 生成一个 IProfinetDeviceCollection (页 85)，其中包含工业网络上每台可访问设备的一个项。这些设备可能包括 CPU、HMI、SCALANCE 设备和分散式 I/O 站。

IProfinetDevice 接口 (页 102)提供适用于各类设备的属性和方法。对于 SCALANCE 设备，IScalance 接口提供特定于 SCALANCE 设备的属性和方法。

要确定设备接口是否表示 SCALANCE 设备，可将 IProfinetDevice 接口 (页 102)转换为 IScalance 接口。若转换成功，则表明该网络设备是 SCALANCE 设备，并且可以使用 IScalance 方法 (页 189)和属性 (页 189)。由于 IScalance 接口继承自 IProfinetDevice 接口，因此所有 IProfinetDevice 方法和属性都可用于 SCALANCE 设备。

以下示例说明了如何确定扫描到的网络设备是否为 SCALANCE 设备：

```
Network myNetwork = new Network();
IProfinetDeviceCollection scannedDevices;
IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
    foreach (IProfinetDevice dev in scannedDevices)
    {
        IScalance devAsScalance = dev as IScalance;
        if (devAsScalance != null)
        {
            //-----------------------------------------------------
            // 该设备为 SCALANCE 设备。
            // IScalance 接口用于与之交互。
            //-----------------------------------------------------
        }
    }
}
```

3.17.2 IScalance 属性

IScalance 接口具有以下属性：

属性名称	返回类型	说明
ProfileName	string	返回 SNMP 配置文件的名称
ProfileNameisValid	bool	存在有效配置文件名称时为真

3.17.3 IScalance 方法

3.17.3.1 SetProfile 方法

用户界面应用程序调用 SetProfile 方法来建立 SCALANCE 设备的配置文件组态。

返回类型	方法名称
Result	SetProfile

参数			
名称	数据类型	参数类型	说明
Profile	ISNMPProflie	In	用于 SCALANCE 设备固件更新的 SNMP 配置文件和 TFTP（普通文件传输协议）组态

可通过以下示例之一查看 SetProfile 方法调用示例：

- 示例：SNMP 版本 1 组态 (页 207)
- 示例：SNMP 版本 2 组态 (页 209)
- 示例：SNMP 版本 3 组态 (页 211)

在这些示例中，SetProfile 调用都位于示例代码的结尾。

3.17.3.2 FirmwareUpdate 方法

用户界面应用程序调用 `FirmwareUpdate` 方法来更新 SCALANCE 设备的固件。

返回类型	方法名称
`Result`	`FirmwareUpdate`

参数			
名称	数据类型	参数类型	说明
-			

可通过以下示例之一查看 FirmwareUpdate 方法调用示例：

- 示例：SNMP 版本 1 组态 (页 207)
- 示例：SNMP 版本 2 组态 (页 209)
- 示例：SNMP 版本 3 组态 (页 211)

在这些示例中，FirmwareUpdate 调用都位于示例代码的结尾。

3.18 ISNMPProfile 接口

3.18.1 ISNMPProfile 属性

ISNMPProfile 是一个属性合集，其中聚集了使用 SNMP 执行 SCALANCE 设备固件更新时所需的属性。适用于给定设备固件更新的属性取决于该设备上组态的 SNMP 协议版本。属性说明中指出了属性适用的 SNMP 版本（V1、V2、V3）。对于所有 SNMP 版本都适用的属性将在属性说明中省略一个版本。

属性名称	返回类型	说明
ProfileName	string	SNMP 配置文件的名称
Version	SNMPVersion	使用的 SNMP 协议版本
ServerIP	string	IP 地址的字符串表示
ServerIPArray	byte[]	TFTP 服务器的 IP 地址
ServerPort	UInt16	TFTP 端口分配
ReadCommunity	string	V1/V2 读取权限 ID 字符串 (PW)
WriteCommunity	string	V1/V2 写入权限 ID 字符串 (PW)
UserName	string	V3 用户名
ContextName	string	V3 管理信息上下文 ID
SecurityLevel	SNMPSecurityLevel	V3 安全等级
AuthAlgorithm	SNMPAuthAlgorithm	V3 身份认证算法
AuthKey	byte[]	V3 身份认证密钥
PrivAlgorithm	SNMPPrivAlgorithm	V3 隐私算法
PrivKey	byte[]	V3 隐私密钥

3.18.2 ISNMPProfile 方法

3.18.2.1 Validate 方法

Validate 方法用于评估 SNMP 配置文件参数，以确保所需固件更新参数的设置保持一致。返回的结果将显示所有不一致的设置。

返回类型	方法名称
`Result`	`Validate`

参数			
名称	数据类型	参数类型	说明
-			

可通过以下示例之一查看 Validate 方法调用示例：

- 示例：SNMP 版本 1 组态 (页 207)
- 示例：SNMP 版本 2 组态 (页 209)
- 示例：SNMP 版本 3 组态 (页 211)

在这些示例中，Validate 调用在设置配置文件参数后进行。

3.18.3 SNMPProfile 类

3.18.3.1 SNMPProfile 类属性

SNMPProfile 类用于存储和编辑 SNMP 配置文件。

属性名称	返回类型	说明
ProfileName	string	SNMP 配置文件的名称
Version	SNMPVersion	使用的 SNMP 协议版本
ServerIP	string	IP 地址的字符串表示
ServerIPArray	byte[]	TFTP 服务器的 IP 地址
ServerPort	UInt16	TFTP 端口分配
ReadCommunity	string	V1/V2 读取权限 ID 字符串
WriteCommunity	string	V1/V2 写入权限 ID 字符串
UserName	string	V3 用户名
ContextName	string	V3 管理信息上下文 ID
SecurityLevel	SNMPSecurityLevel	V3 安全等级
AuthAlgorithm	SNMPAuthAlgorithm	V3 身份认证算法
AuthKey	byte[]	V3 身份认证密钥
PrivAlgorithm	SNMPPrivAlgorithm	V3 隐私算法
PrivKey	byte[]	V3 隐私密钥

3.18.3.2 SNMPProfile 类方法

SetProfileName

SetProfileName 方法用于将输入字符串值分配至配置文件名称。

返回类型	方法名称
`Result`	`SetProfileName`

参数			
名称	**数据类型**	**参数类型**	**说明**
`strName`	`string`	in	配置文件名称

可通过以下示例之一查看 SetProfileName 方法调用示例：

- 示例：SNMP 版本 1 组态 (页 207)
- 示例：SNMP 版本 2 组态 (页 209)
- 示例：SNMP 版本 3 组态 (页 211)

在这些示例中，SetProfileName 调用都位于示例代码的开头。

SetSNMPVersion

SetSNMPVersion 方法用于设置 SNMP 协议版本。SNMP 协议的版本为 1、2 或 3。

返回类型	方法名称
`Result`	`SetSNMPVersion`

参数			
名称	**数据类型**	**参数类型**	**说明**
`nVersion`	`SNMPVersion` (页 226)	in	SNMP 版本 1、2、3

可通过以下示例之一查看 SetSNMPVersion 方法调用示例：

- 示例：SNMP 版本 1 组态 (页 207)
- 示例：SNMP 版本 2 组态 (页 209)
- 示例：SNMP 版本 3 组态 (页 211)

在这些示例中，SetSNMPVersion 调用都位于示例代码的开头。

SetServerIP 方法

SetServerIP 方法用于设置 TFTP 服务器的 IP 地址。此方法适用于所有 SNMP 配置文件。

返回类型	方法名称
`Result`	`SetServerIP`

参数			
名称	**数据类型**	**参数类型**	**说明**
`strIP`	`string`	in	IP 分配

可通过以下示例之一查看 SetServerIP 方法调用示例：

- 示例：SNMP 版本 1 组态 (页 207)
- 示例：SNMP 版本 2 组态 (页 209)
- 示例：SNMP 版本 3 组态 (页 211)

在这些示例中，SetServerIP 调用都位于示例代码的开头。

SetServerPort 方法

SetServerPort 方法用于设置 TFTP 服务器的端口分配。此方法适用于所有 SNMP 配置文件。

返回类型	方法名称
`Result`	`SetServerPort`

参数			
名称	**数据类型**	**参数类型**	**说明**
`nVersion`	`UInt16`	in	TFTP 端口分配

可通过以下示例之一查看 SetServerPort 方法调用示例：

- 示例：SNMP 版本 1 组态 (页 207)
- 示例：SNMP 版本 2 组态 (页 209)
- 示例：SNMP 版本 3 组态 (页 211)

在这些示例中，SetServerPort 调用在 SetServerIP 调用后立即进行。

SetReadCommunity 方法

SetProfileName 方法用于为输入字符串分配读取团体。读取团体字符串使远程设备能够检索设备中的只读信息。此方法适用于 SNMP 版本 1 或版本 2 配置文件。

返回类型	方法名称
`Result`	`SetReadCommunity`

参数			
名称	**数据类型**	**参数类型**	**说明**
`strCommunity`	`string`	in	读取团体分配

可通过以下示例之一查看 SetReadCommunity 方法调用示例：

- 示例：SNMP 版本 1 组态 (页 207)
- 示例：SNMP 版本 2 组态 (页 209)

在这些示例中，SetReadCommunity 调用都位于示例代码的中间位置。

SetWriteCommunity 方法

SetWriteCommunity 方法用于为输入字符串分配写入团体。写入团体字符串允许对设备执行读取操作和写入操作。此方法适用于 SNMP 版本 1 或版本 2 配置文件。

返回类型	方法名称
`Result`	`SetWriteCommunity`

参数			
名称	**数据类型**	**参数类型**	**说明**
`strCommunity`	`string`	in	写入团体分配

可通过以下示例之一查看 SetWriteCommunity 方法调用示例：

- 示例：SNMP 版本 1 组态 (页 207)
- 示例：SNMP 版本 2 组态 (页 209)

在这些示例中，SetWriteCommunity 调用都位于示例代码的开头。

SetUserName 方法

SetUserName 方法用于设置 SNMP 版本 3 配置文件的用户名。

返回类型	方法名称
`Result`	`SetUserName`

参数			
名称	**数据类型**	**参数类型**	**说明**
`strName`	`string`	in	SNMP V3 用户名

可以在 示例：SNMP 版本 3 组态 (页 211) 代码示例的中间位置查看 SetUserName 方法调用示例。

SetContextName 方法

SetContextName 方法用于设置 SNMP 版本 3 配置文件的上下文名称。

返回类型	方法名称
`Result`	`SetContextName`

参数			
名称	数据类型	参数类型	说明
`strName`	`string`	in	SNMP V3 管理信息上下文

可以在 示例：SNMP 版本 3 组态 (页 211) 代码示例的中间位置查看 SetContextName 方法调用示例。

SetSecurityLevel 方法

SetSecurityLevel 方法用于设置 SNMP 版本 3 配置文件的安全等级。

返回类型	方法名称
`Result`	`SetSecurityLevel`

参数			
名称	数据类型	参数类型	说明
level	`SNMPSecurityLevel` (页 226)	in	SNMP V3 安全等级

可以在 示例：SNMP 版本 3 组态 (页 211) 代码示例的中间位置查看 SetSecurityLevel 方法调用示例。

SetAuthAlgorithm 方法

SetAuthAlgorithm 方法用于设置 SNMP 版本 3 配置文件的身份验证算法。当安全等级 (页 198)要求“身份认证”时，身份认证算法适用。当安全等级要求“隐私”及“身份认证”时，身份认证密码适用。

返回类型	方法名称
`Result`	`SetAuthAlgorithm`

参数			
名称	**数据类型**	**参数类型**	**说明**
algorithm	`SNMPAuthAlgorithm` (页 227)	in	SNMP V3 身份认证算法
strPassword	`string`	in	SNMP V3 身份认证密码

可以在 示例：SNMP 版本 3 组态 (页 211) 代码示例的中间位置查看 SetAuthAlgorithm 方法调用示例。

SetPrivAlgorithm 方法

SetAuthAlgorithm 方法用于设置 SNMP 版本 3 配置文件的隐私算法。当安全等级 (页 198)要求“隐私”时，隐私算法适用。

返回类型	方法名称
`Result`	`SetPrivAlgorithm`

参数			
名称	**数据类型**	**参数类型**	**说明**
algorithm	`SNMPPrivAlgorithm` (页 227)	in	SNMP V3 隐私算法
strPassword	`string`	in	SNMP V3 隐私密码

可以在 示例：SNMP 版本 3 组态 (页 211) 代码示例的中间位置查看 SetPrivAlgorithm 方法调用示例。

3.19 SNMPProfile 类

3.19.1 SNMPProfile 类属性

SNMPProfile 类用于存储和编辑 SNMP 配置文件。

属性名称	返回类型	说明
ProfileName	string	SNMP 配置文件的名称
Version	SNMPVersion	使用的 SNMP 协议版本
ServerIP	string	IP 地址的字符串表示
ServerIPArray	byte[]	TFTP 服务器的 IP 地址
ServerPort	UInt16	TFTP 端口分配
ReadCommunity	string	V1/V2 读取权限 ID 字符串
WriteCommunity	string	V1/V2 写入权限 ID 字符串
UserName	string	V3 用户名
ContextName	string	V3 管理信息上下文 ID
SecurityLevel	SNMPSecurityLevel	V3 安全等级
AuthAlgorithm	SNMPAuthAlgorithm	V3 身份认证算法
AuthKey	byte[]	V3 身份认证密钥
PrivAlgorithm	SNMPPrivAlgorithm	V3 隐私算法
PrivKey	byte[]	V3 隐私密钥

3.19.2 SNMPProfile 类方法

3.19.2.1 SetProfileName

SetProfileName 方法用于将输入字符串值分配至配置文件名称。

返回类型	方法名称
Result	SetProfileName

参数			
名称	数据类型	参数类型	说明
strName	string	in	配置文件名称

可通过以下示例之一查看 SetProfileName 方法调用示例：

- 示例：SNMP 版本 1 组态 (页 207)
- 示例：SNMP 版本 2 组态 (页 209)
- 示例：SNMP 版本 3 组态 (页 211)

在这些示例中，SetProfileName 调用都位于示例代码的开头。

3.19.2.2 SetSNMPVersion

SetSNMPVersion 方法用于设置 SNMP 协议版本。SNMP 协议的版本为 1、2 或 3。

返回类型	方法名称
Result	SetSNMPVersion

参数			
名称	数据类型	参数类型	说明
nVersion	SNMPVersion (页 226)	in	SNMP 版本 1、2、3

可通过以下示例之一查看 SetSNMPVersion 方法调用示例：

- 示例：SNMP 版本 1 组态 (页 207)
- 示例：SNMP 版本 2 组态 (页 209)
- 示例：SNMP 版本 3 组态 (页 211)

在这些示例中，SetSNMPVersion 调用都位于示例代码的开头。

3.19.2.3 SetServerIP 方法

SetServerIP 方法用于设置 TFTP 服务器的 IP 地址。此方法适用于所有 SNMP 配置文件。

返回类型	方法名称
Result	SetServerIP

参数			
名称	数据类型	参数类型	说明
strIP	string	in	IP 分配

可通过以下示例之一查看 SetServerIP 方法调用示例：

- 示例：SNMP 版本 1 组态 (页 207)
- 示例：SNMP 版本 2 组态 (页 209)
- 示例：SNMP 版本 3 组态 (页 211)

在这些示例中，SetServerIP 调用都位于示例代码的开头。

3.19.2.4 SetServerPort 方法

SetServerPort 方法用于设置 TFTP 服务器的端口分配。此方法适用于所有 SNMP 配置文件。

返回类型	方法名称
Result	SetServerPort

参数			
名称	数据类型	参数类型	说明
nVersion	UInt16	in	TFTP 端口分配

可通过以下示例之一查看 SetServerPort 方法调用示例：

- 示例：SNMP 版本 1 组态 (页 207)
- 示例：SNMP 版本 2 组态 (页 209)
- 示例：SNMP 版本 3 组态 (页 211)

在这些示例中，SetServerPort 调用在 SetServerIP 调用后立即进行。

3.19.2.5 SetReadCommunity 方法

SetProfileName 方法用于为输入字符串分配读取团体。读取团体字符串使远程设备能够检索设备中的只读信息。此方法适用于 SNMP 版本 1 或版本 2 配置文件。

返回类型	方法名称
`Result`	`SetReadCommunity`

参数			
名称	**数据类型**	**参数类型**	**说明**
`strCommunity`	`string`	in	读取团体分配

可通过以下示例之一查看 SetReadCommunity 方法调用示例：

- 示例：SNMP 版本 1 组态 (页 207)
- 示例：SNMP 版本 2 组态 (页 209)

在这些示例中，SetReadCommunity 调用都位于示例代码的中间位置。

3.19.2.6 SetWriteCommunity 方法

SetWriteCommunity 方法用于为输入字符串分配写入团体。写入团体字符串允许对设备执行读取操作和写入操作。此方法适用于 SNMP 版本 1 或版本 2 配置文件。

返回类型	方法名称
Result	SetWriteCommunity

参数			
名称	数据类型	参数类型	说明
strCommunity	string	in	写入团体分配

可通过以下示例之一查看 SetWriteCommunity 方法调用示例：

- 示例：SNMP 版本 1 组态 (页 207)
- 示例：SNMP 版本 2 组态 (页 209)

在这些示例中，SetWriteCommunity 调用都位于示例代码的开头。

3.19.2.7 SetUserName 方法

SetUserName 方法用于设置 SNMP 版本 3 配置文件的用户名。

返回类型	方法名称
Result	SetUserName

参数			
名称	数据类型	参数类型	说明
strName	string	in	SNMP V3 用户名

可以在 示例：SNMP 版本 3 组态 (页 211) 代码示例的中间位置查看 SetUserName 方法调用示例。

3.19.2.8 SetContextName 方法

SetContextName 方法用于设置 SNMP 版本 3 配置文件的上下文名称。

返回类型	方法名称
Result	SetContextName

参数			
名称	数据类型	参数类型	说明
strName	string	in	SNMP V3 管理信息上下文

可以在 示例：SNMP 版本 3 组态 (页 211) 代码示例的中间位置查看 SetContextName 方法调用示例。

3.19.2.9 SetSecurityLevel 方法

SetSecurityLevel 方法用于设置 SNMP 版本 3 配置文件的安全等级。

返回类型	方法名称
Result	SetSecurityLevel

参数			
名称	数据类型	参数类型	说明
level	SNMPSecurityLevel (页 226)	in	SNMP V3 安全等级

可以在 示例：SNMP 版本 3 组态 (页 211) 代码示例的中间位置查看 SetSecurityLevel 方法调用示例。

3.19.2.10 SetAuthAlgorithm 方法

SetAuthAlgorithm 方法用于设置 SNMP 版本 3 配置文件的身份验证算法。当安全等级 (页 198)要求“身份认证”时，身份认证算法适用。当安全等级要求“隐私”及“身份认证”时，身份认证密码适用。

返回类型	方法名称
`Result`	`SetAuthAlgorithm`

参数			
名称	数据类型	参数类型	说明
algorithm	`SNMPAuthAlgorithm` (页 227)	in	SNMP V3 身份认证算法
strPassword	`string`	in	SNMP V3 身份认证密码

可以在 示例：SNMP 版本 3 组态 (页 211) 代码示例的中间位置查看 SetAuthAlgorithm 方法调用示例。

3.19.2.11 SetPrivAlgorithm 方法

SetAuthAlgorithm 方法用于设置 SNMP 版本 3 配置文件的隐私算法。当安全等级 (页 198)要求“隐私”时，隐私算法适用。

返回类型	方法名称
`Result`	`SetPrivAlgorithm`

参数			
名称	数据类型	参数类型	说明
algorithm	`SNMPPrivAlgorithm` (页 227)	in	SNMP V3 隐私算法
strPassword	`string`	in	SNMP V3 隐私密码

可以在 示例：SNMP 版本 3 组态 (页 211) 代码示例的中间位置查看 SetPrivAlgorithm 方法调用示例。

3.20 IScalance 和 ISNMP 固件更新代码示例

3.20.1 示例：SNMP 版本 1 组态

此示例显示如何组态和启动 SCALANCE 设备的固件更新。在此示例中，SCALANCE 设备使用 SNMP 版本 1：

```
SNMPProfile profileV1 = new SNMPProfile();

//使用 SNMP 版本 1 创建配置文件
Result retVal = profileV1.SetSNMPVersion(SNMPVersion.Version1);
if (retVal.Failed)
      return;

// 设置此配置文件的名称
retVal = profileV1.SetProfileName("Profile_1");
if (retVal.Failed)
      return;

//设置 TFTP 服务器 IP 地址，必需
retVal = profileV1.SetServerIP("192.168.0.1");
if (retVal.Failed)
      return;

//设置 TFTP 服务器端口，默认为 69，可选
retVal = profileV1.SetServerPort(69);
if (retVal.Failed)
      return;

//设置 SCALANCE 读取团体，默认为"公共"，可选
retVal = profileV1.SetReadCommunity("public");
if (retVal.Failed)
      return;

//设置 SCALANCE 写入团体，默认为"私有"，可选
retVal = profileV1.SetWriteCommunity("private");
if (retVal.Failed)
      return;
```

```
//验证配置文件是否有效，这样可确保不会丢失参数
retVal = profileV1.Validate();
if (retVal.Failed)
      return;

//扫描并执行所有 SCALANCE 设备的固件更新
Network myNetwork = new Network();
IProfinetDeviceCollection scannedDevices;

//扫描网络，搜索已附加的设备
IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
      //过滤 scalance 设备
      List<IProfinetDevice> onlyScalance =
       scannedDevices.FilterByDeviceFamily(new List<DeviceFamily>(1)
{ DeviceFamily.SCALANCE });
      foreach (IScalance scalance in onlyScalance)
      {
         //设置与此 SCALANCE 设备进行通信所需的 SNMP 配置文件
         retVal = scalance.SetProfile(profileV1);
         if (retVal.Succeeded)
         {
         //设置新固件文件以更新 SCALANCE
         retVal = scalance.SetFirmwareFile(@"C:\Firmware\FirmwareFile.lad");
         if (retVal.Succeeded)
               retVal = scalance.FirmwareUpdate();
         }
      }
}
```

3.20.2 示例：SNMP 版本 2 组态

此示例显示如何组态和启动 SCALANCE 设备的固件更新。在此示例中，SCALANCE 设备使用 SNMP 版本 2：

```
SNMPProfile profileV2 = new SNMPProfile();

// 使用 SNMP 版本 2 创建配置文件
Result retVal = profileV2.SetSNMPVersion(SNMPVersion.Version2);
if (retVal.Failed)
      return;

// 设置此配置文件的名称
retVal = profileV2.SetProfileName("Profile_2");
if (retVal.Failed)
      return;

// 设置 TFTP 服务器 IP 地址，必需
retVal = profileV2.SetServerIP("192.168.0.1");
if (retVal.Failed)
      return;

// 设置 TFTP 服务器端口，默认为 69，可选
retVal = profileV2.SetServerPort(69);
if (retVal.Failed)
      return;

// 设置 SCALANCE 读取团体，默认为"公共"，可选
retVal = profileV2.SetReadCommunity("public");
if (retVal.Failed)
      return;

// 设置 SCALANCE 写入团体，默认为"私有"，可选
retVal = profileV2.SetWriteCommunity("private");
if (retVal.Failed)
      return;

// 验证配置文件是否有效，这样可确保不会丢失参数
retVal = profileV2.Validate();
if (retVal.Failed)
```

```
        return;

// 扫描并执行所有 SCALANCE 设备的固件更新
Network myNetwork = new Network();
IProfinetDeviceCollection scannedDevices;

// 扫描网络，搜索已附加的设备
IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
{
        // 过滤 scalance 设备
        List<IProfinetDevice> onlyScalance =
         scannedDevices.FilterByDeviceFamily(new List<DeviceFamily>(1)
{ DeviceFamily.SCALANCE });
        foreach (IScalance scalance in onlyScalance)
        {
                // 设置与此 SCALANCE 设备进行通信所需的 SNMP 配置文件
                retVal = scalance.SetProfile(profileV2);
                if (retVal.Succeeded)
                {
                        // 设置新固件文件以更新 SCALANCE
                        retVal = scalance.SetFirmwareFile(@"C:\Firmware\FirmwareFile.lad");
                        if (retVal.Succeeded)
                                retVal = scalance.FirmwareUpdate();
                }
        }
}
```

3.20.3 示例：SNMP 版本 3 组态

此示例显示如何组态和启动 SCALANCE 设备的固件更新。在此示例中，SCALANCE 设备使用 SNMP 版本 3 实现身份认证和隐私：

```
SNMPProfile profileV3_3 = new SNMPProfile();

// 使用 SNMP 版本 3 创建配置文件
Result retVal = profileV3_3.SetSNMPVersion(SNMPVersion.Version3);
if (retVal.Failed)
      return;

// 设置此配置文件的名称
retVal = profileV3_3.SetProfileName("Profile_3.1");
if (retVal.Failed)
      return;

// 设置 TFTP 服务器 IP 地址，必需
retVal = profileV3_3.SetServerIP("192.168.0.1");
if (retVal.Failed)
      return;

// 设置 TFTP 服务器端口，默认为 69，可选
retVal = profileV3_3.SetServerPort(69);
if (retVal.Failed)
      return;

// 设置用户名，必需
retVal = profileV3_3.SetUserName("User1");
if (retVal.Failed)
      return;

// 设置上下文名称，可选
retVal = profileV3_3.SetContextName("Context1");
if (retVal.Failed)
      return;

// 设置 SCALANCE 安全等级 NoAuthNoPriv，必需
retVal = profileV3_3.SetSecurityLevel(SNMPSecurityLevel.AuthPriv);
```

```
//将身份认证算法设置为 MD5 并设置密码，必需
retVal  =  profileV3_3.SetAuthAlgorithm(SNMPAuthAlgorithm.MD5,  "Password1");
if  (retVal.Failed)
    return;

//将隐私算法设置为 DES 并设置密码，必需
retVal  =  profileV3_3.SetPrivAlgorithm(SNMPPrivAlgorithm.DES,  "Password2");
if  (retVal.Failed)
       return;

if (retVal.Failed)
      return;

// 验证配置文件是否有效，这样可确保不会丢失参数
retVal = profileV3_3.Validate();
if (retVal.Failed)
      return;

// 扫描并执行所有 SCALANCE 设备的固件更新
Network myNetwork = new Network();
IProfinetDeviceCollection scannedDevices;

// 扫描网络，搜索已附加的设备
IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out scannedDevices);
if (scanResult.Succeeded)
      {
            // 过滤 SCALANCE 设备
            List<IProfinetDevice> onlyScalance =
             scannedDevices.FilterByDeviceFamily(new List<DeviceFamily>(1)
{ DeviceFamily.SCALANCE });
            foreach (IScalance scalance in onlyScalance)
            {
                  // 设置与 SCALANCE 设备进行通信所需的 SNMP 配置文件
                  retVal = scalance.SetProfile(profileV3_3);
                  if (retVal.Succeeded)
                  {
                        // 设置新固件文件以更新 SCALANCE
                        retVal =
scalance.SetFirmwareFile(@"C:\Firmware\FirmwareFile.lad");
```

```
                    if (retVal.Succeeded)
                        retVal = scalance.FirmwareUpdate();
                }
            }
        }
```

3.21 异常

3.21.1 CriticalInternalErrorException

当检测到严重错误情况时，API 接口将出现异常。

当应用程序检测到 CriticalInternalErrorException 异常已触发时，关闭正在使用 API 的应用程序。发生了严重错误。

```
Network myNetwork = new Network();
try
{
    uint targetIPAddress = 0xC0A80001; // 192.168.0.1
    IProfinetDeviceCollection devices;

    IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out devices);
    if (scanResult.Succeeded)

    {
        IProfinetDevice dev = devices.FindDeviceByIP(targetIPAddress);
        if (dev != null)
        {
            ICPU devAsCpu = dev as ICPU;
            if (devAsCpu != null)
            {
                devAsCpu.SetPassword(new EncryptedString("Password"));
                devAsCpu.Selected = true;
                if (devAsCpu.Failsafe)devAsCpu.SelectedConfirmed = true;
                Result retVal = devAsCpu.ResetToFactoryDefaults();
            }
```

```
            }
        }
    }
    catch (CriticalInternalErrorException e)
    {
        // API 中发生了严重的内部错误
    }
    catch (Exception e)
    {
        // API 中发生了异常

    }
```

3.22 API 枚举

3.22.1 DataChangedType

此枚举定义了 DataChangedEventHandler (页 118) 的可能参数值：

```
Invalid
OperatingState
RackInformation
Folders
File
ProfinetName
IPAddress
Password
```

3.22.2 DeviceFamily

此枚举指定硬件项的产品系列：

```
None
CPU1200
CPU1500
CPU300
CPU400
ET200SP
ET200MP
ET200AL
ET200PRO
ET200ECO
ET200S
ET200M
HMI
SITOPUPS
SCALANCE
SIMOCODE
Unsupported
SIRIUS_ACT
Gateway
NetworkDevice
```

```
MicroDrive
SoftStarter
CommunicationsModule
OpticalReader
```

3.22.3 ConfirmationType

此枚举指定了安全相关操作的用户确认类型：

```
Invalid = 0
SafetyPasswordIsBeingUsed = 0x2f161717
DeletingExistingSafetyProgram = 0x40232122
ReplacingExistingSafetyProgram = 0x492a282b
ReplacingExistingSafetyProgramWithNonSafetyProgram = 0x4a2c2b2d
LoadingSafetyProgram = 0x46292728,
```

3.22.4 ErrorCode

此枚举列出了 Result 对象所有可能的返回值：

```
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
OperationCancelledByUser
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

//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
```

3.22.5 Language

Language 枚举允许用户为返回的字符串数据指定语言。其中包含以下值：

```
English
German
French
Spanish
Italian
Chinese
```

3.22.6 OperatingState

此枚举定义了 OperatingState 属性的可能状态：

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

3.22.7 OperatingStateREQ

此枚举定义了在调用 SetOperatingState (页 156) 方法时可请求的可能状态转换：

```
Stop
Run
```

3.22.8 ProgressAction

此枚举定义了可被发送至 ProgressChangedEventHandler (页 119) 的可能参数值：

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

3.22.9 RemoteInterfaceType

此枚举定义了在 IRemoteInterfaces (页 168) 接口上调用 `InterfaceType` 属性时可以返回的可能状态：

```
None
Profinet
Profibus
ASi
```

3.22.10 ProtectionLevel

`ProtectionLevel` 枚举列出了 CPU 密码的保护等级：

`Unknown`

`Failsafe`

`Full`

`Read`

`HMI`

`NoAccess`

`NoPassword`

3.22.11 ConfirmationType

此枚举用于指示故障安全 CPU 的状态：

说明	值
`Invalid`	0
`SafetyPasswordIsBeingUsed`	0x2f161717
`DeletingExistingSafetyProgram`	0x40232122
`ReplacingExistingSafetyProgram`	0x492a282b
`ReplacingExistingSafetyProgramWithNonSafetyProgram`	0x4a2c2b2d
`LoadingSafetyProgram`	0x46292728

3.22.12 FailsafeOperation

`FailSafeOperation` 枚举表示与安全相关的操作：

说明	值
`Invalid`	0
`ResetToFactoryOperation`	0x2f161717
`FormatMCOperation`	0x46292728
`ProgramUpdateOperation`	0x43252224
`RestoreOperation`	0x45262427

3.22.13 RemoteFolderType

`RemoteFolderType` 枚举表示远程文件夹类型：

说明	值
`None`	0
`Recipe`	1
`Datalog`	2

3.22.14 SNMPVersion

`SNMPVersion` 枚举表示 SCALANCE 设备的 SNMP 版本号：

说明	值
`NotSupported`	-1
`Version1`	0
`Version2`	1
`Version3`	2

3.22.15 SNMPSecurityLevel

`SNMPSecurityLevel` 枚举表示 SNMP 版本 3 配置文件的安全等级。使用 SNMP 版本 3 配置文件的 SCALANCE 设备具有安全等级设置：

说明	值
`NotSupported`	-1
`NoAuthNoPriv`	0
`AuthNoPriv`	1
`AuthPriv`	2

3.22.16 SNMPAuthAlgorithm

`SNMPAuthAlgorithm` 枚举表示使用 SNMP 版本 3 配置文件的 SCALANCE 设备的授权算法：

说明	值
`NotSupported`	-1
`MD5`	0
`SHA`	1

3.22.17 SNMPPrivAlgorithm

`SNMPPrivAlgorithm` 枚举表示使用 SNMP 版本 3 配置文件的 SCALANCE 设备的隐私算法：

说明	值
`NotSupported`	-1
`DES`	0
`AES`	1

3.22.18 ScanErrorType

`ScanErrorType` 枚举指示设备扫描返回的错误类型：

说明	值
`Invalid`	-1
`Success`	0
`Error`	1
`Warning`	2
`Information`	3

3.22.19 HMITransferChannel

`HMITransferChannel` 枚举指示用于与 HMI 设备通信的协议：

说明	值
`Invalid`	-1
`PN_IE`	0
`Ethernet`	1

3.22.20 BackupType

`BackupType` 枚举指示用于与 HMI 设备通信的协议：

说明	值
`Invalid`	0
`FullBackup`	1
`Recipes`	2
`UserAdministration`	3

3.22.21 TimeFormat

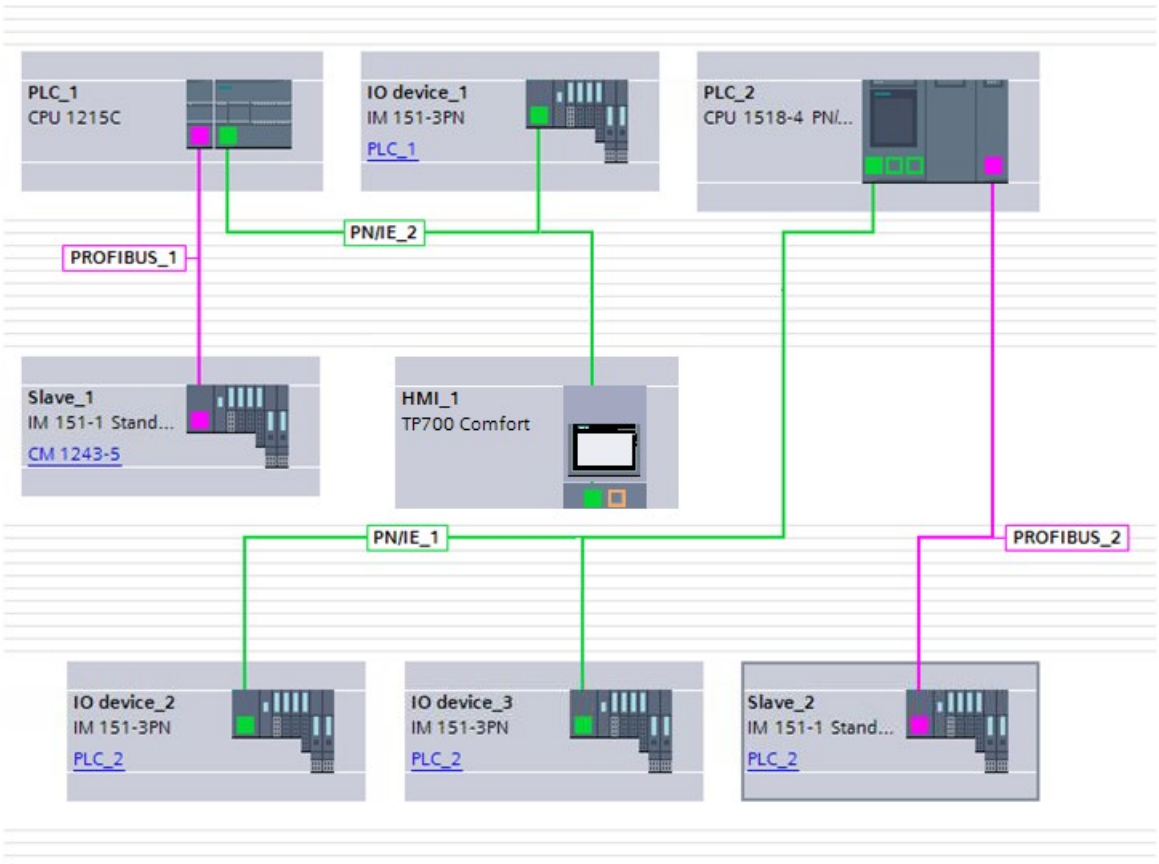
`TimeFormat` 用于指定时间字符串的时间格式：

说明	值
`Local`	0
`UTC`	1

UTC 表示协调世界时。

3.23 网络示例

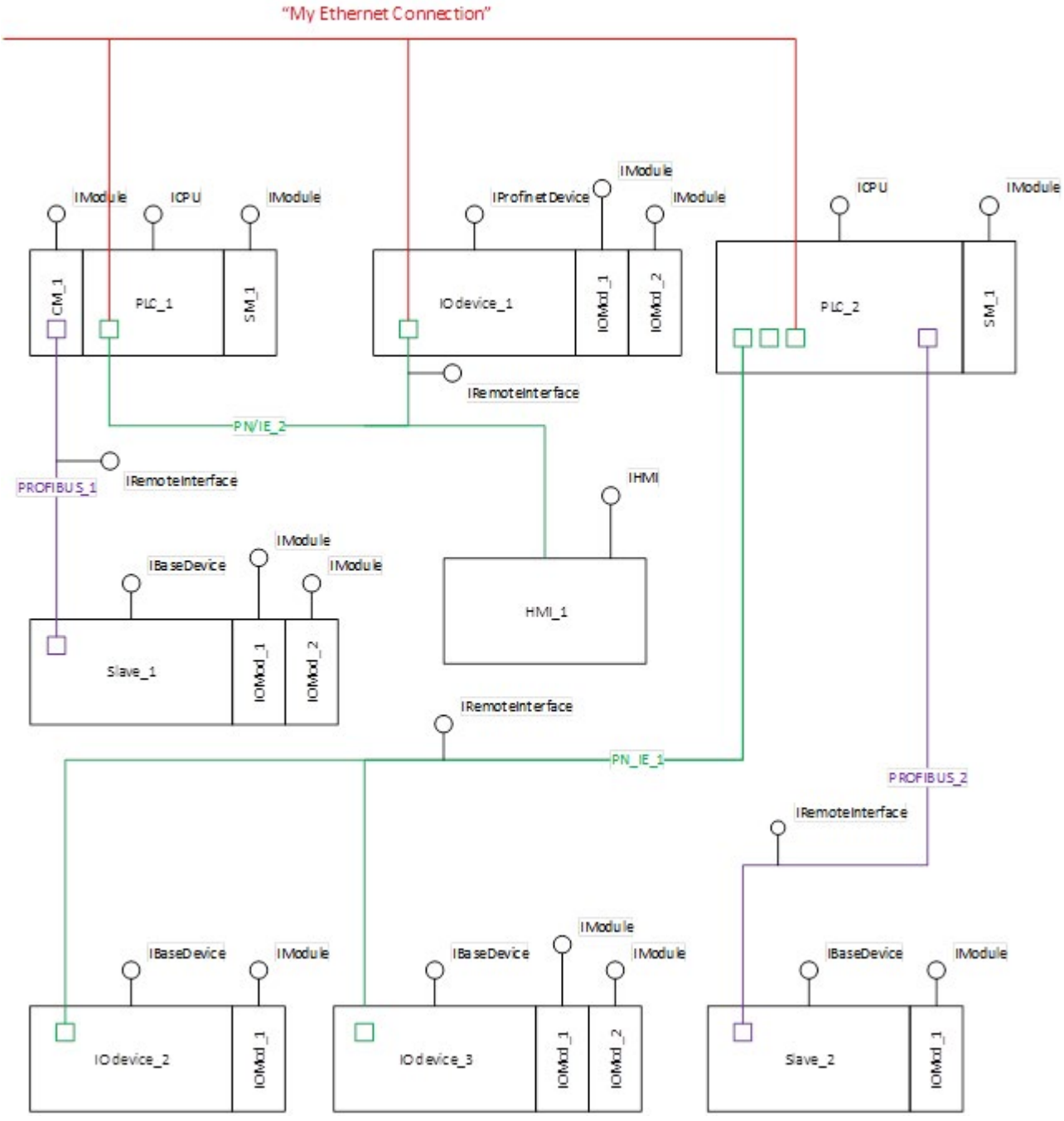
此示例显示了 TIA Portal 网络组态和表示已联网设备的 API 接口：



假设第一行中的所有设备（PLC_1、IO device_1 和 PLC_2）已连接至外部以太网（未显示）。这些设备可通过 SIMATIC Automation Tool API 直接访问。此外，假设连接到 PLC_2 的 PROFINET 子网未连接至外部网络。

SIMATIC Automation Tool API 可以为此组态中的所有 PLC 和 I/O 站提供信息和操作。

下图显示了相同的网络组态和网络中的硬件设备：



在上图中，“棒棒糖”符号显示最能代表每个网络组件的 SIMATIC Automation Tool API 接口类别：

- 直接与外部网络连接的 CPU 由 `ICPU` 接口表示。
- 直接与外部网络连接的 I/O 站由 `IProfinetDevice` 接口表示。
- 源自 CPU 的子网由 `IRemoteInterface` 接口表示。
- 未直接连接至外部网络（但可通过 CPU 访问）的 I/O 站由 `IBaseDevice` 接口表示。
- 连接至 CPU 或 IO 站的 I/O 或通信模块由 `IModule` 接口表示。

索引

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

F

S H

H

Z

W

C H

X

G