



SIMATIC

SIMATIC Automation Tool SDK Windows 用户指南

编程手册

前言

1

了解 SIMATIC Automation
Tool SDK 示例程序

2

更新使用旧版 SDK 制作的自定
义应用程序

3

用于 .NET 框架的 SIMATIC
Automation Tool API

4

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。

目录

1	前言.....	9
1.1	安全性信息.....	9
1.2	SIMATIC Automation Tool SDK 概述.....	9
1.3	SIMATIC Automation Tool SDK 试用版的功能.....	9
1.4	SDK 设备目录.....	10
2	了解 SIMATIC Automation Tool SDK 示例程序.....	11
2.1	示例应用程序简介.....	11
2.2	创建使用 API 的项目.....	11
2.3	为示例程序设置代码.....	12
2.4	选择示例程序的语言.....	18
2.5	选择网络接口.....	20
2.6	通过 IP 地址连接到网络上的 CPU.....	21
2.7	显示 CPU 和模块信息.....	25
2.8	切换 CPU 的操作状态.....	27
2.9	设置 CPU 的 IP 地址、子网和网关.....	28
2.10	设置 CPU 的 PROFINET 名称.....	29
2.11	更新 CPU 的固件.....	30
2.12	将应用程序分发给您的最终用户.....	31
3	更新使用旧版 SDK 制作的自定义应用程序.....	33
4	用于 .NET 框架的 SIMATIC Automation Tool API.....	34
4.1	API 入门指南.....	34
4.1.1	架构概述.....	39
4.1.2	在用户界面应用程序中引用 API.....	40
4.2	API 软件和版本兼容性.....	41
4.3	S7 通信的要求.....	41
4.4	为故障安全设备和安全相关操作设计用户界面应用程序.....	42
4.4.1	安全相关操作和故障安全设备的 API 支持.....	42
4.4.2	安全相关操作的用户界面编程指南.....	43
4.4.3	用户界面中的颜色编码安全域.....	44
4.4.3.1	对 CPU 设备图标进行颜色编码.....	45
4.4.3.2	对设备数据进行颜色编码.....	46
4.4.3.3	对 CPU 密码进行颜色编码.....	47
4.4.3.4	对程序文件夹进行颜色编码.....	48
4.4.3.5	对程序密码进行颜色编码.....	49
4.4.4	汉明码.....	49

4.5	公共支持类别.....	50
4.5.1	EncryptedString 类.....	50
4.5.2	Result 类.....	51
4.5.3	DiagnosticsItem 类.....	53
4.5.4	DataChangedEventArgs 类.....	53
4.5.5	ProgressChangedEventArgs 类.....	54
4.5.6	ExportProgressEventArgs 类.....	54
4.5.7	FileProgressChangedEventArgs 类.....	55
4.6	通用支持接口.....	56
4.6.1	IRemoteFile 接口.....	56
4.6.2	IRemoteFolder 接口.....	56
4.6.3	IRemoteInterface 接口.....	56
4.6.4	IHardware 接口.....	57
4.6.5	IBaseDevice 接口.....	58
4.6.6	IHardwareCollection 接口.....	58
4.6.7	IModuleCollection 接口.....	59
4.6.8	IScanErrorCollection 类.....	59
4.6.9	IScanErrorEvent 类.....	61
4.6.10	IDiagnosticBuffer 接口.....	62
4.6.11	IResult 接口.....	62
4.6.12	ICertificateStore 接口.....	63
4.6.13	ICertificate 接口.....	63
4.7	网络类.....	64
4.7.1	网络构造函数.....	64
4.7.2	QueryNetworkInterfaceCards 方法.....	64
4.7.3	SetCurrentNetworkInterface 方法.....	64
4.7.4	CurrentNetworkInterface 属性.....	65
4.7.5	ScanNetworkDevices 方法.....	65
4.7.6	SetCommunicationsTimeout 方法.....	66
4.7.7	GetCommunicationsTimeout 方法.....	66
4.7.8	GetEmptyCollection 方法.....	67
4.7.9	ValidateNetworkInterface 方法.....	67
4.8	IProfinetDeviceCollection 类.....	68
4.8.1	迭代集合中的项.....	68
4.8.1.1	迭代集合中的项.....	68
4.8.1.2	FindDevicesBySerialNumber 方法.....	69
4.8.1.3	GetEnumerator 方法.....	69
4.8.1.4	SortByIP 方法.....	69
4.8.1.5	Count 属性.....	69
4.8.1.6	[] 特性.....	70
4.8.2	过滤集合中的项目.....	70
4.8.2.1	集合项.....	70
4.8.2.2	FilterByDeviceFamily 方法.....	70
4.8.2.3	FilterOnlyCPUs 方法.....	71
4.8.3	在集合中查找特定设备.....	71
4.8.3.1	FindDeviceByIP 方法.....	71
4.8.3.2	FindDeviceByMAC 方法.....	72
4.8.4	设备集合的序列化.....	73
4.8.4.1	WriteToStream 方法.....	73
4.8.4.2	ReadFromStream 方法.....	73

4.8.4.3	ExportDeviceInformation 方法.....	74
4.8.4.4	ExportDeviceDiagnostics 方法.....	74
4.8.5	手动将项目添加到集合中.....	76
4.8.5.1	InsertDeviceByIP 方法.....	76
4.8.5.2	InsertDeviceByMAC 方法.....	77
4.8.6	从集合中复制数据.....	78
4.8.6.1	CopyUserData 方法.....	78
4.8.7	从集合中移除设备.....	78
4.8.7.1	Clear 方法.....	78
4.8.7.2	Remove 方法.....	78
4.9	IProfinetDevice 接口.....	79
4.9.1	IProfinetDevice 属性.....	79
4.9.2	IProfinetDevice 方法.....	82
4.9.2.1	FirmwareUpdate 方法.....	82
4.9.2.2	FirmwareActivate 方法.....	85
4.9.2.3	Identify 方法.....	88
4.9.2.4	IsFirmwareUpdateAllowed 方法.....	88
4.9.2.5	IsFirmwareUpdate2StepAllowed 方法.....	89
4.9.2.6	IsIPAddressOnNetwork 方法.....	90
4.9.2.7	IsPROFINETNameOnNetwork 方法.....	90
4.9.2.8	IsUploadServiceDataAllowed 方法.....	91
4.9.2.9	QuickPing 方法.....	91
4.9.2.10	RefreshStatus 方法.....	92
4.9.2.11	ResetCommunicationParameters 方法.....	93
4.9.2.12	SetIP 方法.....	94
4.9.2.13	SetProfinetName 方法.....	95
4.9.2.14	UploadServiceData 方法.....	96
4.9.2.15	ValidateIPAddressSubnet 方法.....	97
4.9.2.16	ValidatePROFINETName 方法.....	97
4.9.3	IProfinetDevice 事件.....	98
4.9.3.1	DataChanged 事件.....	98
4.9.3.2	ProgressChanged 事件.....	99
4.10	IModuleCollection 类和模块属性.....	100
4.10.1	模块属性和 IModuleCollection 类.....	100
4.10.2	IModule 接口.....	100
4.11	ICPU 接口.....	102
4.11.1	识别 IProfinetDeviceCollection 中的 CPU 设备.....	102
4.11.2	ICPU 属性.....	102
4.11.3	ICPU 标志.....	104
4.11.3.1	程序更新标志.....	104
4.11.3.2	恢复标志.....	105
4.11.3.3	ICPU 支持和允许的标志.....	105
4.11.4	ICPU 方法.....	106
4.11.4.1	Backup 方法 (ICPU 接口)	106
4.11.4.2	CreateConfigurationDataProtectionCard 方法.....	107
4.11.4.3	CreateFirmwareUpdateCard 方法.....	108
4.11.4.4	CreateProgramUpdateCard 方法.....	109
4.11.4.5	DeleteConfigurationDataProtectionPassword 方法.....	110
4.11.4.6	DeleteDataLog 方法.....	110
4.11.4.7	DeleteRecipe 方法.....	111

4.11.4.8	DetermineConfirmationMessage.....	112
4.11.4.9	DownloadRecipe 方法.....	115
4.11.4.10	FormatMemoryCard 方法.....	116
4.11.4.11	GetCurrentDateTime 方法.....	118
4.11.4.12	GetDiagnosticsBuffer 方法.....	119
4.11.4.13	MemoryReset 方法.....	120
4.11.4.14	ProgramUpdate 方法.....	120
4.11.4.15	ResetToFactoryDefaults 方法.....	123
4.11.4.16	Restore 方法 (ICPU 接口)	125
4.11.4.17	SetBackupFile 方法.....	127
4.11.4.18	SetBackupFilePassword 方法.....	127
4.11.4.19	SetConfigurationDataProtectionPassword 方法.....	128
4.11.4.20	SetCurrentDateTime 方法.....	129
4.11.4.21	SetOperatingState 方法.....	129
4.11.4.22	SetPassword 方法.....	130
4.11.4.23	SetProgramFolder 方法.....	131
4.11.4.24	SetProgramPassword 方法.....	132
4.11.4.25	SetTrustCertificateStore 方法.....	132
4.11.4.26	UploadDataLog 方法.....	133
4.11.4.27	UploadRecipe 方法.....	134
4.11.4.28	ValidateProgramFolder 方法.....	135
4.11.5	RemoteInterfaces 属性.....	136
4.11.5.1	分散式 I/O 模块.....	136
4.11.5.2	IRemoteInterface 属性.....	137
4.12	ICPUCclassic 接口.....	139
4.12.1	识别 IProfinetDeviceCollection 中的经典 CPU 设备.....	139
4.12.2	GetDiagnosticsBuffer 方法.....	140
4.13	IHMI 接口.....	141
4.13.1	IHMI 接口.....	141
4.13.2	IHMI 属性和标志.....	141
4.13.2.1	IHMI 属性.....	141
4.13.2.2	程序更新标志.....	142
4.13.2.3	恢复标志.....	142
4.13.2.4	功能标志.....	142
4.13.3	IHMI 方法.....	143
4.13.3.1	Backup 方法 (IHMI 接口)	143
4.13.3.2	FirmwareUpdate 方法.....	144
4.13.3.3	ProgramUpdate 方法 (IHMI 接口)	144
4.13.3.4	Restore 方法 (IHMI 接口)	145
4.13.3.5	SetBackupFile 方法.....	146
4.13.3.6	SetProgramFolder 方法.....	146
4.13.3.7	SetTransferChannel 方法.....	147
4.14	IScalance 接口.....	147
4.14.1	IScalance 接口.....	147
4.14.2	IScalance 属性.....	148
4.14.3	IScalance 方法.....	148
4.14.3.1	SetProfile 方法.....	148
4.14.3.2	FirmwareUpdate 方法.....	149
4.14.3.3	FirmwareActivate 方法.....	150
4.15	ISNMPProfile 接口.....	150

4.15.1	ISNMPProfile 属性.....	150
4.15.2	ISNMPProfile 方法.....	151
4.15.2.1	Validate 方法.....	151
4.15.3	SNMPProfile 类.....	151
4.15.3.1	SNMPProfile 类属性.....	151
4.15.3.2	SNMPProfile 类方法.....	152
4.16	IScalance 和 ISNMP 固件更新代码示例.....	155
4.16.1	示例：SNMP 版本 1 组态.....	155
4.16.2	示例：SNMP 版本 2 组态.....	157
4.16.3	示例：SNMP 版本 3 组态.....	158
4.17	异常.....	160
4.17.1	CriticalInternalErrorException.....	160
4.18	API 枚举.....	160
4.18.1	BackupOptions.....	160
4.18.2	BackupType.....	161
4.18.3	CertificateStatusError.....	161
4.18.4	ConfigurationDataProtection.....	161
4.18.5	ConfigurationStatus.....	161
4.18.6	ConfirmationType	161
4.18.7	DataChangedType.....	162
4.18.8	DeviceFamily.....	162
4.18.9	ErrorCode.....	162
4.18.10	FailsafeOperation.....	167
4.18.11	FirmwareUpdateType.....	167
4.18.12	HMIBackupType.....	168
4.18.13	HMITransferChannel.....	168
4.18.14	Language.....	168
4.18.15	OperatingState.....	168
4.18.16	OperatingStateREQ.....	168
4.18.17	ProgressAction.....	169
4.18.18	ProtectionLevel	169
4.18.19	RedundancyRole.....	169
4.18.20	RemoteFolderType.....	170
4.18.21	RemoteInterfaceType.....	170
4.18.22	ResetOptions.....	170
4.18.23	ScanErrorType.....	170
4.18.24	SDCardType.....	170
4.18.25	SNMPAuthAlgorithm.....	170
4.18.26	SNMPPrivAlgorithm.....	171
4.18.27	SNMPSecurityLevel.....	171
4.18.28	SNMPVersion.....	171
4.18.29	TimeFormat.....	171
4.18.30	TrustCertificateType.....	171
4.19	CPU 密码访问级别.....	171
4.20	网络示例.....	173
4.21	参考信息.....	175
4.21.1	关于路由器后面的设备.....	175

目录

4.21.2	关于连接到 CM 或 CP 的设备.....	176
索引.....		177

前言

1.1 安全性信息

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/cert>)。

1.2 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：

- [SDK 示例程序概览 \(页 11\)](#)

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。示例程序是简单的命令行应用程序，通过 API 方法调用执行用户函数。用户指南的示例程序部分显示了西门子如何使用 API 在 .NET Runtime 中开发应用程序。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 \(页 34\)](#)

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。

1.3 SIMATIC Automation Tool SDK 试用版的功能

SIMATIC Automation Tool SDK 试用版不包括购买的 SDK 产品全部功能。可以购买该产品以访问全部功能。

试用版中不包含以下功能：

- 备份与恢复到 CPU
- 存储器复位
- 复位为出厂默认值
- 格式化存储卡
- 设置时间
- 读取和删除数据日志
- 创建存储卡
- 过滤扫描
- 快速 Ping

1.4 SDK 设备目录

SIMATIC Automation Tool SDK 提供设备目录。设备目录是一个电子表格，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。

安装程序会在产品安装位置的 Catalog 目录中安装设备目录。

了解 SIMATIC Automation Tool SDK 示例程序

2.1 示例应用程序简介

随 SDK 安装提供的示例程序演示了如何使用 SIMATIC Automation Tool API 中的多种接口和方法。通过学习示例程序，您将了解如何使用 API。该示例程序允许用户在设备网络上使用单个 CPU 并执行以下常见任务：

- 选择用于 CPU 通信的网络接口 ([页 19](#))
- 按 IP 地址插入 CPU ([页 21](#))
- 显示 CPU 的识别信息 ([页 25](#))
- 更改 CPU 的操作状态 ([页 27](#))
- 更改 CPU 的 IP 地址、子网和网关 ([页 28](#))
- 设置 CPU 的 PROFINET 名称 ([页 29](#))
- 更新 CPU 的固件 ([页 30](#))

打开并运行示例程序

要打开示例程序，请按以下步骤操作：

1. 打开命令行接口应用程序。
2. 浏览到产品安装目录。
3. 从产品安装目录浏览到示例目录。
4. 输入以下命令以编译示例程序：dotnet build
5. 要运行示例程序，请从当前目录浏览到“bin/x64/Debug”。然后，输入以下命令：./SATExample

说明

可能需要将 Examples 目录从安装目录复制到用户有读写权限的位置。如果是这种情况，必须复制整个 Examples 目录。随后可通过命令行打开 SATExample.sln 文件或编译并运行该文件。

示例应用程序运行全套 API 功能 ([页 34](#)) 的一部分。在熟悉使用 API 编程后，可以使用可用的 API 接口、类别和方法。

2.2 创建使用 API 的项目

使用首选的文本编辑器通过 SIMATIC Automation Tool API ([页 34](#)) 开发应用程序。用户应用程序和 SIMATIC Automation Tool API .dll 文件必须位于同一目录中。

西门子使用 Microsoft Visual Studio 2022 以及 .NET 6.0 Runtime 开发了 API。本文档中的所有代码示例均采用 C# 编程语言。

组态项目以使用 API

要组态项目以使用 API, 请按以下步骤操作 :

1. 打开解压缩 .dll 文件的位置, 并将“Bin”文件夹的内容复制到 \bin\x64\Debug 文件夹中。
2. 在项目中添加对 bin\x64\Debug\SIMATICAutomationToolAPI.dll 的引用。
3. 在项目中, 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。
4. 构建您的解决方案并验证它是否成功构建。

2.3 为示例程序设置代码

本章所述的方法存在于内部类 StateMachine 中。该类包含使用 API (页 34) 运行程序所需的变量声明、enum 值和方法。除了完成程序中每个动作的方法之外, 该类还包含辅助方法。

说明

请注意, 示例目录中提供的所有内容都是编译和运行示例程序所需的内容。

所需的 using 语句

除了类之外, 程序还包含以下 using 语句 :

`using Siemens.Automation.AutomationTool.API;`

要编译示例程序, “`using Siemens.Automation.AutomationTool.API;`”必须与程序代码存在于同一文件中或存在于全局 using 文件中。

声明

以下代码包含 StateMachine 类中的初始变量声明：

```
private readonly ResourceStrings resourceString = new();
// Default to English before a language is selected
private Language language = Language.English;
private ICPU? CurrentCPU;
Network m_network = new();
Result result = new Result();
private int progressPercentage = -1;
private readonly ResourceStrings resourceString = new();
private uint badIP = 0xffffffff;
List<String>? NetworkInterfaces;
24 references
enum ApplicationStates
{
    Introduction,
    LanguageSelection,
    NICSelection,
    IPAddressEntry,
    CertificateTrustSelection,
    CPUPasswordEntry,
    CommandSelection,
    ExitApplication
}
//Set the starting state as the introduction
ApplicationStates theState = ApplicationStates.Introduction;
```

Start() 方法

程序包含 Start() 方法，其中包含 while 循环，会将程序从一种状态转换为另一种状态。该循环会无限执行，直到用户退出程序。该循环包含根据用户选择调用方法的 switch 语句。switch 语句调用以下方法：

- SetLanguage() (页 18)
- DetermineNICOptions() (页 19)
- SetNIC() (页 19)
- GetIP() (页 21)
- TrustCertificate() (页 21)
- GetPassword() (页 21)
- Commands()

一些程序方法在 switch 语句中设置。

如果存在程序无法处理的错误，则程序将退出。

2.3 为示例程序设置代码

Commands() 方法

程序包含向用户显示每个命令选项并允许用户选择其中一个选项的 Commands() 方法。该方法使用 switch 语句来显示每个选项。随后它会调用 CheckForChanges() 方法。代码如下：

```
// This method presents the user with each of the command choices and allows them to select one
1 reference
private void Commands()
{
    // Commands
    Console.WriteLine();
    Console.WriteLine(resourceString.GetString("commandQuestion", language));
    Console.WriteLine(resourceString.GetString("identify", language));
    Console.WriteLine(resourceString.GetString("basicDeviceInfo", language));
    Console.WriteLine(resourceString.GetString("moduleInfo", language));
    Console.WriteLine(resourceString.GetString("changeOperatingState", language));
    Console.WriteLine(resourceString.GetString("setIP", language));
    Console.WriteLine(resourceString.GetString("setPROFINET", language));
    Console.WriteLine(resourceString.GetString("firmwareUpdate", language));
    Console.WriteLine(resourceString.GetString("pickNewDevice", language));
    Console.WriteLine(resourceString.GetString("exit", language));
    Console.WriteLine();
    Console.Write(resourceString.GetString("promptForCommand", language));
    var command = ReadLine();
    command = TrimSpaces(command);

    switch (command)
    {
        // Identify
        case "1":
            Identify(CurrentCPU);
            break;
        // Read Basic Device Information
        case "2":
            PrintBasicDeviceInformation(CurrentCPU);
            break;
        // Read Module Information
        case "3":
            PrintModuleInformation(CurrentCPU);
            break;
        // Change Operating state
        case "4":
            ChangeState();
            break;
        // Set IP Address
        case "5":
            SetIP(CurrentCPU);
            break;
        // Set PROFINET Name
        case "6":
            SetProfinetName(CurrentCPU);
            break;
        // Firmware Update
        case "7":
            FirmwareUpdate(CurrentCPU);
            break;
        // Pick New Device
        case "8":
            theState = ApplicationStates.IPAddressEntry;
            break;
        // Exit
        case "9":
            theState = ApplicationStates.ExitApplication;
            break;
        // Error
        default:
            Console.WriteLine(resourceString.GetString("commandError", language));
            break;
    }
    CheckForChanges();
}
}
```

CheckForChanges() 方法

该方法检测 CPU 元素何时发生更改或命令之间何时出现错误：

```
// This method is a concise way to check for potential errors between commands
1 reference
private void CheckForChanges()
{
    // If the identity of the CPU has changed
    if (CurrentCPU.IdentityCrisis)
    {
        Console.WriteLine();
        Console.WriteLine(resourceString.GetString("identityCrisisError", language));
        theState = ApplicationStates.IPAddressEntry;
    }
    // If the certificate has changed
    if (CurrentCPU.TLSTrustRequired && CurrentCPU.TrustCertificateStore == TrustCertificateType.SelectionNeeded)
    {
        Console.WriteLine();
        Console.WriteLine(resourceString.GetString("certificateChanged", language));
        theState = ApplicationStates.CertificateTrustSelection;
    }
    // If the password has changed
    if (CurrentCPU.Protected && !CurrentCPU.PasswordValid)
    {
        Console.WriteLine();
        Console.WriteLine(resourceString.GetString("passwordChange", language));
        theState = ApplicationStates.CPUPasswordEntry;
    }
}
```

Identify(ICPU? CurrentCPU) 方法

该方法调用 API 来识别 CPU：

```
// This method calls the API Identify command.
1 reference
public void Identify(ICPU? CurrentCPU)
{
    // Call API to identify the CPU
    result = CurrentCPU.Identify();
    PrintMessages(result);
}
```

2.3 为示例程序设置代码

`CurrentCPU_ProgressChanged(object sender, ProgressChangedEventArgs e)` 方法

该方法为其放置的命令提供了一个进度条：

```
// This method provides a progress bar for the command it is placed around. See Firmware Update for an example.

private void CurrentCPU_ProgressChanged(object sender, ProgressChangedEventArgs e)
{
    // If statement ensures we update the bar during each action
    if (e.Action == ProgressAction.Updating || e.Action == ProgressAction.Downloading ||
        e.Action == ProgressAction.Processing || e.Action == ProgressAction.Rebooting)
    {
        if (e.Index != progressPercentage)
        {
            progressPercentage = e.Index;
            // Add item every 5% of progress
            if (progressPercentage % 5 == 0)
            {
                Console.WriteLine(".");
            }
        }
    }
}
```

`PrintMessages(Result result)` 方法

该方法从 `Result` 对象获取所有成功、错误和警告消息，并打印这些消息：

```
// This method gets all success, error, and warning messages from a Result object and prints them.

private void PrintMessages(Result result)
{
    // Do we have warnings?
    String[] aWarnings = result.GetWarningDescription(language);
    for (int i = aWarnings.Length - 1; i >= 0; i--)
        Console.WriteLine("WARNING: " + aWarnings[i]);

    // If an error occurred
    if (result.Succeeded)
        Console.WriteLine("SUCCESS: " + result.GetErrorDescription(language));
    else
        Console.WriteLine("ERROR: " + result.GetErrorDescription(language));
}
```

PrintMessages (IScanErrorCollection errorCollection) 方法

该方法从 IScanErrorCollection 对象获取所有成功、错误和警告消息，并打印这些消息：

```
// This method gets all success, error, and warning messages from an IScanErrorCollection object and prints them.

private void PrintMessages(IScanErrorCollection errorCollection)
{
    for (int i = errorCollection.Count - 1; i >= 0; i--)
    {
        IScanErrorEvent scanErrorEvent = errorCollection[i];

        Result result = new Result(scanErrorEvent.Code);
        switch (scanErrorEvent.Type)
        {
            case ScanErrorType.Success:
                Console.WriteLine("SUCCESS: " + result.GetErrorDescription(language));
                break;
            case ScanErrorType.Warning:
                Console.WriteLine("WARNING: " + result.GetErrorDescription(language));
                break;
            case ScanErrorType.Information:
                Console.WriteLine("INFORMATION: " + result.GetErrorDescription(language));
                break;
            case ScanErrorType.Error:
                Console.WriteLine("ERROR: " + result.GetErrorDescription(language));
                break;
        }
    }
}
```

ReadLine () 方法

该方法读取用户输入并检查问题。如果解析过程中出现问题，程序将进入错误状态：

```
//This method reads user input and checks to see if it is null

private string ReadLine()
{
    string aString = Console.ReadLine();
    if (aString == null)
        System.Environment.Exit(0);

    return aString;
}
```

2.4 选择示例程序的语言

TrimSpaces(string trimString) 方法

该方法从用户输入中截断前导和尾部空格：

```
// This method is a tool for trimming leading and trailing spaced from user input.  
private string TrimSpaces(string trimString)  
{  
    char[] charsToTrim = { ' ' };  
    string resultString = trimString.Trim(charsToTrim);  
    return resultString;  
}
```

2.4 选择示例程序的语言

示例程序允许用户为菜单文本、提示和消息设置语言。

语言选择示例代码

SetLanguage() 方法打印语言选项并提示用户进行选择：

```
// This method allows the user to select the language that the program will use when outputting to the screen

private bool SetLanguage()
{
    // Select from the following language options:
    // 1: Deutsch
    // 2: English
    // 3: Español
    // 4: Français
    // 5: Italiano
    // 6: Chinese
    // Language Selection:

    Console.WriteLine();
    Console.WriteLine("Select from the following language options: ");
    Console.WriteLine(" 1: Deutsch");
    Console.WriteLine(" 2: English");
    Console.WriteLine(" 3: Español");
    Console.WriteLine(" 4: Français");
    Console.WriteLine(" 5: Italiano");
    Console.WriteLine(" 6: Chinese");
    Console.WriteLine();
    Console.Write("Language Selection: ");
    string languageChoice = ReadLine();

    languageChoice = TrimSpaces(languageChoice);

    switch (languageChoice)
    {
        case "1":
            // Language is German
            language = Language.German;
            return true;
        case "2":
            // Language is English
            language = Language.English;
            return true;
        case "3":
            // Language is Spanish
            language = Language.Spanish;
            return true;
        case "4":
            // Language is French
            language = Language.French;
            return true;
        case "5":
            // Language is Italian
            language = Language.Italian;
            return true;
        case "6":
            // Language is Chinese
            language = Language.Chinese;
            return true;
        default:
            return false;
    }
}
```

示例程序中的语言选项包括 SDK 安装文件夹中提供的语言。语言在 Start() 方法 (页 12) 中设置。

2.5 选择网络接口

示例程序确定用户编程设备可用的网络接口。借助可用接口，用户可以选择一个用于通信的接口。

确定网络接口选项

"Network (页 63)"是 API 提供的类别。

每当用户选择网络接口时，应用程序都会清除任何现有的 CPU 状态信息。

以下代码确定是否有可用的网络接口：

```
// This method determines if there are NIC options available
1 reference
private bool DetermineNICOptions()
{
    // Get a list of all NICs
    result = m_network.QueryNetworkInterfaceCards(out NetworkInterfaces);
    if (result.Succeeded)
    {
        return true;
    }
    PrintMessages(result);
    return false;
}
```

选择网络接口并进行设置

程序显示可用的网络接口并允许用户选择一个接口。然后，程序使用输入来设置网络接口。

如果用户不使用路由器，请选择名称中包含".Auto"的网络接口。这一选择会查找网络上的所有设备。如果用户选择名称中包含".Auto"的接口，且正在与之通信的设备不在同一子网中，网络会向其网络接口添加一个兼容的 IP 地址，以便可以访问该设备。如果名称中没有".Auto"，用户必须向其网络接口手动添加 IP 地址，然后才能与该设备进行通信。

选择并设置网络接口的代码如下：

```
// This method is designed to get all the network interface cards and allow a user to select one.  
// Once they do it calls the API function to set the NIC.  
1 reference  
public bool SetNIC()  
{  
    Console.WriteLine();  
    Console.WriteLine(resourceString.GetString("selectNICQuestion", language));  
    int i = 0;  
    foreach (var item in NetworkInterfaces)  
    {  
        i++;  
        // Print the NICs  
        Console.WriteLine(" " + i + ": " + item);  
    }  
    Console.WriteLine();  
    Console.Write(resourceString.GetString("networkInterfacePrompt", language) + " ");  
    var NICNumString = ReadLine();  
    NICNumString = TrimSpaces(NICNumString);  
    int.TryParse(NICNumString, out int NICNum);  
    if (NICNum <= i && NICNum > 0)  
    {  
        string NIC = NetworkInterfaces[NICNum - 1];  
  
        // Call API to set the NIC  
        result = m_network.SetCurrentNetworkInterface(NIC);  
        PrintMessages(result);  
        if (result.Succeeded)  
        {  
            return true;  
        }  
    }  
    return false;  
}
```

2.6 通过 IP 地址连接到网络上的 CPU

示例程序允许用户连接到网络上的 CPU。随后，该 CPU 将成为设备指令的连接设备。

输入 IP 地址并连接到 CPU

示例程序提示用户输入 IP 地址并将 CPU 添加到设备表。这将连接 CPU。

该方法调用 ParseIP 方法。有关 ParseIP 的更多信息，请参见下一部分。

用于连接 CPU 的代码如下：

```
// This method gets the desired CPU IP address from the user, only CPUs are supported
public bool GetIP()
{
    Console.WriteLine();
    Console.Write(resourceString.GetString("targetIPAddressPrompt", language) + " ");
    var rawIPAddress = ReadLine();
    rawIPAddress = TrimSpaces(rawIPAddress);
    var IPAddress = ParseIP(rawIPAddress);
    if (IPAddress == badIP)
    {
        return false;
    }
    var Device = Network.GetEmptyCollection();

    // Add device to device table
    IProfinetDevice InsertedDevice;
    IScanErrorCollection InsertErrorCollection = Device.InsertDeviceByIP(IPAddress, out InsertedDevice);
    if (InsertErrorCollection.Failed)
    {
        // Output error messages to the screen if the insert fails
        PrintMessages(InsertErrorCollection);
        return false;
    }

    // Only CPUs are supported by this application
    CurrentCPU = InsertedDevice as ICPU;
    if (CurrentCPU == null)
    {
        Console.WriteLine(resourceString.GetString("notSupportedError", language));
        return false;
    }

    // Output the success message to the screen
    PrintMessages(InsertErrorCollection);
    return true;
}
```

说明

请注意，[IProfinetDeviceCollection](#)(页 67)是 API 提供的类别，[ICPU](#)(页 101)是接口。本示例应用程序在所有示例中均使用 Devices 集合和 CurrentCPU 对象。

处理输入的 IP 地址

ParseIP 方法将 IP 地址从文本转换为 32 位无符号整数。该方法还包含错误处理以确保表单包含有效的 IP 条目：

```
// This method parses the IP Address that was input by the user into a usable IP for the program.  
4 references  
public UInt32 ParseIP(string strNetParm)  
{  
    string[] splitString = strNetParm.Split(".");  
    // Check that the IP has 4 entries  
    if (splitString.Length != 4)  
    {  
        Console.WriteLine(resourceString.GetString("parsingError", language));  
        return badIP;  
    }  
    //Check that each of the entries are non-empty  
    foreach (string str in splitString)  
    {  
        if(str.Length <=0)  
        {  
            Console.WriteLine(resourceString.GetString("parsingError", language));  
            return badIP;  
        }  
    }  
    try  
    {  
        System.Net.IPEndPoint ip = System.Net.IPEndPoint.Parse(strNetParm);  
        byte[] bytes = ip.GetAddressBytes();  
        Array.Reverse(bytes);  
        return BitConverter.ToInt32(bytes, 0);  
    }  
    catch  
    {  
        Console.WriteLine(resourceString.GetString("parsingError", language));  
        return badIP;  
    }  
}
```

选择证书信任等级

程序会检测 CPU 是否使用证书。如果使用证书，则用户可以选择证书信任等级，提供以下选项：

- 始终 (Always)
- 从不 (Never)

以下代码提示用户输入并接受或拒绝证书：

```
// If needed, this method allows the user to trust the CPU certificate

private bool TrustCertificate()
{
    // Secure TLS communications might be enabled. If TLS is not enabled then trusting the certificate is not necessary.
    // If TLS is enabled the user must trust the certificate if it is not automatically trusted because it is signed.
    if (CurrentCPU.TLSTrustRequired)
    {
        if (CurrentCPU.TrustCertificateStore != TrustCertificateType.Always)
        {
            // TLS secure connection detected. Choose a certificate trust option:
            // 1: Always
            // 2: Never

            // Certificate Option:
            Console.WriteLine();
            Console.WriteLine(resourceString.GetString("TLSQuestion", language));
            Console.WriteLine(resourceString.GetString("TLSAlwaysOption", language));
            Console.WriteLine(resourceString.GetString("TLSNeverOption", language));
            Console.WriteLine();
            Console.Write(resourceString.GetString("certificateOptionPrompt", language));
            var certificateTrustOption = ReadLine();
            certificateTrustOption = TrimSpaces(certificateTrustOption);
            switch (certificateTrustOption)
            {
                // Accept the certificate
                case "1":
                    result = CurrentCPU.SetTrustCertificateStore(TrustCertificateType.Always);
                    PrintMessages(result);
                    break;
                // Reject the certificate and start over at the IP address entry
                case "2":
                    result = CurrentCPU.SetTrustCertificateStore(TrustCertificateType.Never);
                    PrintMessages(result);
                    Console.WriteLine(resourceString.GetString("communicationsDisabledWarning", language));
                    theState = ApplicationStates.IPAddressEntry;
                    return false;
                // Error handling if the user puts in a bad input
                default:
                    Console.WriteLine(resourceString.GetString("commandError", language));
                    return false;
            }
        }
    }
    return true;
}
```

输入密码

如果 CPU 受密码保护，程序会提示用户输入 CPU 密码并验证其输入。程序还会检查密码是否具有足够的访问级别。如果 CPU 受到保护，则需要读写密码。

```
// This method prompts the user for the CPU password and then validates it

private bool GetPassword()
{
    // The CPU might be protected.
    // If the CPU is protected then the read-write password is needed for the functionality of this program.
    if (CurrentCPU.Protected)
    {
        Console.WriteLine();
        Console.Write(resourceString.GetString("enterPasswordQuestion", language) + " ");
        var plcPassword = ReadLine();
        plcPassword = TrimSpaces(plcPassword);

        // Check for a NULL or empty password string
        // If this is the case then output an error and prompt again
        if (plcPassword == String.Empty)
        {
            Console.WriteLine(resourceString.GetString("emptyPasswordError", language));
            return false;
        }

        // Call API to set the CPU password
        result = CurrentCPU.SetPassword(new EncryptedString(plcPassword));
        if (result.Failed)
        {
            PrintMessages(result);
            return false;
        }

        // Check that the password entered has sufficient access level for this program
        bool bSufficientAccess = CurrentCPU.PasswordProtectionLevel == ProtectionLevel.Failsafe ||
                               CurrentCPU.PasswordProtectionLevel == ProtectionLevel.Full;
        if (CurrentCPU.PasswordValid && !bSufficientAccess)
        {
            Console.WriteLine(resourceString.GetString("insufficientAccessError", language));
            return false;
        }

        // Output the success message to the screen
        PrintMessages(result);
    }

    return true;
}
```

2.7 显示 CPU 和模块信息

连接到 CPU [\(页 21\)](#) 后，示例程序允许用户显示有关 CPU 及其模块的信息。

显示 CPU 信息

以下代码读取并打印基本 CPU 信息：

```
// This method reads and prints the basic device information of the CPU
public void PrintBasicDeviceInformation(ICPU? CurrentCPU)
{
    // A refresh status is needed here because this command is not coming directly from the API.
    result = CurrentCPU.RefreshStatus(false);
    if (result.Failed)
    {
        PrintMessages(result);
        return;
    }
    Console.WriteLine();

    // Type of Device
    Console.WriteLine(resourceString.GetString("DeviceTypeLabel", language) + " " + CurrentCPU.Description);
    // Article Number
    Console.WriteLine(resourceString.GetString("articleNumberLabel", language) + " " + CurrentCPU.ArticleNumber);
    // Serial Number
    Console.WriteLine(resourceString.GetString("serialNumberLabel", language) + " " + CurrentCPU.SerialNumber);
    // Hardware Number
    Console.WriteLine(resourceString.GetString("hardwareNumberLabel", language) + " " + CurrentCPU.HardwareNumber);
    // Firmware Version
    Console.WriteLine(resourceString.GetString("firmwareVersionLabel", language) + " " + CurrentCPU.FirmwareVersion);
    // MAC Address
    Console.WriteLine(resourceString.GetString("MACLabel", language) + " " + CurrentCPU.MACString);
    // IP Address
    Console.WriteLine(resourceString.GetString("ipPrompt", language) + " " + CurrentCPU.IPString);
    // PROFINET Name
    Console.WriteLine(resourceString.GetString("PROFINETLabel", language) + " " + CurrentCPU.ProfinetName);
    // Operating State
    Console.WriteLine(resourceString.GetString("operatingStatusLabel", language) + " " +
                      CurrentCPU.OperatingMode.ToString().ToUpper());
    // Totally Integrated Automation Portal Version
    Console.WriteLine(resourceString.GetString("TIAPVersionLabel", language) + " " + CurrentCPU.TIAPVersion);
}
```

显示模块信息

以下代码读取并打印已连接 CPU 的所有模块的模块信息：

```
// This method reads and prints the module information for all local modules connected to the CPU
1 reference
public void PrintModuleInformation(ICPU? CurrentCPU)
{
    // A refresh status is needed here because this command is not coming directly from the API.
    result = CurrentCPU.RefreshStatus(false);
    PrintMessages(result);
    if (result.Failed)
    {
        return;
    }

    // If there are no Modules
    if (CurrentCPU.Modules.Count == 0)
    {
        Console.WriteLine(resourceString.GetString("noModulesFoundError", language));
        return;
    }

    // Loop over all modules and print each one
    foreach (var module in CurrentCPU.Modules)
    {
        Console.WriteLine();
        // Name
        Console.WriteLine(resourceString.GetString("nameLabel", language) + " " + module.Name);
        // Type of Device
        Console.WriteLine(resourceString.GetString("DeviceTypeLabel", language) + " " + module.Description);
        // Slot number
        Console.WriteLine(resourceString.GetString("slotLabel", language) + " " + module.SlotName);
        // Configuration Status
        Console.WriteLine(resourceString.GetString("configurationLabel", language) + " " + module.StatusConfiguration);
        // Article Number
        Console.WriteLine(resourceString.GetString("articleNumberLabel", language) + " " + module.ArticleNumber);
        // Serial Number
        Console.WriteLine(resourceString.GetString("serialNumberLabel", language) + " " + module.SerialNumber);
        // Hardware Number
        Console.WriteLine(resourceString.GetString("hardwareNumberLabel", language) + " " + module.HardwareNumber);
        // Firmware Version
        Console.WriteLine(resourceString.GetString("firmwareVersionLabel", language) + " " + module.FirmwareVersion);
    }
}
```

2.8 切换 CPU 的操作状态

在示例程序连接到网络接口并将 CPU 插入到设备集合之后，允许用户切换 CPU 的操作状态。

说明

切换操作状态要求 CPU 包含有效的用户程序。 (未复位为出厂设置。)

读取用户输入以切换操作状态的示例代码

`ChangeState()` 方法允许用户将 CPU 状态切换为 RUN 或 STOP。以下代码用于获取用户输入并设置操作状态：

```
// This method allows the user to change the operating state of the CPU.
2 references
public void ChangeState()
{
    Console.WriteLine();
    Console.WriteLine(resourceString.GetString("operatingStateQuestion", language));
    Console.WriteLine(resourceString.GetString("runChoice", language));
    Console.WriteLine(resourceString.GetString("stopChoice", language));
    Console.WriteLine();
    Console.Write(resourceString.GetString("statePrompt", language));
    var state = ReadLine();
    state = TrimSpaces(state);
    CurrentCPU.Selected = true;
    switch (state)
    {
        //Run
        case "1":
            result = CurrentCPU.SetOperatingState(OperatingStateREQ.Run);
            PrintMessages(result);
            break;
        //Stop
        case "2":
            result = CurrentCPU.SetOperatingState(OperatingStateREQ.Stop);
            PrintMessages(result);
            break;
        //Error
        default:
            Console.WriteLine(resourceString.GetString("commandError", language) + " ");
            ChangeState();
            break;
    }
}
```

2.9 设置 CPU 的 IP 地址、子网和网关

示例程序允许用户设置 CPU 的 IP 地址、子网和网关。

说明

如果 CPU 包含 STEP 7 用户程序，该操作需要满足以下条件：

CPU 的 PROFINET 接口 IP 协议的设备组态指定在 TIA 项目中“在设备中直接设置 IP 地址”(IP address is set directly at the device) 和“在设备中直接设置 PROFINET 设备名称”(PROFINET device name 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)

接受用户输入并更改 IP 协议值

以下代码接受并设置每个值的用户输入：

```
// This method prompts the user for a new IP address then calls the API to set the new IP address
1 reference
public void SetIP(ICPU? CurrentCPU)
{
    Console.WriteLine();
    Console.Write(resourceString.GetString("enterIPQuestion", language) + " ");
    var newIP = ReadLine();
    newIP = TrimSpaces(newIP);
    var newIPParsed = ParseIP(newIP);
    if (newIPParsed == badIP)
    {
        Console.WriteLine(resourceString.GetString("invalidIP", language));
        return;
    }

    Console.Write(resourceString.GetString("enterSubnetQuestion", language) + " ");
    var subnet = ReadLine();
    subnet = TrimSpaces(subnet);
    var parsedSubnet = ParseIP(subnet);
    if (parsedSubnet == badIP)
    {
        Console.WriteLine(resourceString.GetString("invalidSubnet", language));
        return;
    }

    Console.Write(resourceString.GetString("enterGatewayQuestion", language) + " ");
    var gateway = ReadLine();
    gateway = TrimSpaces(gateway);
    var parsedGateway = ParseIP(gateway);
    if (parsedGateway == badIP)
    {
        Console.WriteLine(resourceString.GetString("invalidGateway", language));
        return;
    }

    result = CurrentCPU.SetIP(newIParsed, parsedSubnet, parsedGateway);
    PrintMessages(result);
}
}
```

2.10 设置 CPU 的 PROFINET 名称

示例程序允许用户设置已连接 CPU 的 PROFINET 名称。

说明

如果 CPU 包含 STEP 7 用户程序，该操作需要满足以下条件：

CPU 的 PROFINET 接口 IP 协议的设备组态指定在 TIA 项目中“在设备中直接设置 IP 地址”(IP address is set directly at the device) 和“在设备中直接设置 PROFINET 设备名称”(PROFINET device name 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)
-

设置 PROFINET 名称

示例程序读取用户输入并设置 PROFINET 名称：

```
// This method prompts the user for a new PROFINET name and then calls the API to set the new PROFINET name

public void SetProfinetName(ICPU? CurrentCPU)
{
    Console.WriteLine();
    Console.Write(resourceString.GetString("PROFINETQuestion", language) + " ");
    var newProfinetName = ReadLine();
    newProfinetName = TrimSpaces(newProfinetName);
    result = CurrentCPU.SetProfinetName(newProfinetName);
    PrintMessages(result);
}
```

2.11 更新 CPU 的固件

示例程序允许用户输入固件更新文件。然后，程序尝试使用输入更新 CPU。

说明

更新 CPU 固件要求 CPU 中具有用户程序。（CPU 未复位为出厂默认值。）

选择文件并更新 CPU 固件

以下代码接受来自用户的 UDP 固件更新文件，然后尝试更新 CPU：

```
// This method gets a firmware update (UDP) file from the user and then calls the API to update the firmware
public void FirmwareUpdate(ICPU? CurrentCPU)
{
    Console.WriteLine();
    Console.Write(resourceString.GetString("UDPFFilePathQuestion", language) + " ");
    var udpFile = ReadLine();
    udpFile = TrimSpaces(udpFile);
    // Validate the file
    result = CurrentCPU.SetFirmwareFile(udpFile);
    if (result.Succeeded)
    {
        CurrentCPU.Selected = true;

        // Begin progress bar
        CurrentCPU.ProgressChanged += CurrentCPU_ProgressChanged;
        Console.Write(resourceString.GetString("progressBar", language));
        // Update the firmware
        result = CurrentCPU.FirmwareUpdate(CurrentCPU.ID, true);

        Console.WriteLine();
        PrintMessages(result);
        // Finish progress bar
        CurrentCPU.ProgressChanged -= CurrentCPU_ProgressChanged;
    }
    // File path invalid
    else
    {
        PrintMessages(result);
        FirmwareUpdate(CurrentCPU);
    }
    Console.WriteLine();
}
```

2.12 将应用程序分发给您的最终用户

应用程序的最终用户需要以下内容：

- Bin 目录
- Prerequisite 目录
- 自定义应用程序 .exe 文件以及与您的应用程序关联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 目录中。

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。安装程序会在安装目录的“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
德语	ReadMe_OSS_de_DE.html
英语	ReadMe_OSS_en_US.html
西班牙语	ReadMe_OSS_es_ES.html

2.12 将应用程序分发给您的最终用户

语言文件夹	文件名
法语	ReadMe_OSS_fr_FR.html
意大利语	ReadMe_OSS_it_IT.html
简体中文	ReadMe_OSS_zh_CHS.html

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。

为用户创建安装程序

使用选择的工具，为用户创建一个安装必要文件和目录的安装程序。

更新使用旧版 SDK 制作的自定义应用程序

如果使用早期版本的 SIMATIC Automation Tool SDK 创建并分发了自定义应用程序，则可以将其更新为 SDK 的最新版本，然后将其重新分发给最终用户。SIMATIC Automation Tool SDK 的更新版本可以提供改进和增强功能，这可能会改变应用程序编程接口 (API) (页 34)。此外，每个发行版的支持组件可能有所不同。

西门子建议您将自定义应用程序更新为最新发布的版本：

- 重新编译项目 (页 11)。
- 请更正 API 改变可能导致的所有编译错误。
- 测试您的应用程序，以确保与 API 的功能兼容性。
- 如前所述，将更新后的项目分发给 (页 31) 最终用户。

4.1 API 入门指南

用于 PROFINET 网络和设备维护的 SIMATIC Automation Tool 应用程序编程接口 (API) 允许用户使用各种 .NET 接口、类和方法 ([页 38](#)) 创建自定义应用程序。您可以针对特定用途设计自定义应用程序。

设置网络接口

应用程序必须执行的第一个任务是设置或选择用于与 PROFINET 网络通信的网络接口。您的计算机或编程设备很可能有多个网络接口。

如果您不知道想要使用的网络接口的名称，可以使用网络类别的 `QueryNetworkInterfaceCards` ([页 64](#)) 方法读取全部网络接口的列表。可以将此列表展示给您的应用程序用户，以选择特定网络接口。

然后，可以使用 `SetCurrentNetworkInterface` ([页 64](#)) 方法设置应用程序用于设备通信的网络接口。

有关选择网络接口的具体详细信息，请参见示例应用程序一章中的“选择网络接口”。

与设备通信

您在设计应用程序时，可以选择在通过网络扫描找到的设备上运行，还是在通过 `Insert Device` 方法 ([页 76](#)) 添加的设备上运行。如果您的网络上仅有少量设备，并且您已知道这些设备的 IP 或 MAC 地址，则基于应用程序性能的考量，可以使用 `Insert Device` 方法。如果您不知道 IP 或 MAC 地址，则使用 `ScanNetworkDevices` ([页 65](#)) 方法以找到连接 PROFINET 网络的全部设备。

扫描网络

应用程序可能需要连接到 PROFINET 网络并进行扫描，以找到全部连接的设备。在这种情况下，应用程序可以扫描网络，从而能够将网络上的所有设备填入一个集合中。为此，API 提供了 `ScanNetworkDevices` ([页 65](#)) 方法。此方法将 DCP 广播命令发送到网络并将可访问设备填入 `IProfinetDeviceCollection` ([页 67](#))。

此方法还允许指定限制扫描中包含的设备的过滤器。该过滤器定义网络中的哪些 IP 地址包含在扫描结果中。任何与 IP 地址过滤器不符的设备都将从扫描结果中删除。这样可大幅缩短扫描和设备初始化时间。

插入设备

如果您或应用程序用户知道设备的 IP 或 MAC 地址，您可能希望将应用程序设计为您或您的用户可以添加特定设备。API 提供通过 IP 地址或 MAC 地址插入设备的方法。如果您预期用户会在添加特定设备后在这些设备上执行操作，则可以选择基于 [InsertDeviceByIP \(页 76\)](#) 方法或 [InsertDeviceByMAC \(页 77\)](#) 方法开发应用程序。

说明

路由器后面的设备

如果要与路由器后面的设备通信，必须使用 [InsertDeviceByIP \(页 76\)](#) 方法。网络扫描无法发现路由器后面的设备。

使用受保护 CPU

如果 CPU 受到保护，则对于 ICPU 接口，[Protected \(页 102\)](#) 属性为 true。在调用 [SetPassword \(页 130\)](#) 方法前必须先检查 CPU 是否受到保护。如果 CPU 未受到保护，则不要调用 [SetPassword \(页 130\)](#) 方法。如果为没有受到保护的 CPU 调用 [SetPassword \(页 130\)](#) 方法，API 会引发严重错误异常。出现严重错误异常意味着您未正确使用 API。对于受保护的 CPU，应用程序必须使用能提供充分访问级别 ([页 171](#)) 的密码调用 [SetPassword \(页 130\)](#) 方法。设置有效密码后，可以刷新设备状态。

最佳做法是，在扫描网络后要立即进行通信的所有设备上都调用 [SetPassword \(页 130\)](#)。将标准 CPU 上的密码设置为具有完全访问权限的密码 ([页 171](#))，将 F-CPU 上的密码设置为安全密码 ([页 171](#))，以避免出现 API 错误。

CPU 的固件可能支持传输层安全 (TLS)，从而提供安全通信。传输层安全是一种专门保证通信网络安全的加密协议。对支持安全通信的 CPU 执行设备操作时，SAT API 客户端必须满足额外的 TLS 要求，如信任 CPU 数字证书。

更多信息，请参见证书处理 API 接口。了解 TLS 概念和证书处理将有所帮助。

使用 RefreshStatus

[IProfinetDeviceCollection \(页 67\)](#) 中表示设备的对象在网络扫描后仅含有各设备的部分数据。要获得关于设备的全部数据并正确使用 API，必须执行以下步骤：

1. 为每个受保护的 CPU 调用 [SetPassword \(页 130\)](#)。要读取全部设备数据，CPU 密码必须提供足够的权限。
2. 为 [IProfinetDeviceCollection \(页 67\)](#) 中的每个设备调用 [RefreshStatus \(页 92\)](#) 方法。

[RefreshStatus \(页 92\)](#) 方法更新表示设备状态的全部数据。

严重错误异常

如果 API 引发严重错误异常，说明您未正确地将 API 用于设备状态。记得先调用 [RefreshStatus \(页 92\)](#) 方法来将设备的数据更新为最新状态，然后再使用其它 API 功能。

其它要求

许多 API 方法的使用有特定的先决条件。方法和接口说明列出了适用的先决条件。要执行设备特定的操作，应用程序必须将 Selected 属性设为 True 才能执行设备操作。
在应用程序设计中，F-CPU 上的安全相关操作 ([页 41](#)) 需要额外的保护措施。

示例：通过扫描网络初始化设备

```
using Siemens.Automation.AutomationTool.API;
using System.Security.Cryptography.X509Certificates;
#region
//-----
//-----
//-----
Network myNetwork = new Network();
List<String> interfaces = new List<String>();
Result retVal = myNetwork.QueryNetworkInterfaceCards
(out interfaces);
if (retVal.Succeeded)
{
    for (Int32 index = 0; index < interfaces.Count; index++)
    {
        String strInterfaceName = interfaces[index];
        // -----
        // PROFINET
        // NIC
        //      NIC
        // -----
        if (strInterfaceName == "myNIC.Auto.1")
        {
            retVal = myNetwork.SetCurrentNetworkInterface
(strInterfaceName);
        }
    }
    //-----
    //-----
    //-----
IProfinetDeviceCollection scannedDevices;
IScanErrorCollection scanResults = myNetwork.ScanNetworkDevices
(out scannedDevices);
if (scanResults.Succeeded)
{
    foreach (IProfinetDevice device in scannedDevices)
    {
        ICPU cpu = device as ICPU;
        if (cpu != null)
        {
            if (cpu.Protected)
            {
                // -----
                // PROFINET      CPU
                //      API
                // -----
                retVal = cpu.SetPassword(new EncryptedString
("CPUPassword"));
            }
        }
    }
}
```

```

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

retVal = device.RefreshStatus();  

if (retVal.Succeeded)  

{  

    //  

    if (cpu.CertificateStore.Count > 0)  

    {  

        // Windows - UI  

        cpu.CertificateStore.ShowDialog();  

        bool bTrust = true;  

        if (bTrust)  

            retVal = cpu.SetTrustCertificateStore  

(TrustCertificateType.Always);  

        else  

            retVal = cpu.SetTrustCertificateStore  

(TrustCertificateType.Never);  

    }  

}  

}  

//-----  

//----- API  

//-----  

#endregion

```

示例：使用 Insert Device 初始化设备

```

using Siemens.Automation.AutomationTool.API;  

#region  

//-----  

//-----  

//-----  

Network myNetwork = new Network();  

List<String> interfaces = new List<String>();  

Result retVal = myNetwork.QueryNetworkInterfaceCards  

(out interfaces);  

if (retVal.Succeeded)  

{  

    for (Int32 index = 0; index < interfaces.Count; index++)  

    {  

        String strInterfaceName = interfaces[index];  

        // -----  

        // PROFINET  

        // NIC  

        // NIC  

        // -----  

        if (strInterfaceName == "myNIC.Auto.1")  

        {  

            retVal = myNetwork.SetCurrentNetworkInterface  

(strInterfaceName);  

            if (retVal.Succeeded)  

            {  

                break;
            }
        }
    }
}
//-----  

//-----  


```

```
//-----
IProfinetDeviceCollection insertedDevices = Network.
GetEmptyCollection();
IProfinetDevice insertedDevice = null;
IScanErrorCollection
insertErrorCollection = insertedDevices.InsertDeviceByIP(0xC0A80001,
out insertedDevice); // 192.168.0.1
if (RetVal.Succeeded)
{
    foreach (IProfinetDevice device in insertedDevices)
    {
        ICPUs cpu = device as ICPUs;
        if (cpu != null)
        {
            if (cpu.Protected)
            {
                //-----
                // PROFINET      CPU
                //
                //          API
                //-----
                retVal = cpu.SetPassword(new EncryptedString
("CPUPassword"));
            }
        }
        //-----
        //
        //-----
        retVal = device.RefreshStatus();
        //
        if (cpu.TLSTrustRequired)
        {
            //           -      UI
            cpu.CertificateStore.ShowDialog();
            bool bTrust = true;
            if (bTrust)
                retVal = cpu.SetTrustCertificateStore
(TrustCertificateType.Always);
            else
                retVal = cpu.SetTrustCertificateStore
(TrustCertificateType.Never);
        }
    }
    //-----          API
    //-----

/*
 *          */
/*
 *          */
/*
 *          */
#endifregion
```

4.1.1 架构概述

API 提供类、接口和方法以支持与 SIMATIC 设备的 PROFINET 网络进行通信。

说明

在应用程序中仅限使用本用户指南中描述的 API 类、接口和方法

除了用于应用程序编程的公共方法之外，API 还包含用于支持 SIMATIC Automation Tool 用户界面的公共方法。开发应用程序时，仅限使用本用户指南中描述的类、接口和方法。本用户指南中未描述的其他 API 公共方法均保留在 SIMATIC Automation Tool 用户界面专用。

网络

.NET 类 Network (页 63) 表示整个 PROFINET 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。Network 类用于搜索可用的网络接口卡以及选择连接到您的 PROFINET 网络的网络接口。

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 63-64)
- QueryNetworkInterfaceCards 方法 (页 64)
- SetCurrentNetworkInterface 方法 (页 64)
- CurrentNetworkInterface 特性 (页 65)
- ScanNetworkDevices 方法 (页 65)
- SetCommunicationsTimeout 方法 (页 66)
- GetCommunicationsTimeout 方法 (页 66)
- GetEmptyCollection 方法 (页 67)

设备

PROFINET 网络中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 (页 79) – PROFINET 网络上所有可以访问的 PROFINET 设备都可以由该接口表示，因为所有设备都源于此类。

ICPU (页 101) – 该接口表示直接连接到网络的 S7-1X00 CPU。CPU 支持特定功能。

ICPUCclassic (页 139) – 该接口表示直接连接到网络的 S7-300 和 S7-400 型经典 CPU。

IHMI (页 140) – 该接口表示直接连接到网络的 SIMATIC HMI。HMI 支持特定功能。

IBaseDevice (页 58) – 该接口表示未直接连接到 PROFINET 网络，但可通过其它设备访问的设备。例如，连接到网络上 CPU 的 PROFIBUS 从站表示为 IBaseDevice。

IModule (页 100) – 该接口用于表示插入到 CPU、PROFINET 设备或 PROFIBUS 站的各 I/O 模块。

IHardware (页 57) – 表示所有其它接口的基础类。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。

IScalance (页 147) – 此接口表示 SCALANCE 设备。

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 (页 67) – 网络上可直接访问的所有设备的集合。

IModuleCollection (页 59) – 可表示插入 CPU 或 IM 的模块的集合。

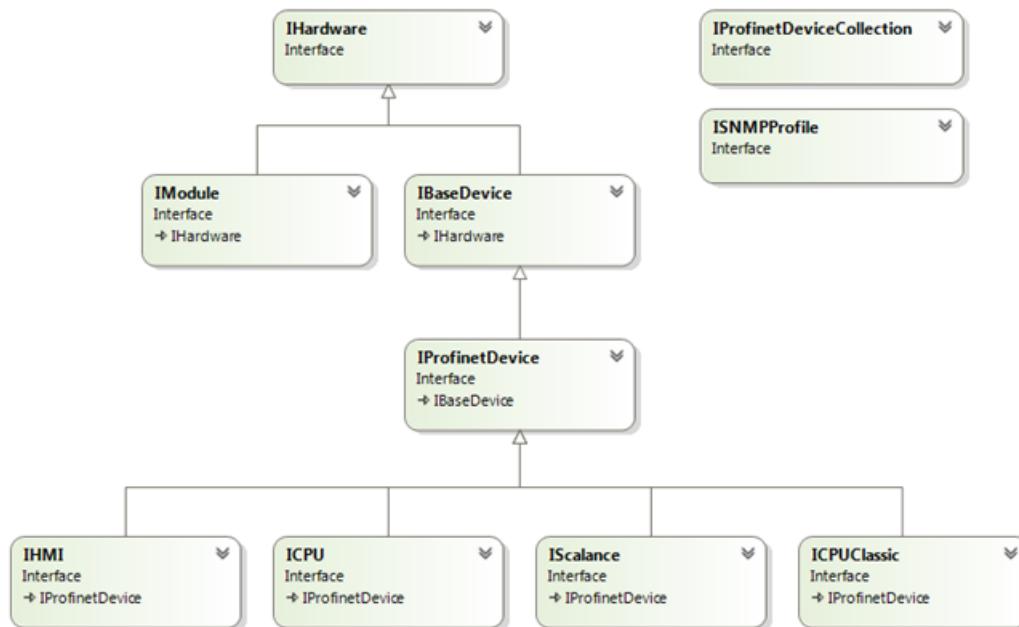
IHardwareCollection (页 58) – 此集合表示 CPU 及其所有模块。

IScanErrorCollection (页 59) – 此集合表示执行网络扫描操作时从所有设备返回的错误集。

设备类别、接口和方法：

- IProfinetDeviceCollection 类别 ([页 67](#))
- IProfinetDevice 接口 ([页 79](#))
- ICPU 接口 ([页 101](#))
- IHMI 接口 ([页 141](#))

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 PROFINET 网络和用于表示各网络组件的 SIMATIC Automation Tool API 类。

4.1.2 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。这些文件均位于所安装 SDK 的 Bin 目录中。

使用该 API 时，这些文件必须位于开发自定义应用程序的同一目录中。

编程环境

西门子使用 Microsoft Visual Studio 2022 以及 .NET 6.0 Runtime 开发了 API。

解决方案中的引用

要在应用程序中包含 API，请在解决方案中添加 **SIMATICAutomationToolAPI.dll** 作为引用。

所需的“using”语句

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;
```

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) 或全局 Using 文件中使用正确的 using 语句。为简单起见，本文档中的各个代码示例不包含 using 语句。

4.2 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

与之前的版本兼容

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。

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。

注意使用 CPU 操作方法的 CPU 密码访问级别 ([页 171](#))，特别是对于故障安全 CPU 的操作 ([页 41](#))。

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。

4.3 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。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。

4.4 为故障安全设备和安全相关操作设计用户界面应用程序

4.4.1 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）”保护等级的 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。

如果自定义应用程序对 F-CPU 执行了安全相关操作，应通过 TÜV SÜD 或同等资质验证机构对应用程序加以认证。Siemens API 经过认证并不意味着您的应用程序会自动得到认证。

API 为您的应用程序提供了一系列的保护措施。您的应用程序也应遵循推荐的一组 F-CPU 用户界面使用指导 ([页 42](#))。

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
- 对受保护的 F-CPU 执行任何安全相关操作都需要使用安全密码 ([页 171](#))
- 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 49](#)) 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

4.4.2 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。软件开发人员应承担所有责任。

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

西门子建议您在对 F-CPU 执行安全相关操作前提供两步操作过程。

1. 在继续执行第二步之前，用户应采取明确操作以选择 F-CPU，并选择要执行的操作。
2. 软件应向用户显示一个提示，提示用户针对安全相关操作的西门子定义的确认消息 [\(页 112\)](#) 之一。在代码调用针对 F-CPU 执行的安全相关操作之前，用户必须确认其希望继续操作。

使用 API 开发用户界面应用程序时，应确定操作是否属于以下针对 F-CPU 的安全相关操作之一：

- 程序更新 [\(页 120\)](#)
- 从备份文件恢复设备 [\(页 124\)](#)
- 复位为出厂默认值 [\(页 123\)](#)
- 格式化存储卡 [\(页 116\)](#)

对于安全相关操作，向用户提供确认提示。使用 DetermineConfirmationMessage API 方法 [\(页 112\)](#) 确定要显示的确认消息类型。提供额外的确认方式可防止用户不小心执行意外的安全相关操作。

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。
- 需要输入安全密码 [\(页 171\)](#) 才能执行安全相关操作。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。
- 如上所述，为用户提供确认提示以确认对 F-CPU 执行的安全相关操作。
- 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。如果您未正确使用 API，API 会因检测到严重内部错误而引发异常。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。如果操作成功，向用户显示一条消息。如果操作失败，显示一条错误消息。
- 在应用程序中使用汉明码 [\(页 49\)](#) 实现布尔状态。
- 如果在设计 Windows GUI 应用程序，请使用黄色的颜色标记 [\(页 44\)](#) 指示故障安全设备、安全程序、安全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提示，以重新选择故障安全设备并确认以下操作：

- 将使用安全密码启动对标准程序执行的操作
- 将删除现有安全程序
- 将使用其它安全程序更新现有安全程序
- 现有安全程序将替换为标准程序
- 安全程序将首次加载

安全程序更新后，API 自动验证 CPU 中更新的程序的 F 签名。检查所有函数返回值。

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

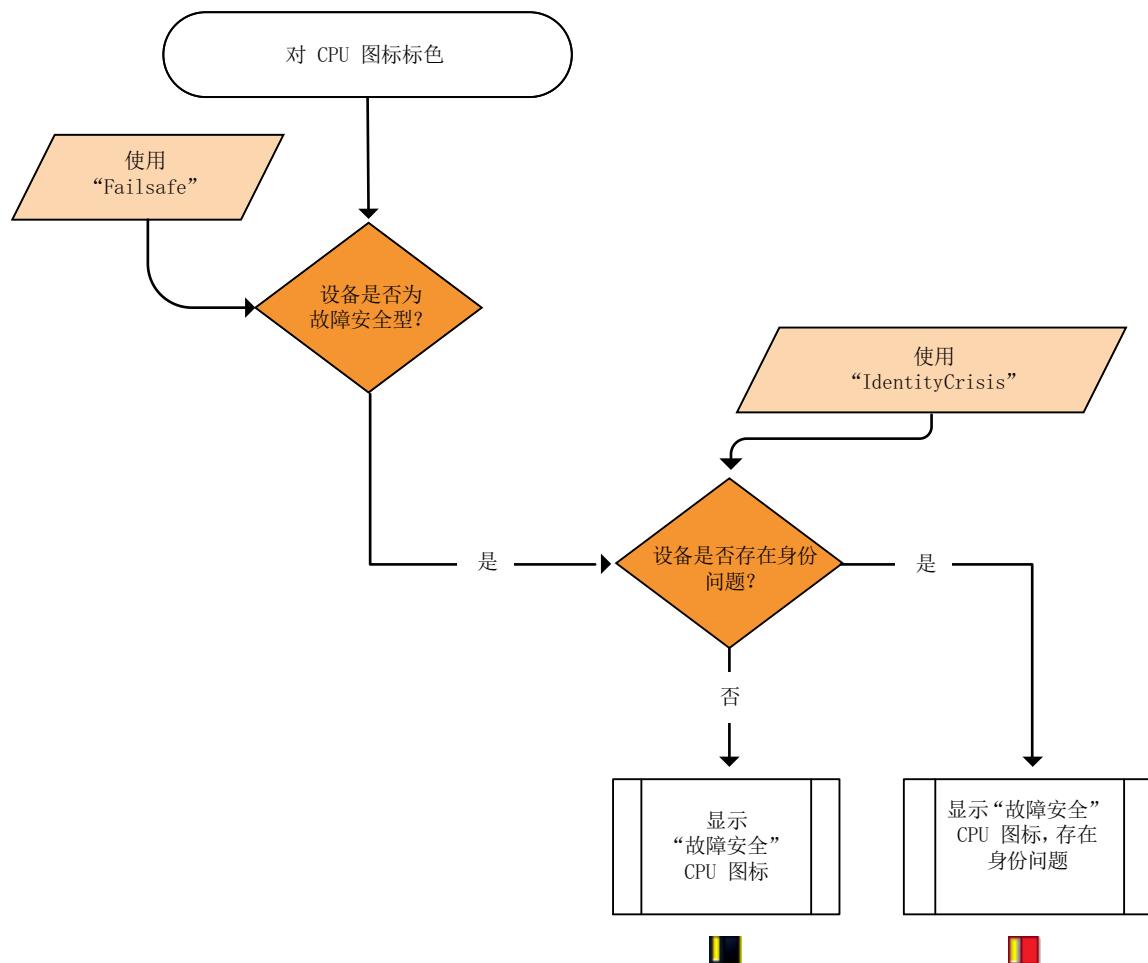
获取用户界面应用程序的认证

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。

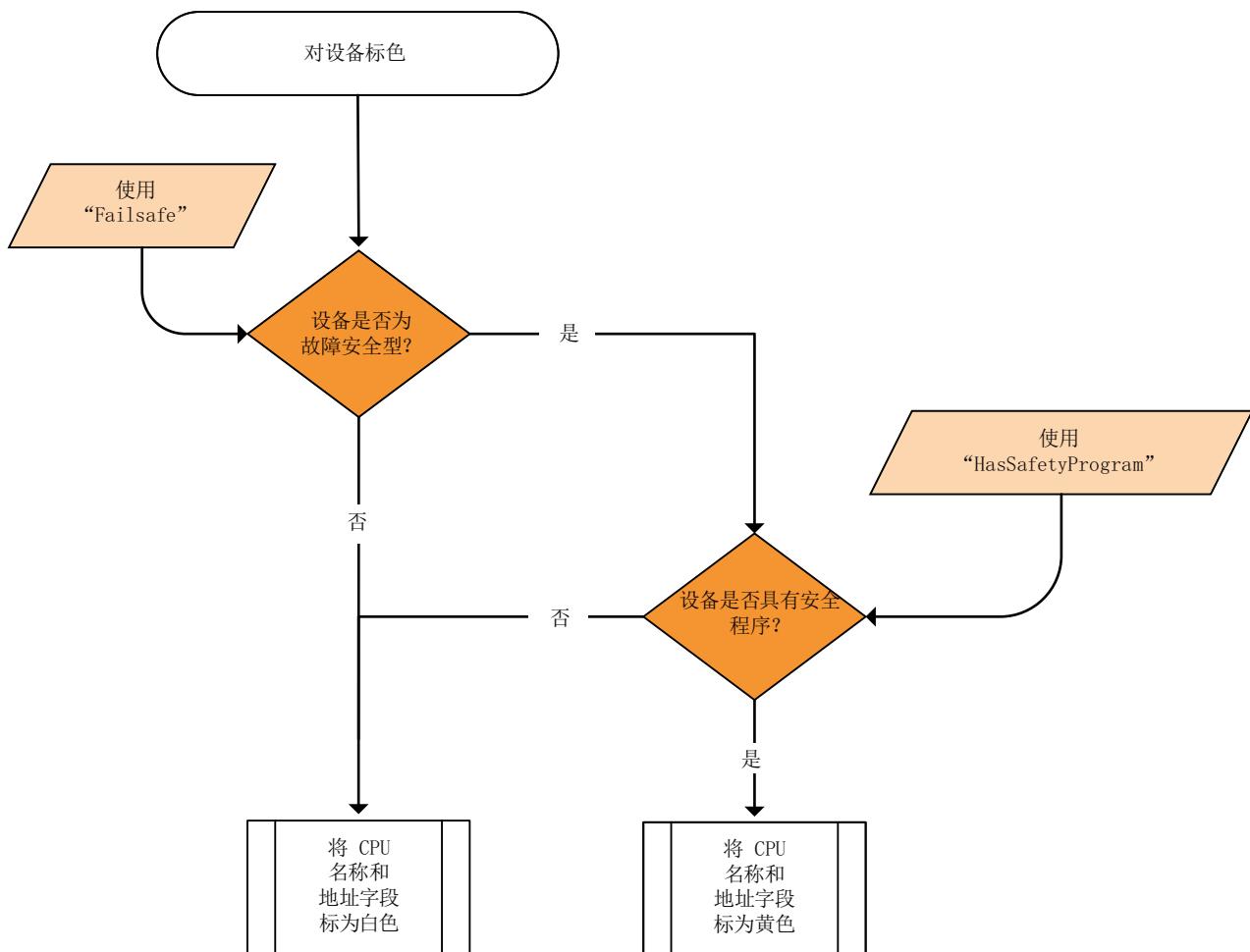
4.4.3 用户界面中的颜色编码安全域

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。

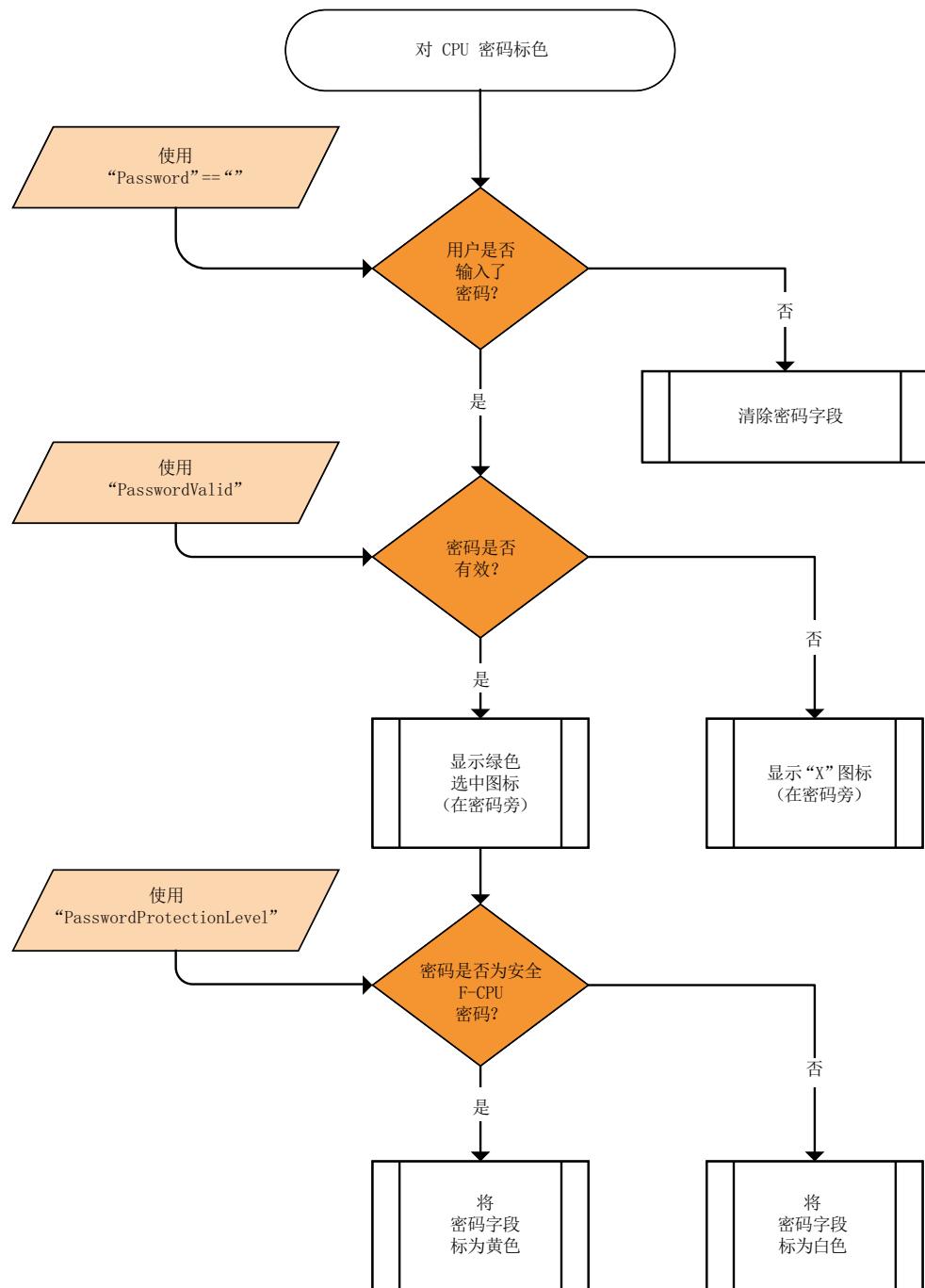
4.4.3.1 对 CPU 设备图标进行颜色编码



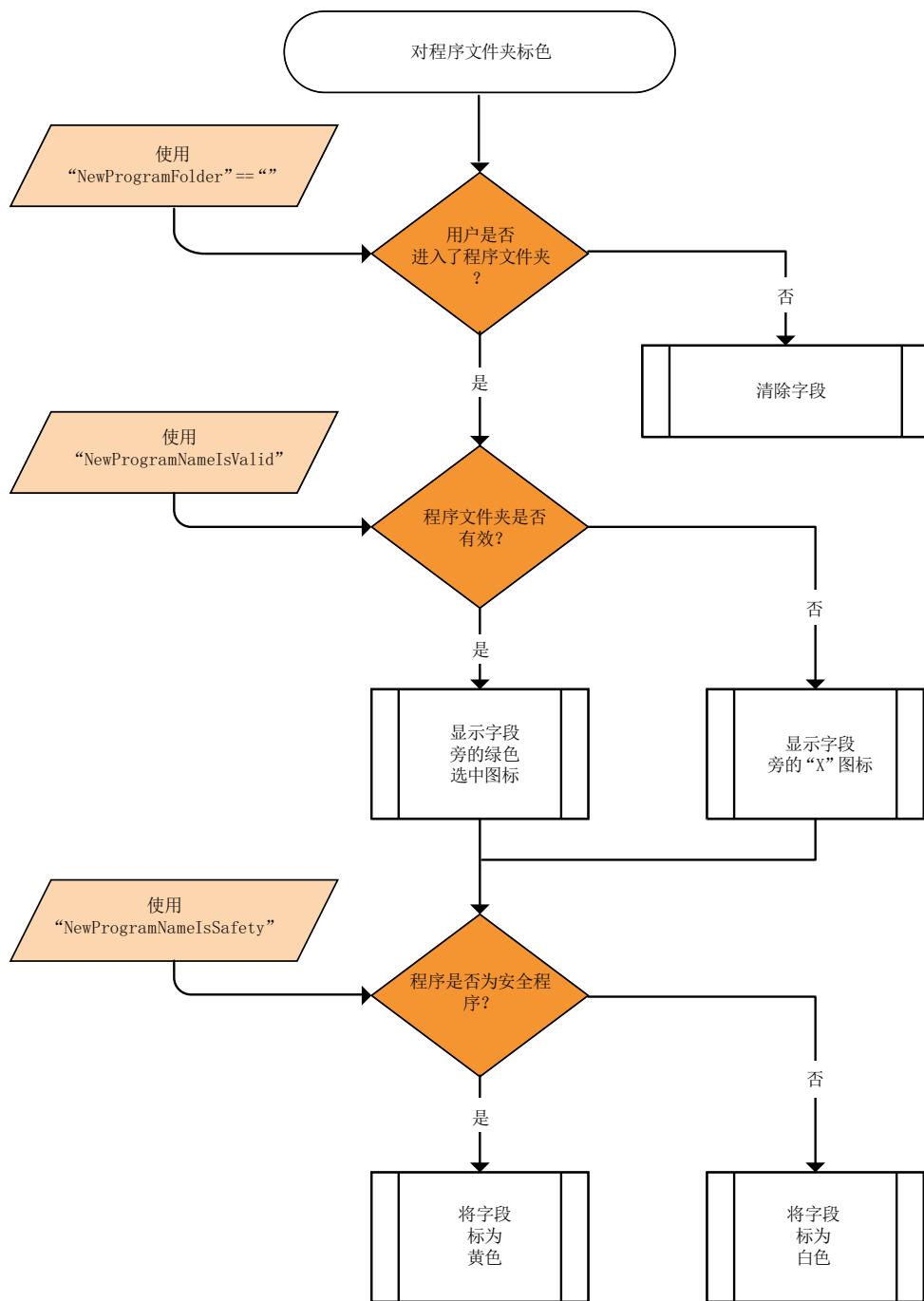
4.4.3.2 对设备数据进行颜色编码



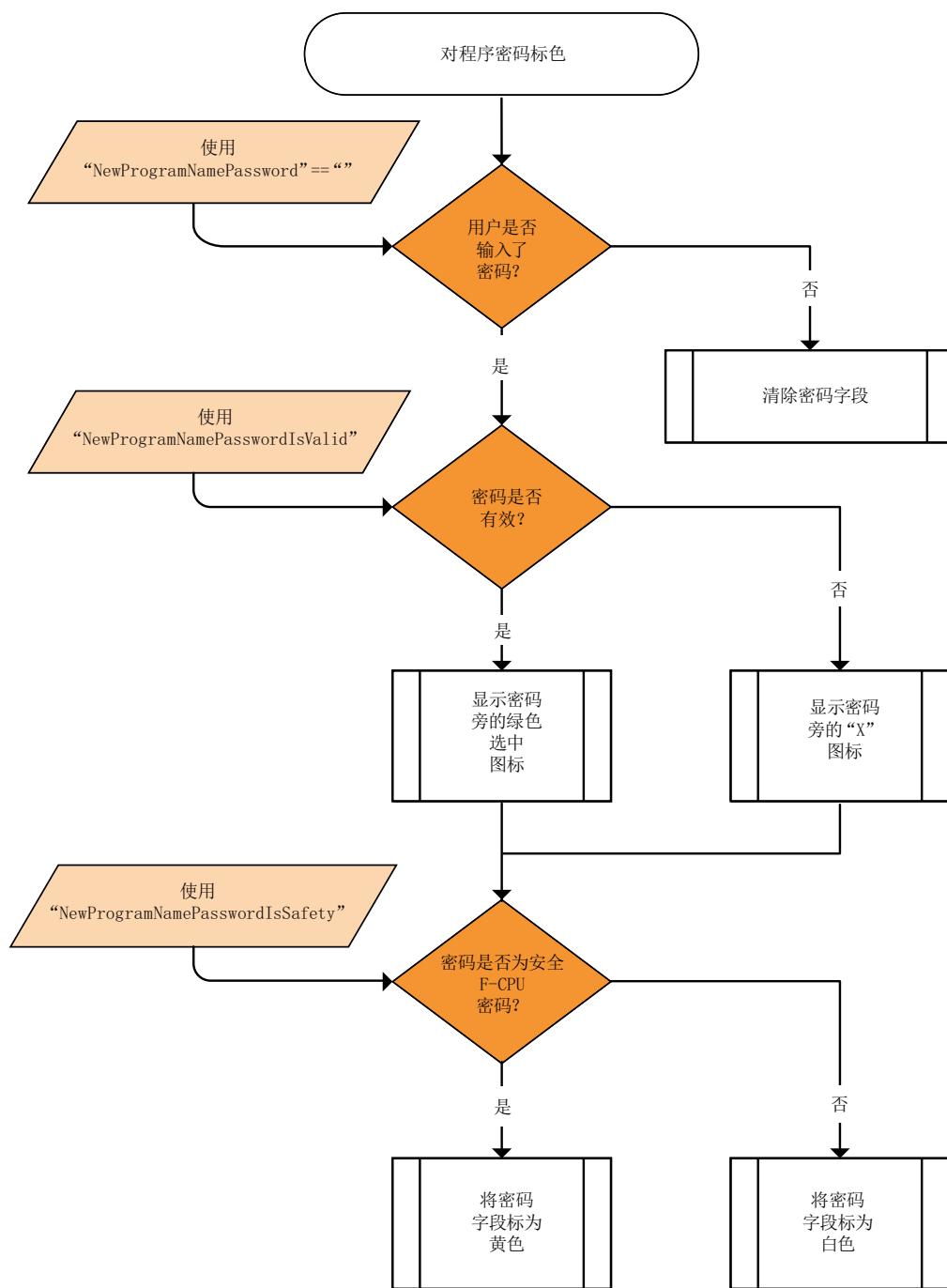
4.4.3.3 对 CPU 密码进行颜色编码



4.4.3.4 对程序文件夹进行颜色编码



4.4.3.5 对程序密码进行颜色编码



4.4.4 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。

4.5 公共支持类别

4.5.1 EncryptedString 类

很多针对 ICPU 接口的 API 操作都需要与受保护的 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。CPU 接受加密格式的密码。为了完成密码加密，API 提供 `EncryptedString` 类。

构造函数	说明
<code>EncryptedString()</code>	空加密字符串
<code>EncryptedString(string strText)</code>	加密字符串

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

方法名称	返回类型	说明
<code>ToString()</code>	<code>string</code>	加密密码的十六进制字符串表示法
<code>Clear()</code>	<code>void</code>	清除加密密码
<code>Copy(EncryptedString password)</code>	<code>void</code>	复制加密密码
<code>GetHash()</code>	<code>byte[]</code>	密码的密码加密哈希数组表示法
<code>WriteToStream(Stream stream)</code>	<code>void</code>	序列化流密码
<code>ReadFromStream(Stream stream)</code>	<code>void</code>	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。然后可以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");
myCPU.SetPassword(pwd);
```

`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。

注意

不适用于数据加密

`EncryptedString` 提供有限的运行系统保护，且不得用于数据加密。

4.5.2 Result 类

`Result` 类提供指定 API 操作是否成功的相关信息。检查 API 操作所返回的 `Result` 对象，以确定操作是否成功。

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

属性名称	返回类型	说明
<code>Warnings</code>	<code>ErrorCode[] {get;}</code>	返回错误代码组中的所有警告
<code>Error</code>	<code>ErrorCode {get;}</code>	返回错误代码
<code>HasWarnings</code>	<code>bool {get;}</code>	存在警告时为真
<code>Failed</code>	<code>bool {get;}</code>	结果失败时为真
<code>Succeeded</code>	<code>bool {get;}</code>	结果成功时为真
<code>OMSCode</code>	<code>bool {get;}{set;}</code>	获取并设置 OMS 代码

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

示例：检查操作是否成功

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。要检查特定错误，请使用 `ErrorCode` 属性 (页 162)。如果设备返回了设备特定的错误，则返回的 `ErrorCode` 是 `DeviceDefinedError`。方法 `GetErrorDescription` 提供设备特定的错误字符串。

要检查指定操作是否成功，可检查 `Succeeded` 属性。

请参见下例中使用的属性：

```
//-----
//    "      API"
//-----
IProfinetDevice dev = scannedDevices[0];
```

```

ICPU myCPU = dev as ICPU;
if (myCPU != null)
{
    //-----
    //      CPU
    //      ICPU
    //      "      "true"
    //-----
    dev.Selected = true;
    Result retVal = dev.RefreshStatus();
    if (retVal.Succeeded)
    {
        //-----
        //      ....
        //-----
    }
    /* In other cases it may be helpful to have more information
    about the failure.*/
    /* To inspect the specific error, use the ErrorCode property, as
    follows: */
    dev.Selected = true;
    retVal = myCPU.RefreshStatus();
    if (retVal.Succeeded)
    {
        //-----
        //      ....
        //-----
    }
    else
    {
        //-----
        //      ....
        //-----
        switch (retVal.Error)
        {
            case ErrorCode.AccessDenied:
            break;
            case ErrorCode.TooManySessions:
            break;
        }
    }
}
}

```

示例：获取特定语言的错误描述

Result 类还提供特定于语言的错误描述。GetErrorMessage 方法将 Language 值 (页 168) 用作参数，以提供指定语言的错误描述。

例如，以下代码会返回德语的错误描述：

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

示例：检查警告

API 具有警告功能，可用于了解已发生的问题。例如，如果 ProgramUpdate 结束时在设备上执行刷新，可能会生成与主调用函数没有直接关系的警告。可通过 Result 类访问这些警告，如下所示：

```

if (retVal.HasWarnings)
{
    foreach (ErrorCode warning in retVal.Warnings)
}

```

```

{
    //-----
    //
    //-----
}
}

```

4.5.3 DiagnosticsItem 类

诊断项包含单个事件的诊断信息。可以通过 GetDiagnosticsBuffer 方法 ([页 119](#)) 从 CPU 读取诊断缓冲区。

构造函数	描述
DiagnosticsItem()	创建默认诊断项

属性名称	返回类型	描述
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

4.5.4 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 \(页 79\)](#)”一章。

构造函数	描述
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	描述
Type	DataChangedType	事件类型

DataChangedEventArgs 类用于以下事件处理程序：

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

4.5.5 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“[IProfinetDevice 接口 \(页 79\)](#)”一章。

构造函数	描述
ProgressChangedEventArgs(ProgressAction action, int index, int count, uint hardwareID)	用于创建和默认进度更改事件 args 类。
ProgressChangedEventArgs(ProgressAction action, int index, int count, int fileNumber, 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
int fileNumber	文件编号（如果适用）
uint hardwareID	正在处理的项目的 ID

属性名称	返回类型	描述
Action	ProgressAction {get;}	此事件的动作类型
Cancel	bool {get; set;}	设置为真以终止当前操作
Count	int {get;}	最大值
FileNumber	int {get;}	适用的文件编号
ID	uint {get;}	硬件项的硬件 ID
Index	int {get;}	当前值

ProgressChangedEventArgs 类用于以下事件处理程序：

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

4.5.6 ExportProgressEventArgs 类

导出进度更改事件包含关于 API 内已更改的数据的信息。请参阅有关 IProfinetDeviceCollection 类的部分。

构造函数	描述
ExportProgressEventArgs(int WorkItem, int MaxEntries)	用于创建和默认导出进度更改事件 args 类
int WorkItem	正在处理的当前项目的索引
int MaxEntries	待处理的总项目

属性名称	返回类型	描述
Cancel	bool {get; set;}	设置为真以终止当前操作
MaxEntries	int {get;}	最大值

属性名称	返回类型	描述
WorkItem	int {get;}	当前值

ExportProgressEventArgs 类用于以下事件处理程序：

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

4.5.7 FileProgressChangedEventArgs 类

文件进度更改事件包含关于 API 内已更改的文件数据的信息。有关详细信息，请参见 IProfinetDevice 接口 ([页 79](#))一章。

构造函数	说明
FileProgressChangedEventArgs(ProgressAction action, int nIndex, int nCount)	用于创建和默认文件进度更改事件 args 类。
FileProgressChangedEventArgs(ProgressAction action)	用于创建和默认进度更改事件 args 类。
FileProgressChangedEventArgs()	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 nIndex	正在处理的当前项目的索引
int nCount	待处理的总项目

属性名称	返回类型	说明
Action	ProgressAction {get;set;}	此事件的动作类型
Cancel	bool {get;set;}	设置为真以终止当前操作
DoThisForRemainingOperations	bool {get;set;}	为其余操作执行动作
FileCount	uint {get;set;}	文件数
FileIndex	int {get;set;}	当前文件的索引
FilePercentComplete	int {get;set;}	已完成操作的百分比
LocalFile	string {get;set;}	本地文件名
RemoteFile	string {get;set;}	远程文件名
Replace	bool {get;set;}	更换文件

方法名称	返回类型	说明
GetRemoteFileNameOnly()	string	只获取远程文件名

FileProgressChangedEventArgs 类用于以下事件处理程序：

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

4.6 通用支持接口

4.6.1 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

4.6.2 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	描述
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	描述
Exists	bool {get;}	设备上存在此文件夹时为真
Files	List<IRemoteFile> {get;}	此文件夹中的文件列表
FileUpdateAllowed	bool {get;}	文件夹可以添加或替换列表中的文件时为真
FolderType	RemoteFolderType {get;}	文件夹类型（数据日志或配方）
Name	string {get; set;}	文件夹名称
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
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
Selected	bool {get; set;}	已选中文件夹时为真
SelectedCount	int {get;}	选择的文件数

4.6.3 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

4.6.4 IHardware 接口

IHardware 是一个接口, 用于表示设备和模块的基本通用硬件接口。

方法名称	返回类型	说明	
SetFirmwareFile()	Result	设置固件文件以更新该设备或模块	
参数	数据类型	参数类型	
strFile	string	In	固件文件的完整文件名

方法名称	返回类型	说明
ValidateUserData	Result	确认用户数据有效。

方法名称	返回类型	说明	
ValidateFirmwareFile()	Result	确认固件文件有效	
参数	数据类型	参数类型	
strFile	string	In	固件文件的完整文件名
bValid	string	Out	固件文件对此设备有效时返回 True
strVersion	string	Out	返回此文件的固件版本

属性名称	返回类型	说明
ArticleNumber	string {get;}	设备或模块的订货号
Comment	string {get;set;}	各设备和模块的注释
Configured	bool {get;}	此设备或模块已组态时为真
ConfiguredVersion	string {get;}	该设备组态的版本号
Description	string {get;}	设备或模块的订货号说明
Failsafe	bool {get;}	设备或模块为故障安全型时为 True
FirmwareUpdateSupported	bool {get;}	该设备或模块支持固件更新时为真
FirmwareVersion	string {get;}	设备或模块的固件版本
FWUpdateActivationFlag	bool {get;}	固件已下载到设备但尚未激活
HardwareNumber	short {get;}	设备或模块的硬件修订号
ID	uint {get;}	设备或模块的 ID
Name	string {get;}	设备或模块的名称

4.6 通用支持接口

属性名称	返回类型	说明
NewFirmwareFile	string {get;}	固件文件的完整文件路径
NewFirmwareNameErrorCode	Result {get;}	SetFirmwareFile 方法的最后一个错误
NewFirmwareNameIsValidId	bool {get;}	固件文件对此设备或模块有效时为真
NewFirmwareVersion	string {get;}	下拉列表中显示的固件文件的版本
Selected	bool {get; set;}	用于所选状态的外部存储
SerialNumber	string {get;}	设备或模块的序列号
Slot	uint {get;}	设备或模块的插槽编号
SlotName	string {get;}	设备或模块的插槽的名称
StationNumber	uint {get;}	设备或模块的站号
StatusConfiguration	ConfigurationStatus {get;}	已组态硬件与实际硬件之间的当前比较状态
SubSlot	uint {get;}	设备或模块的子插槽编号
Supported	bool {get;}	支持此设备或模块时为真

4.6.5 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	描述
GetHardwareFromID(uint hardwareID)	IHardware	使用 ID 查找设备或模块

属性名称	返回类型	描述
HardwareInDisplayOrder	IHardware	按显示顺序排列的硬件项集合
HardwareInFirmwareOrder	IHardware	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	描述
ProgressChanged	ProgressChangedEventArgs	被调用监视进度
DataChanged	DataChangedEventArgs	API 中的数据发生变化时进行调用
FileProgressChanged	FileProgressChangedEventArgs	当 SAT 启动的文件传输已更新进度以进行报告时调用。

4.6.6 IHardwareCollection 接口

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

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

参数			
hardwareID	uint	in	硬件标识符编号

4.6.7 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
-		

4.6.8 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]	IScanErrorEvent	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：

- 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

4.6 通用支持接口

方法名称	返回类型	说明
Clear()	void	清除所有错误、警告和信息结果。

方法名称	返回类型	说明	
AddDeviceLog()	void	将结果添加到设备日志	
参数			
名称	数据类型	参数类型	说明
result	Result	In	要添加到集合的结果代码
device	IProfinetDevice	In	要添加到集合的设备信息。

方法名称	返回类型	说明	
AddDeviceSuccess()	void	将结果作为成功条目添加到设备日志	
参数			
名称	数据类型	参数类型	说明
result	Result	In	要添加到集合的结果代码
device	IProfinetDevice	In	要添加到集合的设备信息。

方法名称	返回类型	说明	
AddDeviceLogInformation()	void	将结果作为信息条目添加到设备日志。	
参数			
名称	数据类型	参数类型	说明
result	Result	In	要添加到集合的结果代码
device	IProfinetDevice	In	要添加到集合的设备信息。

方法名称	返回类型	说明	
AddDeviceLogWarning()	void	将结果作为警告条目添加到设备日志。	
参数			
名称	数据类型	参数类型	说明
result	Result	In	要添加到集合的结果代码
device	IProfinetDevice	In	要添加到集合的设备信息。

方法名称	返回类型	说明	
AddGeneralLog()	void	将结果作为常规条目添加到设备日志	
参数			
名称	数据类型	参数类型	说明
result	Result	In	要添加到集合的结果代码
device	IProfinetDevice	In	要添加到集合的设备信息。

方法名称	返回类型	说明
AddGeneralLogWarning()	void	将结果作为常规警告条目添加到设备日志

方法名称		返回类型	说明
参数			
名称	数据类型	参数类型	说明
result	Result	In	要添加到集合的结果代码
device	IProfinetDevice	In	要添加到集合的设备信息。

方法名称		返回类型	说明
参数			
名称	数据类型	参数类型	说明
result	Result	In	要添加到集合的结果代码
device	IProfinetDevice	In	要添加到集合的设备信息。

方法名称		返回类型	说明
参数			
名称	数据类型	参数类型	说明
collection	IScanErrorCollection	In	要添加到集合的结果代码

操作员名称		返回类型	说明
参数			
名称	数据类型	参数类型	说明
collection	IScanErrorCollection	In	要添加到集合中的集合

操作员名称		返回类型	说明
参数			
名称	数据类型	参数类型	说明
operator +=	void		将集合添加到集合
collection	Result	In	要添加到集合的结果代码

4.6.9 IScanErrorEvent 类

该类提供调用 ScanNetworkDevices (页 65) 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

4.6 通用支持接口

属性名称	返回类型	说明
Code	ErrorCode {get;}	扫描错误
IP	string {get;}	设备的 IP 地址
MAC	string {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

4.6.10 IDiagnosticBuffer 接口

IDiagnosticBuffer 接口是用于获取诊断缓冲区的接口

方法名称	返回类型	描述	
GetDiagnosticsBuffer()	Result	获取诊断缓冲区	
参数			
Name	Data type	参数类型	
aDiagnosticItems	List<DiagnosticsItem>	out	诊断项的集合
language	Language	in	用于诊断条目的语言

属性名称	返回类型	描述
DiagBufferAllowed	bool {get;}	允许诊断缓冲区采集
ScheduleDeviceDiagnostics	bool {get;set;}	安排获取诊断操作

4.6.11 IResult 接口

IResult 接口支持 IResult 类。

属性名称	返回类型	描述
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	描述
GetErrorDescription(Language language)	string	获取以指定语言描述的当前错误字符串

方法名称	返回类型	描述
GetWarningDescription(Language language)	string[]	获取以指定语言描述的警告字符串

4.6.12 ICertificateStore 接口

ICertificateStore 接口通过保持安全通信中使用的 ICertificate 的方式支持使用 ICertificate 类。

方法名称	返回类型	说明
GetEnumerator()	IEnumerator<ICertificate>	允许在所有证书中进行枚举。示例：允许 foreach 工作。

方法名称	返回类型	说明	
ShowDialog()	void	显示“Windows 证书链”(Windows Certificate Chain) 对话框 Linux 不支持此方法。	
参数			
名称	数据类型	参数类型	说明
handleParent	IntPtr	In (可选)	父 Windows 句柄 - 用于模式对话框 Linux 不支持此方法。

属性名称	返回类型	说明
Count	Int32 {get;}	证书数
[]	ICertificate {get(Int32 nIndex);}	从索引获取证书
ContainsCPUGeneratedCertificate	bool {get;}	如果一个或多个证书是由 CPU 生成的，则为 True。
ContainsCAVerifiedChain	bool {get;}	如果链中的所有证书都是 CA 认证链，则为 True。

4.6.13 ICertificate 接口

ICertificate 接口支持证书。

属性名称	返回类型	描述
Data	Byte[] {get;}	包含证书的 Blob 字节。
CPUGeneratedCertificate	bool {get;}	如果证书是由 CPU 生成的，则为 True。

4.7 网络类

4.7.1 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 类用于搜索可用接口卡以及选择与 PROFINET 网络通信的接口卡。

要与 PROFINET 网络交互，程序应声明 Network 类型的变量，如下所示：

```
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。

4.7.2 QueryNetworkInterfaceCards 方法

返回类型	方法名称
Result	QueryNetworkInterfaceCards

名称	数据类型	参数类型	描述
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

要识别可用网络接口卡，可使用 QueryNetworkInterfaceCards 方法。此方法将输出一个字符串列表。列表中的每一项代表一个用名称标识的可用网络接口。

示例：查询网络接口

要了解查询计算机或编程设备上的网络接口的过程，请参阅 API 入门指南 ([页 34](#)) 中的示例和说明。

4.7.3 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	描述
strInterface	string	In	要使用的网络接口的名称。 这是从 QueryNetworkInterfaceCards (页 64) 方法。

要使用计算机或编程设备上的一个网络接口访问 PROFINET 网络，必须“设置”此接口。SetCurrentNetworkInterface 允许您的应用程序与特定 PROFINET 网络以及该网络上的设备通信。

示例：设置网络接口

要了解设置网络接口的过程，请参阅 API 入门指南 ([页 34](#)) 中的示例和说明。

4.7.4 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：

如果之前调用 `SetCurrentNetworkInterface` ([页 64](#)) 方法时未选择任何网络接口，则该属性将返回一个空字符串。

示例：获取当前网络接口

```
-----  
// API 入门指南 (页 34)  
//  
-----  
#region Getting the current network interface  
string currentInterface = myNetwork.CurrentNetworkInterface;  
  
/* */  
/* */  
/* */  
#endregion
```

4.7.5 ScanNetworkDevices 方法

设置网络接口 ([页 64](#)) 后，可以在 PROFINET 网络上搜索设备。`ScanNetworkDevices` 方法会输出一个 `IProfinetDeviceCollection` ([页 67](#))。该集合中的每一项代表一个直接连接到 PROFINET 网络的设备。这些设备可包括 CPU、HMI 和其它设备。

API 提供过滤设备扫描的功能。过滤器会识别一组 IP 地址或 IP 地址范围，供应用程序查找、初始化和采集设备数据。过滤扫描可显著缩短网络扫描时间，并可为用户提供更多控制功能。

返回类型	方法名称
<code>IScanErrorCollection</code>	<code>ScanNetworkDevices</code>

参数			
名称	数据类型	参数类型	说明
<code>baseDevices</code>	<code>IProfinetDeviceCollection</code>	<code>Out</code>	包含 PROFINET 网络中每台可访问设备的 <code>IProfinetDevice</code> 元素的集合
<code>strFilter</code>	<code>string</code>	<code>In</code> (可选)	可选 IP 地址过滤器字符串可将扫描设备减少为与过滤器匹配的集合。

扫描包含多台设备的网络可能需要几分钟。返回的 `IScanErrorCollection` ([页 59](#)) 中的 `Succeeded` 属性指示扫描是否成功。

示例：扫描网络以查找设备

要了解扫描网络的完整操作过程和需要的信息，请参阅 API 入门指南 ([页 34](#)) 中的示例和说明。

要在“开始使用 API”示例中使用过滤扫描，请将现有 ScanNetworkDevices 调用替换为以下代码：

```
// 192.168.0.1 192.168.0.20
IScanErrorCollection scanResults = myNetwork.ScanNetworkDevices
(out scannedDevices, "192.168.0.[1-20]");
// 192.168.0.2 192.168.0.5 192.168.0.10
IScanErrorCollection scanResults = myNetwork.ScanNetworkDevices
(out scannedDevices, "192.168.0.[2,5,10]");
```

4.7.6 SetCommunicationsTimeout 方法

可以为使用 API 调用的 S7-1200 和 S7-1500 CPU 通信操作设置时间限制。

`SetCommunicationsTimeout` 允许指定介于 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。

返回类型	方法名称
Result	<code>SetCommunicationsTimeout</code>

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

示例：设置通信超时

```
-----  
// API 入门指南 (页 34)  
-----  
#region CPU  
// CPU  
uint timeout = Network.GetCommunicationsTimeout();  
if (timeout > 180) // 3  
{  
    retVal = Network.SetCommunicationsTimeout(180);  
}  
/* */  
/* */  
/* */  
#endregion
```

4.7.7 GetCommunicationsTimeout 方法

使用 `GetCommunicationsTimeout` 方法检索 CPU 网络通信超时值（以秒为单位）。

返回类型	方法名称
uint	<code>GetCommunicationsTimeout</code>

示例：获取通信超时

```

//-----
//      API 入门指南 (页 34)
//-----
//      CPU
uint timeout = Network.GetCommunicationsTimeout();

if (timeout > 180) //set timeout to 3 minutes
{
    retVal = Network.SetCommunicationsTimeout(180);
}

```

4.7.8 GetEmptyCollection 方法

GetEmptyCollection 方法返回一个空的 IProfinetDeviceCollection。您可以将设备插入空集合中。将设备插入空集合中是扫描网络以查找设备的替代方案。

返回类型	方法名称
IProfinetDeviceCollection	GetEmptyCollection

示例：查询网络接口

要了解通过 GetEmptyCollection 方法获取空集合并插入设备的过程，请参阅 API 入门指南 (页 34) 中的示例和说明。

4.7.9 ValidateNetworkInterface 方法

调用 ValidateNetworkInterface 方法确定网络接口是否有效。如果此方法无法找到网络接口，会返回错误“InvalidNetworkInterface”。如果接口卡已移除或处于故障状态，则会导致出现此错误。

方法名称	返回类型	描述	
ValidateNetworkInterface()	Result	验证此网络接口在该系统上是否仍然有效。	
参数			
名称	数据类型	参数类型	描述
nTimeout	string	in	要验证的网络接口

4.8 IProfinetDeviceCollection 类

4.8.1 迭代集合中的项

4.8.1.1 迭代集合中的项

`ScanNetworkDevices` (页 65) 方法输出类型为 `IProfinetDeviceCollection` (页 67) 的对象。此类可用于使用 `foreach` 语法迭代集合中的项，或使用数组语法访问各 `IProfinetDevice` (页 79)。

扫描包含多台设备的网络可能需要几分钟。返回的 `IScanErrorCollection` 中的 `Succeeded` 属性指示扫描是否成功。

示例：循环访问集合中的每台设备

```
//-----
//  API 入门指南 (页 34)
//-----
#region      PROFINET
foreach (IProfinetDevice dev in scannedDevices)
{
    //-----
    // "dev" represents the
    // IProfinetDeviceCollection
    // 中的下一项-----
}
#endregion
```

示例：将扫描的设备作为数组进行循环访问

```
//-----
//  API 入门指南 (页 34)
//-----
#region
for (int deviceIdx = 0; deviceIdx < scannedDevices.Count;
deviceIdx++)
{
    //-----
    // IProfinetDevice
    // "dev"
    //-----
    IProfinetDevice dev = scannedDevices[deviceIdx];
}
#endregion
```

4.8.1.2 FindDevicesBySerialNumber 方法

使用 FindDevicesBySerialNumber 方法查找集合中序列号相同的所有设备。

方法名称	返回类型	说明	
FindDevicesBySerialNumber ()	IProfinetDevice []	查找集合中序列号相同的设备	
参数			
名称	数据类型	参数类型	描述
strArticleNumber	string	In	您要查找的设备的订货号。
strSerialNumber	string	In	您要查找的设备的序列号。
device	IProfinetDevice	Out	返回对已找到设备的引用

4.8.1.3 GetEnumerator 方法

使用 GetEnumerator 方法枚举 IProfinetDeviceCollection 中的所有 IProfinetDevices。

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
-			

4.8.1.4 SortByIP 方法

方法名称	返回类型	描述
SortByIP ()	void	根据 IP 地址以数字顺序对设备集合进行排序。

4.8.1.5 Count 属性

属性名称	返回类型	说明
Count	int {get;}	集合中的 PROFINET 设备数
ConnectedDevices	int {get;}	集合中 DeviceFound 设为 True 的设备数
CPUCount	int {get;}	集合中的 CPU 数
HMICount	int {get;}	集合中的 HMI 数

此属性会返回 IProfinetDeviceCollection 中的 IProfinetDevices 数目计数

4.8.1.6 [] 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)
{
    //-----
    // 
    //-----
    IProfinetDevice device = collection[0];
}
```

4.8.2 过滤集合中的项目

4.8.2.1 集合项

在该集合中，PROFINET 网络上的每台设备都有一个对应的项。该可以包含来自多个不同产品系列的设备（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。这可能有助于过滤集合以使其仅包含某些设备的情况。

4.8.2.2 FilterByDeviceFamily 方法

FilterByDeviceFamily 方法返回一个集合，该集合仅包含指定产品系列的设备。

返回类型	方法名称
List<IProfinetDevice>	FilterByDeviceFamily

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

首先，构建一个或多个设备系列的过滤器。将此过滤器传递给 FilterByDeviceFamily 方法。结果是只包含指定产品系列的设备的 IProfinetDeviceCollection。

说明

传递空的 List<DeviceFamily> 会返回空集合。

示例：按照 CPU 系列过滤扫描的设备

此示例仅从扫描设备中筛选出 S7-1200 和 S7-1500 系列设备。

```
//-----
//  API 入门指南 (页 34)
//
//-----
#region      S7-1200      S7-1500

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

4.8.2.3 FilterOnlyCPUs 方法

API 支持很多只适用于 CPU 的操作。FilterOnlyCPUs 方法可用于对集合进行过滤，使其仅包含已扫描网络上的 CPU。

返回类型	方法名称
List<ICPU>	FilterOnlyCPUs

此方法会返回一个 ICPU 列表。ICPU interface (页 101) 提供用于 CPU 的属性和方法。

示例：仅从扫描设备中过滤出 CPU

此示例仅从扫描设备中过滤出 CPU。

```
//-----
//  API 入门指南 (页 34)
//
//-----
#region          CPU

List<ICPU> cpus = scannedDevices.FilterOnlyCpus();
foreach (ICPU cpu in cpus)
{
    //-----
    //      CPU
    //-----
}
#endregion
```

4.8.3 在集合中查找特定设备

4.8.3.1 FindDeviceByIP 方法

使用 FindDeviceByIP 方法按特定设备的 IP 地址在集合中搜索该设备。

返回类型	方法名称
IProfinetDevice	FindDeviceByIP

4.8 IProfinetDeviceCollection 类

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

示例：查找位于特定 IP 地址的设备

```

//-----
//  API 入门指南 (页 34)
//
//-----
#region      IP
IProfinetDevice dev = scannedDevices.FindDeviceByIP(0xC0A80001); // 192.168.0.1
if (dev != null)
{
    //
}
#endregion

```

4.8.3.2 FindDeviceByMAC 方法

使用 FindDeviceByMAC 方法根据特定设备的 MAC 地址在集合中搜索该设备。

返回类型	方法名称
IProfinetDevice	FindDeviceByMAC

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

示例：查找位于特定 MAC 地址的设备

```

//-----
//  API 入门指南 (页 34)
//
//-----
#region      MAC
IProfinetDevice dev =
scannedDevices.FindDeviceByMAC(0x112233445566);
// MAC      11:22:33:44:55:66
if (dev != null)
{
    //
}
#endregion

```

4.8.4 设备集合的序列化

4.8.4.1 WriteToStream 方法

使用 WriteToStream 方法将集合的内容存储到外部目标，例如文件。

返回类型	方法名称
Result	WriteToStream

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

示例：将集合写入文件

```
-----  
// API 入门指南 (页 34)  
//  
-----  
#region  
FileStream f = File.Create("myDataFile.SAT");  
RetVal = scannedDevices.WriteToStream(f);  
f.Close();  
  
/* */  
/* */  
/* */  
#endregion
```

4.8.4.2 ReadFromStream 方法

ReadFromStream 方法会通过之前创建的序列化文件创建集合。

返回类型	方法名称
Result	ReadFromStream

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

示例：通过文件创建集合

```
-----  
// API 入门指南 (页 34)  
//  
-----  
#region  
IProfinetDeviceCollection devices = Network.GetEmptyCollection();  
FileStream f = File.OpenRead("myDataFile.SAT");  
RetVal = devices.ReadFromStream(f);
```

4.8 IProfinetDeviceCollection 类

```
f.Close();
/*
 */
/*
 */
#endregion
```

4.8.4.3 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	描述
ExportFilePath	string	In	生成的导出文件的目标文件路径

示例：导出设备信息

```
//-----
//  API 入门指南 (页 34)
//
//-----
#region

String exportFilePath = @"c:\export\DeviceInformation.csv";
retVal = scannedDevices.ExportDeviceInformation(exportFilePath);
/*
 */
/*
 */
#endregion
```

4.8.4.4 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
IScanErrorCollection	ExportDeviceDiagnostics

参数			
名称	数据类型	参数类型	描述
strPath	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

对于集合中的每个 CPU，返回的 `IScanErrorCollection` (页 59) 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，则该方法会为 CPU 返回错误。错误导致为 CPU 存储一个空的数据条目。

示例：导出设备诊断

```

//-----
//  API 入门指南 (页 34)
//
//-----
#region

//-----
//          CPU
//-----
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;
}

//      API
String exportFilePath = @"c:\export\Device\DeviceDiagnostics.csv";
IScanErrorCollection exportResult =
    scannedDevices.ExportDeviceDiagnostics(exportFilePath,
    Language.English, TimeFormat.Local);
/*
 *          */
/*
 *          */
/*
 *          */
#endregion

```

示例：使用 `ProgressChanged` 事件监控设备诊断的导出进程

API 提供 `ProgressChanged` (页 99) 事件以监视耗时过长的方法的进度。

`ExportDeviceDiagnostics` 是一种耗时可能过长的方法。

要使用 `ProgressChanged` 事件来监视 `ExportDeviceDiagnostics` 的进度，可为该事件附加一个事件处理程序。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` 事件监视设备诊断信息的导出进度。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。

```

//-----
//  API 入门指南 (页 34)
//
//-----
#region

{
    //
    scannedDevices.ProgressChanged += Export_ProgressChanged;
    IScanErrorCollection ExportDeviceDiagnosticsErrors =

```

```

scannedDevices.ExportDeviceDiagnostics(@"C:
\MyDocuments\DeviceDiagnostics.csv", Language.English,
TimeFormat.UTC);
//
scannedDevices.ProgressChanged -= Export_ProgressChanged;
}
void Export_ProgressChanged(object sender, ExportProgressEventArgs e)
{
    String strProgress = String.Format("Processing {0} of {1}",
e.WorkItem, e.MaxEntries);
    //
    e.Cancel = false;
}
/*
*/
/*
*/
#endregion

```

4.8.5 手动将项目添加到集合中

API 提供了以下用于将设备插入集合的方法：

- [InsertDeviceByIP 方法 \(页 76\)](#)
- [InsertDeviceByMAC 方法 \(页 77\)](#)

根据 PROFINET 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，但可以通过 IP 地址添加。此外，您也可以选择将应用程序设计为插入设备而不是通过网络扫描发现设备。对于 IP 地址路由器或 NAT 路由器后面的设备，只能将它们插入到集合中。网络扫描无法发现 IP 地址路由器或 NAT 路由器后面的设备。请参阅路由器后面的设备和打开端口的要求 ([页 175](#)) 的相关主题。

4.8.5.1 InsertDeviceByIP 方法

[InsertDeviceByIP 方法](#) 将设备添加至 [IProfinetDeviceCollection \(页 67\)](#)。该方法将具有特定 IP 地址的设备插入指定索引位置。

返回类型	方法名称
IScanErrorCollect- ion	InsertDeviceByIP

参数			
名称	数据类型	参数类型	说明
ip	uint	In	要添加到集合的设备的 IP 地址
routerIP	uint	In (可选)	当设备在路由器后面时，这是适用的路由器 IP 地址
Password	EncryptedString	In (可选)	可选 CPU 密码。使用 NAT 路由器时需要使用。

参数			
名称	数据类型	参数类型	说明
insertedDevice	IProfinetDevice	Out	如果此设备已插入，则将是有效引用。否则返回的这个值将为 Null。

示例：通过 IP 地址插入设备

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：

```
//-----
// "      API" (页 34)
// -----
uint missingDeviceIPAddress = 0xC0A80001; // 192.168.0.1
IProfinetDevice insertedDevice;
IScanErrorCollection insertErrorCollection =
scannedDevices.InsertDeviceByIP(missingDeviceIPAddress, out
insertedDevice);
```

4.8.5.2 InsertDeviceByMAC 方法

InsertDeviceByMAC 方法将设备添加至 IProfinetDeviceCollection (页 67)。该方法将具有特定 MAC 地址的设备插入指定索引位置。

返回类型	方法名称
IScanErrorCollecti- on	InsertDeviceByMAC

参数			
名称	数据类型	参数类型	说明
mac	ulong	In	要添加到集合的设备的 MAC 地址
insertedDevi- ce	IProfinetDevice	Out	如果此设备已插入，则将是有效引用。否则返回的这个值将为 Null。

示例：通过 MAC 地址插入设备

```
//-----
// API 入门指南 (页 34)
// -----
UInt64 targetMAC = 0x112233445566; //           "11:22:33:44:55:66"
IProfinetDevice insertedDevice;
IScanErrorCollection insertErrorCollection =
scannedDevices.InsertDeviceByMAC(targetMAC, out insertedDevice);
```

4.8.6 从集合中复制数据

4.8.6.1 CopyUserData 方法

使用 CopyUserData 方法将用户输入的数据从一个 IProfinetDeviceCollection 复制到另一个集合中。您可以使用此方法让用户免于再次输入数据。

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	现有的包含用户数据的集合

示例：将一次扫描获得的用户数据复制到另一次扫描

```

//-----
//    API 入门指南 (页 34)
//-----
#region
IProfinetDeviceCollection rescannedDevices;
IScanErrorCollection scanResult = myNetwork.ScanNetworkDevices(out
rescannedDevices);
retVal = rescannedDevices.CopyUserData(scannedDevices);
/*           */
/*           */
/*           */
#endregion

```

4.8.7 从集合中移除设备

4.8.7.1 Clear 方法

Clear 方法清除已扫描设备的内容。

返回类型	方法名称
void	Clear

4.8.7.2 Remove 方法

Remove 方法从集合中删除特定设备。

返回类型	方法名称
void	Remove

参数			
名称	数据类型	参数类型	描述
device	IProfinetDevice	In	集合中待移除的设备。

4.9 IProfinetDevice 接口

4.9.1 IProfinetDevice 属性

IProfinetDeviceCollection (页 67) 集合中的每一项都由 IProfinetDevice 接口表示。

可以通过该接口访问数据并对直接连接至 PROFINET 网络的所有设备进行操作。

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。为确保属性能够返回当前信息，在读取属性前应首先对设备调用 RefreshStatus (页 92) 方法。

属性名称	返回类型	描述
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
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？
DuplicateNewIP	bool {get;}	新提供的 IP 地址重复
DuplicateProfinetName	bool {get;}	该设备的 PROFINET 名称是否重复？
DuplicateNewProfinetName	bool {get;}	新提供的 PROFINET 名称重复
Failsafe	bool {get;}	根据其 ArticleNumber 确定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 (页 162) 的描述。

4.9 IProfinetDevice 接口

属性名称	返回类型	描述
FamilyName	string {get;}	此设备的系列名称是什么？
FirmwareUpdateAllowed	bool {get;}	此设备是否可以进行固件更新？
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。
IdentifyAllowed	bool {get;}	该设备当前是否启用并允许识别？
IdentifySupported	bool {get;}	该设备是否支持识别？
Initialized	bool {get;}	True : 设备处于初始化后的状态。
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 0xCOA80001 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。
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 (页 100) 集合。
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	描述
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。
NewSubnetMask	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。
NotAccessiblewithDCP	string{get;}	True : 该设备当前无法进行 DCP 访问。它在路由器之后
ProfinetConvertedName	string {get;}	设备的转换后 PROFINET 名称
ProfinetName	string {get;}	设备的 PROFINET 名称
ResetCommunicationsPara-metersAllowed	bool {get;}	该设备当前是否启用并允许“复位通信参数”？
ResetCommunicationsPara-metersSupported	bool {get;}	该设备是否支持“复位通信参数”？
ResetToFactoryAllowed	bool {get;}	设备是否支持 ResetToFactory？
ResetToFactorySupported	bool {get;}	该设备是否支持恢复出厂设置？
RouterIP	uint {get;}	路由器的 IP 地址，在使用时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 0xCOA80001 等同于更常见的字符串表示形式“192.168.0.1”。 如果未使用任何路由器，则此属性为 0。 注：SIMATIC Automation Tool 仅支持 IPv4 地址。
RouterIPString	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”）
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
SetIPAllowed	bool {get;}	该设备当前是否启用并允许“设置 IP 地址”？
SetIPSupported	bool {get;}	该设备是否支持“设置 IP 地址”？
SetProfinetNameAllowed	bool {get;}	该设备当前是否启用并允许“设置 PROFINET 名称”？

4.9 IProfinetDevice 接口

属性名称	返回类型	描述
SetProfinetNameSupported	bool {get;}	该设备是否支持“设置 PROFINET 名称”？
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。
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 0xFFFFFFF00 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）

4.9.2 IProfinetDevice 方法

4.9.2.1 FirmwareUpdate 方法

使用 IHardware 方法 SetFirmwareFile (页 57) 和 IProfinetDevice 方法 FirmwareUpdate 通过 CPU 接口更新以下硬件的固件：

- CPU
- 本地 I/O 模块
- 分布式 I/O
- 显示器、信号板、信号模块、通信模块等

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。
type	FirmwareUpdateType (页 167)	In (可选)	可选输入参数，指定要执行的固件更新操作的类型

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。
-

限制

API 仅支持通过 CPU 网络接口更新固件。不能通过 CM 或 CP 接口更新固件。

预防措施

更新固件将 CPU 置于停止模式。

直接连接至网络接口的 CPU 的固件更新

要更新 CPU 固件，应用程序必须执行以下步骤：

1. 设置 CPU 的 Selected (页 79) 属性。
2. 通过适用于此 CPU 的·UPD 文件调用 SetFirmwareFile (页 57) 方法。
3. 通过 CPU ID 为 CPU 调用 FirmwareUpdate 方法。
4. 清除 CPU 的 Selected (页 79) 属性。

直接连接的 CPU 本地模块、显示器、信号板和信号模块的固件更新

要更新直接连接的 CPU 的本地模块的固件，应用程序必须执行以下步骤：

1. 设置本地模块的 Selected (页 79) 属性。
2. 通过本地模块的相应·UPD 文件调用 SetFirmwareFile (页 57) 方法。
3. 通过本地模块的 ID 为 CPU 调用 FirmwareUpdate 方法。
4. 清除本地模块的 Selected (页 79) 属性。

说明

为直接连接网络接口的 PROFINET 设备更新固件

如果 PROFINET 设备不是 CPU 并且直接连接至网络接口，您可以沿用相同的过程和代码示例更新这些设备的固件。

访问分布式 I/O 设备

您也可以为组态为 PROFINET 网络上的分布式 I/O 的设备更新固件。必须在 TIA Portal 的设备组态中组态分布式 I/O 设备。必须将组态下载到 CPU，以使 CPU 能够将固件更新转发给相应的设备。

限制

API 仅支持通过 PROFINET 进行固件更新。API 不支持 PROFIBUS 和 AS-i 分布式 I/O 网络选项。

通过 CPU 接口更新 ET200 IM（分布式 I/O）的固件

要更新固件，应用程序必须执行以下步骤：

1. 设置 PROFINET 设备的 Selected (页 79) 属性。
2. 通过此 PROFINET 设备的相应·UPD 文件调用 SetFirmwareFile (页 57) 方法。
3. 通过 PROFINET 设备的 ID 为 CPU 调用 FirmwareUpdate 方法。
4. 清除 PROFINET 设备的 Selected (页 79) 属性。

通过 CPU 接口对 ET200 IM 本地模块（分布式 I/O）进行固件更新

要更新固件，应用程序必须执行以下步骤：

1. 设置 PROFINET 设备本地模块的 Selected (页 79) 属性。
2. 通过此 PROFINET 设备本地模块的相应·UPD 文件调用 SetFirmwareFile (页 57) 方法。
3. 通过 PROFINET 设备本地模块的 ID 为 CPU 调用 FirmwareUpdate 方法。
4. 清除 PROFINET 设备本地模块的 Selected (页 79) 属性。

示例：固件更新

```
-----  
//      API 入门指南 (页 34)  
//  
//-----  
#region Firmware Update  
IProfinetDevice myDevice =  
scannedDevices.FindDeviceByIP(0xC0A80001); // 192.168.0.1  
if (myDevice != null)  
{  
    //-----  
    //      CPU      PROFINET  
    //-----  
    myDevice.Selected = true;  
    retVal = myDevice.SetFirmwareFile("C:\\DeviceFirmwareFile.upd");  
    retVal = myDevice.FirmwareUpdate(myDevice.ID, false);  
    myDevice.Selected = false;  
    //-----  
    //      CPU      PROFINET  
    //-----  
    foreach (IModule module in myDevice.Modules)  
    {  
        //-----  
        //      CPU      PROFINET  
        //-----
```

```

        module.Selected = true;
        retVal =
module.SetFirmwareFile("C:\\LocalModuleFirmwareFile.upd");
        retVal = myDevice.FirmwareUpdate(module.ID, false);
        module.Selected = false;
    }
//-----
//      CPU          I/O
//-----
ICPU myCPU = myDevice as ICPU;
if (myCPU != null)
{
    foreach (IRemoteInterface interFace in myCPU.RemoteInterfaces)
    {
        foreach (IBaseDevice remoteDevice in interFace.Devices)
        {
            //-----
//              I/O IM
//-----
            remoteDevice.Selected = true;
            retVal =
remoteDevice.SetFirmwareFile("C:\\RemoteDeviceFirmwareFile.upd");
            retVal = myCPU.FirmwareUpdate(remoteDevice.ID, false);
            remoteDevice.Selected = false;
        }
//-----
//              /O IM
//-----
        foreach (IModule module in remoteDevice.Modules)
        {
            module.Selected = true;
            retVal =
module.SetFirmwareFile("C:\\RemoteModuleFirmwareFile.upd");
            retVal = myCPU.FirmwareUpdate(module.ID, false);
            module.Selected = false;
        }
    }
}
/*
 */
/*
 */
/*
 */
#endifregion

```

4.9.2.2 FirmwareActivate 方法

FirmwareActivate 方法将激活设备上指定硬件项 (hardwareID) 的已下载固件更新文件。 hardwareID 可指定设备本身或同一机架中的模块。

返回类型	方法名称
Result	FirmwareActivate

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

固件更新的类型

API 支持以下固件更新类型：

- 包括文件下载与激活的典型固件更新
- 两步固件更新，不包括激活。文件下载在 RUN 模式下进行。
- 两步固件更新，不包括激活。文件下载在 STOP 模式下进行。

如果设备不是 CPU，则没有操作模式（运行/停止）。第二种和第三种类型相同。

包括下载和激活的固件更新

典型固件更新将固件更新文件下载到 CPU 或设备，并立即激活新固件。下载固件更新文件之前，[FirmwareUpdate \(页 82\)](#) 方法将所有 CPU 置于 STOP 模式。要查看说明和示例代码，请参阅 [FirmwareUpdate \(页 82\)](#) 主题。

两步固件更新（在 RUN 模式下载，不包括激活）

可以选择将固件下载到设备上，不立即激活固件。可以通过调用 [FirmwareActivate](#) 方法选择稍后在自选时间激活。两步指的是下载步骤 + 激活步骤。您可以选择将固件更新文件下载到所有支持此功能的设备上，同时这些设备控制着您的操作过程，随后进入 RUN 模式。在所有设备都收到固件更新文件后，您可以选择同时激活所有设备。激活过程将 CPU 置于 STOP 模式

两步固件更新（在停止模式下载，不包括激活）

可以选择将固件下载到设备上，不立即激活固件。可以通过调用 [FirmwareActivate](#) 方法选择稍后在自选时间激活。两步指的是下载步骤 + 激活步骤。可以选择在下载前将支持该功能的所有设备都置于 STOP 模式，从而将固件更新文件下载到所有这些设备上。在所有设备都收到固件更新文件后，您可以选择同时激活所有设备

限制

API 仅支持通过 CPU 网络接口更新固件。不能通过 CM 或 CP 接口更新固件。

预防措施

激活固件会将 CPU 置于 STOP 模式（如果尚未处于 STOP 模式）。

示例：两步固件更新

```
//-----
//  API 入门指南 (页 34)
//-----
#region Two Step Firmware Update
IProfinetDevice myDevice = scannedDevices.FindDeviceByIP
(0xC0A80001); // 192.168.0.1
if (myDevice != null)
{
```

```

//-----
//      CPU      PROFINET
//-----
myDevice.Selected = true;
 retVal = myDevice.SetFirmwareFile("C:\\DeviceFirmwareFile.upd");
//
 retVal = myDevice.FirmwareUpdate(myDevice.ID, false,
FirmwareUpdateType.DownloadWithoutActivation);
//
 retVal = myDevice.FirmwareActivate(myDevice.ID);
myDevice.Selected = false;
//-----
// Updating CPU Local Modules or PROFINET Device
//-----
//-----
foreach (IModule module in myDevice.Modules)
{
    module.Selected = true;
    retVal = module.SetFirmwareFile("C:\\LocalModuleFirmwareFile.
upd");
    //
    retVal = myDevice.FirmwareUpdate(module.ID, false,
FirmwareUpdateType.DownloadWithoutActivation);
    //
    retVal = myDevice.FirmwareActivate(module.ID);
    module.Selected = false;
}
//-----
//      CPU          I/O
//-----
ICPU myCPU = myDevice as ICPU;
if (myCPU != null)
{
    foreach (IRemoteInterface interFace in myCPU.RemoteInterfaces)
    {
        foreach (IBaseDevice remoteDevice in interFace.Devices)
        {
            //-----
            //      I/O IM
            //-----
            remoteDevice.Selected = true;
            retVal = remoteDevice.SetFirmwareFile("C:
\\RemoteDeviceFirmwareFile.upd");
            //
            retVal = myCPU.FirmwareUpdate(remoteDevice.ID, false,
FirmwareUpdateType.DownloadWithoutActivation);
            //
            retVal = myCPU.FirmwareActivate(remoteDevice.ID);
            remoteDevice.Selected = false;
            //-----
            //      I/O IM
            //-----
            foreach (IModule module in remoteDevice.Modules)
            {
                module.Selected = true;
                retVal = module.SetFirmwareFile("C:
\\RemoteModuleFirmwareFile.upd");
                //
                retVal = myCPU.FirmwareUpdate(module.ID, false,
FirmwareUpdateType.DownloadWithoutActivation);
            }
        }
    }
}

```

```

        //
        retVal = myCPU.FirmwareActivate(module.ID);
        module.Selected = false;
    }
}
/*
 */
/*
 */
#endifregion

```

4.9.2.3 Identify 方法

通过 Identify 方法可使特定网络设备的设备 LED 或 HMI 屏幕闪烁。闪烁光信号有助于确定设备的物理位置。不能使用 Identify 方法识别路由器后面的设备。

返回类型	方法名称
Result	Identify

示例：通过闪烁识别设备

```

//-----
//  API 入门指南 (页 34)
//-----
#region Identify a device
uint targetIPAddress = 0xC0A80001; // 192.168.0.1
if (scanResult.Succeeded)
{
    //-----
    // IP          LED
    //-----
    IProfinetDevice dev =
scannedDevices.FindDeviceByIP(targetIPAddress);
    if (dev != null)
    {
        Result retVal = dev.Identify();
    }
}
#endifregion

```

4.9.2.4 IsFirmwareUpdateAllowed 方法

IsFirmwareUpdateAllowed 方法根据父设备和子设备的功能来确定是否可以在设备上执行固件更新。

返回类型	方法名称
bool	IsFirmwareUpdateAllowed

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

示例：检查是否允许在该设备上进行固件更新。

```

//-----
// "      API" (页 34)
//-----
#region Is fireware update allowed
dev = scannedDevices[0];
//-----
//      IModule
//-----
IModuleCollection modules = dev.Modules;

foreach (IModule mod in modules)
{
    //-----
    //Get/Check the allowed flag for every module on the station
    //-----
    bool bModuleCanDoFU = dev.IsFirmwareUpdateAllowed(mod.ID);
}

//-----
// Get the allowed flag for the parent device
//-----
bool DeviceCanDoFU = dev.IsFirmwareUpdateAllowed(dev.ID);
#endregion

```

4.9.2.5 IsFirmwareUpdate2StepAllowed 方法

IsFirmwareUpdate2StepAllowed 方法根据父设备和子设备的功能来确定是否可以在设备上执行两步固件更新。

返回类型	方法名称
bool	IsFirmwareUpdate2StepAllowed

参数			
名称	数据类型	参数类型	描述
hardwareID	uint	In	模块的硬件标识符

4.9.2.6 IsIPAddressOnNetwork 方法

`IsIPAddressOnNetwork` 方法确定 IP 地址在 `IProfinetDeviceCollection` (页 67) 中是否唯一。

返回类型	方法名称
<code>bool</code>	<code>IsIPAddressOnNetwork</code>

参数			
名称	数据类型	参数类型	说明
<code>nIP</code>	<code>uint</code>	<code>In</code>	要检查的 IP 地址

示例：检查某个 IP 地址是否在集合中

```

//-----
//      API 入门指南 (页 34)
//-----
#region Check if an IP address is already assigned to a device on
this network
    ulong targetMACAddress = 0x112233445566; // 11:22:33:44:55:66
IProfinetDeviceCollection scannedDevices;

//-----
//      IP
//-----
    IProfinetDevice dev =
scannedDevices.FindDeviceByMAC(targetMACAddress);

    if (dev != null)
    {
        if (dev.IsIPAddressOnNetwork(dev.IP))
        {
            // IP is assigned to another device in the collection
        }
    }
#endregion

```

4.9.2.7 IsPROFINETNameOnNetwork 方法

`IsProfinetNameOnNetwork` 方法确定 PROFINET 名称在 `IProfinetDeviceCollection` (页 67) 中是否唯一。

返回类型	方法名称
<code>bool</code>	<code>IsProfinetNameOnNetwork</code>

参数			
名称	数据类型	参数类型	说明
<code>strName</code>	<code>string</code>	<code>In</code>	需要检查的 PROFINET 名称

示例：检查 PROFINET 名称是否在集合中

```

//-----
//      API 入门指南 (页 34)
//-----
#region Check if an PROFINET name is already assigned to a device in
the scanned collection
ulong targetMACAddress = 0x112233445566; // 11:22:33:44:55:66
IProfinetDeviceCollection scannedDevices;

//-----
//          PROFINET
//-----
IProfinetDevice dev =
scannedDevices.FindDeviceByMAC(targetMACAddress);

if (dev != null)
{
    if (dev.IsProfinetNameOnNetwork(dev.ProfinetName))
    {
        // PROFINET name is a duplicate
    }
}
#endregion

```

4.9.2.8 IsUploadServiceDataAllowed 方法

使用 `IsUploadServiceDataAllowed` 方法根据父设备和子设备的功能来确定是否可以从设备上传服务数据。

返回类型	方法名称
<code>bool</code>	<code>IsUploadServiceDataAllowed</code>

参数			
名称	数据类型	参数类型	描述
<code>hardwareID</code>	<code>uint</code>	<code>In</code>	模块的硬件标识符

4.9.2.9 QuickPing 方法

`QuickPing` 方法可确定设备是否位于网络中。

方法名称	返回类型	说明
<code>QuickPing()</code>	<code>Result</code>	确定设备是否可在网络中访问。

示例：使用 QuickPing

下例会遍历网络上的扫描设备并尝试对每个设备执行 ping 操作。

```

//-----
//      API
//-----
//-----

```

```

Result result;
for (int deviceIdx = 0; deviceIdx < scannedDevices.Count;
deviceIdx++)
{
    //-----
    //      IProfinetDevice.
    //-----
    device = scannedDevices[deviceIdx];
    result = device.QuickPing();
    if (result.Succeeded)
    {
        //      ping
    }
    else
    {
        //      ping
    }
}
}

```

4.9.2.10 RefreshStatus 方法

当 ScanNetworkDevice (页 65)s 方法创建 IProfinetDeviceCollection (页 67) 时，扫描只返回关于各设备的最少量信息。要获取设备的所有可用信息，应调用 RefreshStatus 方法。通过此方法可与设备建立连接，并查询各种信息，然后断开与设备的连接。

方法名称	返回类型	说明
RefreshStatus()	Result	与此设备建立网络连接并读取可用设备信息。

使用 RefreshStatus

IProfinetDeviceCollection (页 67) 中表示设备的对象在网络扫描后仅含有各设备的部分数据。要获得关于设备的全部数据并正确使用 API，必须执行以下步骤：

1. 为每个受保护 CPU 调用 SetPassword (页 130)。要读取全部设备数据，CPU 密码必须提供足够的权限。
2. 为 IProfinetDeviceCollection (页 67) 中的每个设备调用 RefreshStatus 方法。RefreshStatus 方法更新表示设备状态的全部数据。

严重错误异常

如果 API 引发严重错误异常，说明您未正确地将 API 用于设备状态。在使用更多 API 功能前，记得调用 RefreshStatus 以使关于设备的数据保持最新。

示例：刷新状态

```

//-----
//      "      API"
//-----
#region Refresh scanned devices status
Result retVal = new Result();
foreach (IProfinetDevice dev in scannedDevices)
{

```

```

ICPU cpu = device as ICPU;
if (cpu != null)
{
    if (cpu.Protected)
    {
        //-----
        // PROFINET      CPU
        //                  API
        //-----
    }
    retVal = cpu.SetPassword(new
EncryptedString("CPUPassword"));
}
retVal = dev.RefreshStatus();
if (retVal.Succeeded)
{
    //-----
    //
    //-----
}
#endregion

```

说明

如果使用的设备是 S7-1500 或 S7-1200 CPU，可能需要输入密码才能进行刷新。这在上面的示例中有所说明。

4.9.2.11 ResetCommunicationParameters 方法

使用 `ResetCommunicationParameters` 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

返回类型	方法名称
<code>Result</code>	<code>ResetCommunicationParameters</code>

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` (页 101) 提供了 `ResetToFactoryDefaults` 方法 (页 123) 来复位 CPU。另外也无法为路由器后面的设备复位通信参数。

示例：复位设备的通信参数

```

//-----
//  API 入门指南 (页 34)
//-----
#region Reset a device's communication parameters
uint targetIPAddress = 0xC0A80001; // 192.168.0.1
//-----
-- IP
//-----
IProfinetDevice dev =
scannedDevices.FindDeviceByIP(targetIPAddress);
if (dev != null)
{
    Result retVal = dev.ResetCommunicationParameters();
}
#endregion

```

4.9.2.12 SetIP 方法

使用 SetIP 方法设置或修改设备的 IP 地址。

为确保此操作可成功执行，STEP 7 项目中的设备端口组态必须设置为“直接在设备上设置 IP 地址”(IP address is set directly on the device)。不能使用 SetIP 方法来为路由器后面的设备设置 IP 地址。

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

示例：设置 IP 地址

```

//-----
//  API 入门指南 (页 34)
//-----
#region Set device IP method
ulong targetMACAddress = 0x112233445566; // 11:22:33:44:55:66
//-----
// MAC          IP
//-----
IProfinetDevice dev =
scannedDevices.FindDeviceByMAC(targetMACAddress);
if (dev != null)
{
    Result retVal = dev.SetIP(0xC0A80001, 0xFFFFFFF00, 0x0);
}

```

```

    }

#endregion

```

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：

```

#region encode IP address
string userEnteredAddress = @"192.168.0.1"; //
//-----
//          uint
//-----
System.Net.IPEndPoint ip =
System.Net.IPEndPoint.Parse(userEnteredAddress);
byte[] bytes = ip.GetAddressBytes();
Array.Reverse(bytes);
uint encodedIp = BitConverter.ToInt32(bytes, 0); //      IP
#endregion

```

4.9.2.13 SetProfinetName 方法

使用 `SetProfinetName` 方法设置或修改设备的 PROFINET 站名称。

为确保此操作可成功执行，STEP 7 项目中的设备端口组态必须设置为“直接在设备上设置 PROFINET 名称”(PROFINET name is set directly on the device)。不能使用 `SetProfinetName` 方法来为路由器后面的设备设置 PROFINET 名称。

返回类型	方法名称
Result	<code>SetProfinetName</code>

参数			
名称	数据类型	参数类型	说明
<code>strName</code>	string	In	PROFINET 站的新名称

示例：设置 PROFINET 名称

```

//-----
//      API 入门指南 (页 34)
//-----
#region Set the PROFINET name of a device
ulong targetMACAddress = 0x112233445566; // 11:22:33:44:55:66
//-----
//      MAC          PROFINET
//-----
IProfinetDevice dev =
scannedDevices.FindDeviceByMAC(targetMACAddress);
if (dev != null)
{
    Result retVal = dev.SetProfinetName("new name");
}
#endregion

```

4.9.2.14 UploadServiceData 方法

UploadServiceData 方法可以从存在故障的 CPU 上传服务数据。

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”。
hardwareID	uint	In (可选)	设备的硬件 ID

示例：从存在故障的 CPU 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：

```

//-----
//  API 入门指南 (页 34)
//-----
//-----



#region Upload Service Data
uint targetIPAddress = 0xC0A80001; // 192.168.0.1
string strDiagFolder = @"c:\Diagnostics";

IProfinetDevice dev = devices.FindDeviceByIP(targetIPAddress);
if (dev != null)
{
    ICPU myCPU = dev as ICPU;

    if (myCPU != null)
    {
        myCPU.SetPassword(new EncryptedString("Password"));
        myCPU.Selected = true;
        if (myCPU.OperatingMode == OperatingState.Defective)
        {
            //
            Result retVal = myCPU.UploadServiceData(strDiagFolder);

            //
            //      UTC
            retVal = myCPU.UploadServiceData(strDiagFolder,
TimeFormat.UTC);
        }
        myCPU.Selected = false;
    }
}
#endregion

```

4.9.2.15 ValidateIPAddressSubnet 方法

使用 ValidateIPAddressSubnet 方法验证 IP 地址和子网掩码的组合是否兼容。

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

示例：验证设备的 IP 地址和子网掩码

```
-----  
//      API 入门指南 (页 34)  
//  
//-----  
#region Validate the subnet for the specified IP address  
ulong targetMACAddress = 0x112233445566; // 11:22:33:44:55:66  
//-----  
//      MAC          IP  
//-----  
IProfinetDevice dev =  
scannedDevices.FindDeviceByMAC(targetMACAddress);  
  
if (dev != null)  
{  
    Result retVal = dev.ValidateIPAddressSubnet(dev.IP,  
dev.SubnetMask);  
}  
#endregion
```

4.9.2.16 ValidatePROFINETName 方法

ValidatePROFINETName 方法用于验证提供的 PROFINET 名称。

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

示例：设置 PROFINET 名称前先对其进行验证

```
-----  
//      API 入门指南 (页 34)
```

```

// -----
#region Validate the PROFINET Name for a device
ulong targetMACAddress = 0x112233445566; // 11:22:33:44:55:66
// -----
//      MAC          PROFINET
// -----
IProfinetDevice dev =
scannedDevices.FindDeviceByMAC(targetMACAddress);

if (dev != null)
{
    string name = "ValidName";
    Result retVal = dev.ValidateProfinetName(name);
    if (retVal.Succeeded)
    {
        retVal = dev.SetProfinetName(name);
    }
}
#endregion

```

4.9.3 IProfinetDevice 事件

4.9.3.1 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：

```

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

void Dev_DataChanged(object sender, DataChangedEventArgs e)
{
    if (e.Type == DataChangedType.OperatingState)
    {
    }
}
#endregion

```

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 事件不主动监视实时网络，而是监视 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 (页 53) 对象。如以上示例中所示，此类具有一个 DataChangedType (页 161) 数据类型的单一属性“Type”。

4.9.3.2 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度。例如 FirmwareUpdate (页 82) 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：

```
#region
void UpdateCpuAtAddress(IProfinetDeviceCollection ProfiDevices, uint
MyTargetIPAddress, string updateFile)
{
    IProfinetDevice Profidevice =
    devices.FindDeviceByIP(MyTargetIPAddress);
    if (Profidevice != null)
    {
        Profidevice.ProgressChanged += new
        ProgressChangedEventHandler(Dev_ProgressChanged);
        Profidevice.SetFirmwareFile(updateFile);
        Profidevice.FirmwareUpdate(Profidevice.ID, true);
        dev.ProgressChanged -= new
        ProgressChangedEventHandler(Dev_ProgressChanged);
    }
}

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

ProgressChangedEventArgs 类

ProgressChanged 事件处理程序接收 ProgressChangedEventArgs ([页 53](#)) 对象。

属性名称	返回类型	说明
Action	ProgressAction (页 169)	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

4.10 IModuleCollection 类和模块属性

4.10.1 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。Modules 属性返回这些模块的集合 ([页 59](#))。

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

```
#region Identifying modules on a station
IProfinetDeviceCollection scannedDevices;
//-----
//      RefreshStatus()
//-----
Result retVal = scannedDevices[0].RefreshStatus();
if (retVal.Succeeded)
{
    //-----
    // The Modules property returns a collection of local IModule
    //-----
    IModuleCollection modules = scannedDevices[0].Modules;
    foreach (IModule mod in modules)
    {
        //-----
        // Get the article number for every module on the station.
        //-----
        string displayArticleNum = mod.ArticleNumber;
    }
}
#endregion
```

4.10.2 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 接口扩展了 **IHardware** 接口 (页 57)。**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
ConfiguredVersion	string	已组态版本的描述
Description	string	硬件项描述，基于订货号。此描述与用户可在 TIA Portal 和设备目录中看到的描述相同。（例如，“CPU-1215 DC/DC/DC”）
Failsafe	bool	根据其 ArticleNumber，此设备是否为故障安全设备？
FirmwareUpdateSupported	bool	该设备是否支持固件更新？
FirmwareVersion	string	设备的当前固件版本
HardwareNumber	int	硬件标识号
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
NewFirmwareNameIs-Valid	bool	固件文件对此设备或模块有效时。
NewFirmwareVersion	string	新固件版本
NewFirmwareFile	string	新固件的路径和文件名
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
ServiceDataSupported	bool	设备支持服务数据更新
Slot	uint	硬件项的插槽号
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	bool	API 是否支持此设备？

4.11 ICPU 接口

4.11.1 识别 IProfinetDeviceCollection 中的 CPU 设备

ScanNetworkDevices 方法 (页 65) 会生成一个 IProfinetDeviceCollection (页 67)。在该集合中，网络接口上的每台可访问设备都有一个对应的项。这些设备可包括 CPU、HMI 和其它设备。IProfinetDevice 接口提供适用于全部设备类别的属性和方法。ICPU 接口继承自 IProfinetDevice，因而支持所有 IProfinetDevice 属性和方法 (页 79)。

要确定指定的 IProfinetDevice 接口是否确实表示一个 CPU 设备，可将其转换为 ICPU。若转换成功，则表明该网络设备是 CPU，并且可以使用 ICPU 接口上的属性/方法。

说明

对于任何在 ICPU 上的操作，均必须设置 Selected (页 79) 标志。

如果 ICPU 上的操作是与安全相关的操作 (页 41-42)，必须设置 SelectedConfirmed (页 102)。

示例：确定 PROFINET 设备是否为 CPU

```
//-----
//    API 入门指南 (页 34)
//-----
#region PROFINET      ICPU
    IProfinetDeviceCollection scannedDevices;
    foreach (IProfinetDevice dev in scannedDevices)
    {
        ICPU myCPU = dev as ICPU;
        if (myCPU != null)
        {
            //-----
            //    CPU
            //    ICPU
            //-----
        }
    }
#endregion
```

4.11.2 ICPU 属性

ICPU 接口对 IProfinetDevice (页 79) 进行了扩展，增加了以下属性。为确保其能够返回当前信息，程序代码应首先调用 RefreshStatus (页 92) 方法。

属性名称	返回类型	说明
CertificateStore	ICertificateStore	包含有关证书及其是否正在使用的信息
CollectiveFSignature	UInt32	F-CPU 的集合 F 签名。对于标准 CPU，该值将为 0。
ConfigurationProtectionData	ConfigurationDataProtection	表示当前组态数据保护状态的枚举
ConnectedToCPCM	bool	该 CPU 通过 CP 或 CM 连接

属性名称	返回类型	说明
CPUProtectionLevel	ProtectionLevel	在 CPU 上设置的保护等级，与密码无关
DataLogFolder	IRemoteFolder	CPU 的 SIMATIC 存储卡中有关所有数据日志的信息
FileSystemFolder	ICardFolder	此属性用于 SIMATIC Automation Tool 用户界面中。此属性与 API 操作无关。
FSignature	UInt32	F-CPU 的 F 签名。对于标准 CPU，该值将为 0。
IdentityCrisis	bool	当无法确定设备的身份时为真
Initialized	bool	当设备或模块具有有效组态时为真
InterfaceNumber	uint	用于连接设备的端口
LastRefreshSuccessful	bool	当对 RefreshStatus 的上一次调用成功完成时为真
OperatingMode	OperatingState	表示 CPU 的当前模式。该值是只读的。
Password	EncryptedString	用于在设备上执行的函数中的 CPU 密码
PasswordProtectLevel	ProtectionLevel	合法 CPU 密码的保护等级
PasswordValid	bool	对 SetPassword() 的调用是否有效？
Protected	bool	CPU 当前是否受到保护？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 (页 171)。
RedundantCPU	bool	如果该 CPU 为冗余 CPU，则为 True。不能组态为冗余 CPU。
RecipeFolder	IRemoteFolder	CPU 的 SIMATIC 存储卡中有关所有配方的信息
RemoteInterfaces	List<IRemoteInterface>	为 CPU 组态的所有远程 I/O 接口 (页 137) 的列表。此属性的用法将在本文档的后续部分中介绍。
SDCard	SDCardType	获取 CPU 的 SDCard 类型，读取、写入、未知或无
SelectedConfirmed	bool	当用户从操作的确认提示中重新选择一个或多个设备并确认操作时，执行安全相关操作的方法必须将 SelectedConfirmed 标志设置为 TRUE。SelectedConfirmed 表示已选择且已确认操作。
SiblingSerialNumber	String	冗余 CPU 成员的序列号否则为空。
TIAVersion	String	与 CPU 项目关联的 TIA Portal 版本
TLSRequired	bool	当该标志为 True 时，将使用 TLS 证书，且必须信任该证书才能进行写操作。
TrustCertificateStore	TrustCertificateType	用于表示当前通信信任的枚举。

4.11.3 ICPU 标志

4.11.3.1 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	描述
HasSafetyProgram	bool	布尔值，用来设置设备是否具有安全程序。这可在连接到 CPU 时确定。
NewProgramFolder	string	表示新程序的文件夹位置。 该值通过 SetProgramFolder (页 131) 方法设置。
NewProgramName	string	新程序的名称。
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
NewProgramNameFSignature	uint	表示新项目的 FSignature。 在比较过程中用于确定 ProgramUpdate (页 120) 是否已成功完成
NewProgramNameGateway	uint	新程序中设备的网关
NewProgramNameHasSafetyPassword	bool	当使用具有安全密码 (页 171) 的有效安全程序调用 SetProgramFolder (页 131) 方法时为 True。
NewProgramNameIP	uint	存储在新程序中的 IP 地址
NewProgramNameIsSafety	bool	当使用有效的安全程序调用 SetProgramFolder (页 131) 方法时为 True。
NewProgramNameIsValid	bool	当使用有效程序调用 SetProgramFolder (页 131) 时为 True。
NewProgramNamePassword	EncryptedString	用于在 ProgramUpdate (页 120) 完成后尝试建立连接的 CPU 密码。 通过使用 SetProgramPassword (EncryptedString) (页 132) 来设置值。
NewProgramNamePasswordErrorCode	Result	存储上一调用以下内容时的错误代码： SetProgramPassword (页 132)
NewProgramNamePasswordIsSafety	bool	当使用有效密码调用 SetProgramPassword (页 132) 方法且新程序的密码为安全密码 (页 171) 时为 True。
NewProgramNamePasswordIsValid	bool	当使用有效密码调用 SetProgramPassword (页 132) 方法时为 True。
NewProgramNamePasswordLevel	ProtectionLevel	新程序的 CPU 密码的保护等级 (页 171)。
NewProgramNamePasswordPresent	bool	当调用 SetProgramFolder (页 131) 方法且程序受到保护时为 True
NewProgramNameSubnetMask	uint	新程序中设备的子网掩码

属性名称	返回类型	描述
ProgramUpdateSucceeded	bool	当 ProgramUpdate (页 120) 方法成功时为 True。 程序更新仍会返回错误。

4.11.3.2 恢复标志

API 中包括以下标志，这样就能够在安全设备中安全、正确地执行 Restore (页 124) 方法。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	描述
NewRestoreFile	string	新程序的文件位置。 该值通过 SetBackupFile (页 127) 方法设置
NewRestoreName	string	新程序的名称。
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
NewRestoreNameFSignature	uint	新项目的 FSignature。 在比较过程中用于确定 Restore (页 124) 是否已成功完成
NewRestoreNameIsSafety	bool	当调用 SetBackupFile (页 127) 方法且恢复文件为安全程序时为 True
NewRestoreNameIsValid	bool	当调用 SetBackupFile (页 127) 方法且恢复文件有效时为 True
NewRestoreNamePassword	EncryptedString	用于在 Restore (页 124) 完成后尝试建立连接的 CPU 密码。 通过使用 SetBackupFilePassword(EncryptedString) (页 127) 来设置值。
NewRestoreNamePasswordIsSafety	bool	当调用 SetBackupFilePassword (页 127) 包含有效的安全密码 (页 171) 时为 True
NewRestoreNamePasswordIsValid	bool	当调用 SetBackupFilePassword (页 127) 包含有效密码时为 True
RestoreNamePasswordIsSafety	bool	当调用 SetBackupFilePassword (页 127) 包含有效的安全密码 (页 171) 时为 True
RestoreNamePasswordIsValid	bool	当调用 SetBackupFilePassword (页 127) 包含有效密码时为 True
RestoreSucceeded	bool	Restore (页 124) 操作是否成功？

4.11.3.3 ICPU 支持和允许的标志

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
BackupAllowed	bool	该设备当前允许备份时为真
BackupSupported	bool	此设备支持备份时为真
CardBrowserAllowed	bool	
CardBrowserSupported	bool	
ChangeModeAllowed	bool	此设备当前允许 CPU 运行模式更改时为真
ChangeModeSupported	bool	此设备支持 CPU 运行模式更改时为真
FormatMCSAllowed	bool	该设备当前允许 MCS 格式时为真
FormatMCSSupported	bool	此设备支持 MCS 格式时为真
MemoryResetAllowed	bool	该设备当前允许存储器复位时为真
MemoryResetSupported	bool	此设备支持存储器复位时为真
PasswordAllowed	bool	该设备当前允许密码时为真
PasswordSupported	bool	此设备支持密码时为真
ProgramUpdateAllowed	bool	该设备当前允许程序更新时为真
ProgramUpdateSupported	bool	此设备支持程序更新时为真
RemoteDataLogsAllowed	bool	该设备当前允许读取数据日志时为真
RemoteDataLogsSupported	bool	该设备支持读取数据日志时为真
RemoteRecipesAllowed	bool	该设备当前允许配方访问时为真
RemoteRecipesSupported	bool	此设备支持配方访问时为真
RestoreAllowed	bool	该设备当前允许恢复时为真
RestoreSupported	bool	此设备支持恢复时为真
SecuritySupported	bool	此设备支持 TLS 安全和组态数据保护
SecurityAllowed	bool	此设备当前允许 TLS 安全和组态数据保护。
ServiceDataAllowed	bool	此设备当前允许读取服务数据时为真
ServiceDataSupported	bool	此设备支持读取服务数据时为真
SetTimeAllowed	bool	此设备当前允许设置系统时间时为真
SetTimeSupported	bool	此设备支持设置系统时间时为真

4.11.4 ICPU 方法

4.11.4.1 Backup 方法 (ICPU 接口)

Backup 方法用于备份 CPU 中的数据。当您调用 Backup 方法时，CPU 将创建备份文件。

Backup 方法将生成的备份文件存储于您在 strFile 输入参数中提供的位置。此备份文件可用于 Restore (页 124) 操作。备份文件包含完整 CPU 程序（硬件组态 + 软件）。不能只备份和恢复部分程序。

返回类型	方法名称
Result	Backup

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

限制

API 仅支持通过 CPU 网络接口备份 CPU。不能通过 CM 或 CP 接口备份 CPU。

备份 CPU

要将 CPU 备份到文件中，应用程序必须执行以下步骤：

1. 设置 Selected (页 102) 属性以选择 CPU。
2. 调用 Backup。
3. 清除 Selected (页 102) 属性。

示例：备份 CPU

```
-----  
// API 入门指南 (页 34)  
//  
-----  
#region CPU  
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;  
// 192.168.0.1  
if (myCPU != null)  
{  
    myCPU.Selected = true;  
    retVal = myCPU.Backup("C:\\MyBackup.s7pbkp");  
    myCPU.Selected = false;  
}  
/* */  
/* */  
/* */  
#endregion
```

4.11.4.2 CreateConfigurationDataProtectionCard 方法

使用 CreateConfigurationDataProtectionCard 方法将组态数据保护密码传送到随后可插入到 CPU 中的存储卡。重新启动 CPU 会将组态数据保护密码传送到 CPU。

方法名称	返回类型	说明
CreateConfigurationDataProtectionCa- rd ()	Result	创建组态数据保护卡。
参数		

方法名称		返回类型	说明
名称	数据类型	参数类型	描述
password	EncryptedString	In	密码
strDestinationDrive	String	In	存储卡的驱动器字母路径 ("E:")

示例：创建组态数据保护卡

```

//-----
//      API 入门指南 (页 34)
//-----
#region
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
if (myCPU != null)
{
    String strDrive = @"E:"; EncryptedString password = new
EncryptedString("password");
    //
    retVal = myCPU.CreateConfigurationDataProtectionCard(password,
strDrive);
}
/*
 */
/*
 */
#endregion

```

4.11.4.3 CreateFirmwareUpdateCard 方法

使用 CreateFirmwareUpdateCard 方法将一个或多个固件更新文件传送到随后可插入到 CPU 中的存储卡上。对 CPU 循环上电将随后执行固件更新

方法名称	返回类型	说明	
CreateFirmwareUpdateCard()	Result	通过一系列固件文件创建固件更新卡。	
参数			
名称	数据类型	参数类型	描述
aFirmwareFiles	String []	in	一系列固件更新文件及完整路径。
strDestinationDrive	String	in	存储卡的驱动器字母路径 ("E:")

示例：创建固件更新卡

```

//-----
//      API 入门指南 (页 34)
//-----
#region
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001)      ICPU
// 192.168.0.1
if (myCPU != null)
{
    String[] aFirmwareFiles = new String[2] { @"C:\Firmware1.upd",

```

```

 @"C:\Firmware2.upd" };
    String strDrive = @"E:";
    //
    retVal = myCPU.CreateFirmwareUpdateCard(aFirmwareFiles,
    strDrive);
}

/*
 *          */
/*
 *          */
#endifregion

```

4.11.4.4 CreateProgramUpdateCard 方法

使用 CreateProgramUpdateCard 方法将 CPU 程序传送到随后可插入到 CPU 中的存储卡上

方法名称	返回类型	说明		
CreateProgramUpdateCard() ()	Result	创建程序更新卡。		
参数				
名称	数据类型	参数类型	说明	
strProgramFolder	String	in	有效 CPU 程序卡项目的文件夹路径。	
strDestinationDrive	String	in	存储卡的驱动器字母路径 ("E:")	

示例：创建程序更新卡

```

//-----
//  API 入门指南 (页 34)
//
//-----
#region Create program update card
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
if (myCPU != null)
{
    String strProgramPath = @"C:\MyProgram"; // Memory card image
    created using TIA Portal
    String strDrive = @"E:";

    //
    retVal = myCPU.CreateProgramUpdateCard(strProgramPath, strDrive);
}
/*
 *          */
/*
 *          */
#endifregion

```

4.11.4.5 DeleteConfigurationDataProtectionPassword 方法

使用 DeleteConfigurationDataProtectionPassword 方法删除 CPU 中的组态数据保护密码。

方法名称	返回类型	说明
DeleteConfigurationDataProtectionPassword()	Result	删除 CPU 中的组态数据保护密码。

示例：删除组态数据保护密码

```

//-----
//  API 入门指南 (页 34)
//
//-----
#region
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
if (myCPU != null)
{
    //-----
    //  CPU
    //-----
    if (myCPU.SecurityAllowed)
    {
        //-----
        // 
        //-----
        retVal = myCPU.DeleteConfigurationDataProtectionPassword();
    }
}
/*           */
/*           */
/*           */
#endregion

```

4.11.4.6 DeleteDataLog 方法

DeleteDataLog 方法用于从 CPU 存储卡中删除数据日志文件。

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

示例：删除数据日志

```

//-----
//  API 入门指南 (页 34)
//

```

```

//-----
#region
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1           if (myCPU != null)
{
    //-----
    // CPU
    //----- (页 105)
    if (myCPU.RemoteDataLogsAllowed)
    {
        //-----
        // 
        // 
        //----- if (myCPU.DataLogFolder.Exists)
        {
            //----- 
            // 
            //----- foreach (IRemoteFile datalog in myCPU.DataLogFolder.Files)
            {
                datalog.Selected = true;
                //----- 
                // 
                //----- retVal = myCPU.DeleteDataLog(datalog.Name);
                datalog.Selected = false;
            }
        }
    }
    /* */ /* */ /* */
#endregion

```

4.11.4.7 DeleteRecipe 方法

使用 DeleteRecipe 方法从 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。

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

示例：删除配方

```

//-----
// API 入门指南 (页 34)
//
```

```

//-----
#region
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
if (myCPU != null)
{
    //-----
    // CPU          (页 105)
    //-----
    if (myCPU.RemoteRecipesAllowed)
    {
        //-----
        // -----
        //-----
        if (myCPU.RecipeFolder.Exists)
        {
            //-----
            // -----
            //-----
            foreach (IRemoteFile recipe in myCPU.RecipeFolder.Files)
            {
                recipe.Selected = true;
                //-----
                // -----
                //-----
                retVal = myCPU.DeleteRecipe(recipe.Name);
                recipe.Selected = false;
            }
        }
    }
}
#endregion

```

4.11.4.8 DetermineConfirmationMessage

使用 `DetermineConfirmationMessage` 方法确定在 F-CPU 上调用安全相关操作之前显示给用户的安全操作。编写在 F-CPU 上运行的任何代码之前，请参阅 安全相关操作的用户界面编程指南 (页 42)。

返回类型	方法名称
<code>ConfirmationType</code>	<code>DetermineConfirmationMessage</code>

参数			
名称	数据类型	参数类型	说明
<code>operation</code>	<code>FailsafeOperation</code>	<code>In</code>	要评估的操作

限制

API 包含一系列防御性编码检查。这些检查旨在确保您能正确地将 API 用于故障安全 CPU。如果遇到引发异常的严重错误，则说明您未正确使用 API。代码示例显示了 API 方法的特定调用顺序。遵循此调用顺序以正确使用 API。

ResetToFactoryDefaults 和 FormatMemoryCard 的先决条件

在调用 DetermineConfirmationMessage 之前，安全相关的操作

ResetToFactoryDefaults (页 123) 和 FormatMemoryCard (页 116) 有着相同的先决条件：

- 不要为标准 CPU 调用 DetermineConfirmationMessage，否则会导致严重错误。DetermineConfirmationMessage 方法仅适用于 F-CPU。
 - 满足以下任意条件时，将为 F-CPU 调用 DetermineConfirmationMessage：
 - CPU 具有安全程序。
 - CPU 受到保护。
- 否则，您无需确认消息。

ProgramUpdate 和 Restore 的先决条件

在调用 DetermineConfirmationMessage 之前，安全相关的操作 ProgramUpdate (页 120) 和 Restore (页 124) 有着相同的先决条件：

- 不要为标准 CPU 调用 DetermineConfirmationMessage，否则会导致严重错误。DetermineConfirmationMessage 方法仅适用于 F-CPU。
 - 满足以下任意条件时，将为 F-CPU 调用 DetermineConfirmationMessage：
 - CPU 具有安全程序。
 - CPU 受到保护。
 - 要更新的程序或要恢复的备份程序均是安全程序。
- 否则，您无需确认消息。

检查返回值以确定要显示的文本

DetermineConfirmationMessage 方法接受 FailsafeOperation (页 167) 类型的输入操作参数，该操作作为一种与安全相关的操作。如果 DetermineConfirmationMessage 调用正确，会返回 ConfirmationType (页 161) 枚举类型，可用来了解为确认消息显示的文本。下表显示了针对每个 ConfirmationType (页 161) 枚举显示的文本。

返回的 ConfirmationType	要显示的确认消息
SafetyPasswordIsBeingUsed	将使用安全密码启动针对标准程序的操作。
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。

在确认消息框中将消息显示给用户。如果想要在多个 F-CPU 上操作而不想显示一系列消息框，可以实施显示每个 F-CPU 的确认消息的代码。有关详细信息，请参见“安全相关操作的用户界面编程指南 (页 42)”。

示例：DetermineConfirmationMessage

```
#region
if (myCPU.Failsafe == false)
{
```

```
//-----
//-----
//-----
    return;
}
//-----
//      CPU
//-----
if ((operation == FailsafeOperation.FormatMCOperation) ||
    (operation == FailsafeOperation.ResetToFactoryOperation))
{
    if ((myCPU.HasSafetyProgram == false) && (myCPU.
Protected == false))
    {
        //-----
//-----
//-----
        return;
    }
}
else if (operation == FailsafeOperation.ProgramUpdateOperation)
{
    if ((myCPU.HasSafetyProgram == false) &&
        (myCPU.NewProgramNameIsSafety == false) &&
        (myCPU.Protected == false))
    {
        //-----
//-----
//-----
        return;
    }
}
else if (operation == FailsafeOperation.RestoreOperation)
{
    if ((myCPU.HasSafetyProgram == false) &&
        (myCPU.NewRestoreNameIsSafety == false) &&
        (myCPU.Protected == false))
    {
        //-----
//-----
//-----
        return;
    }
}
else
{
    //-----
//-----
//-----
    return;
}

//-----
//-----
//-----
String messageText = "";
ConfirmationType confirmType = myCPU.DetermineConfirmationMessage
(operation);
switch (confirmType)
{
    case ConfirmationType.SafetyPasswordIsBeingUsed:
        messageText = "An operation to a standard program is about to-
```

```

        be initiated using the safety password";
        break;
    case ConfirmationType.DeletingExistingSafetyProgram:
        messageText = "An existing safety program is about to be deleted";
        break;
    case ConfirmationTypeReplacingExistingSafetyProgram:
        messageText = "An existing safety program is about to be updated with another safety program";
        break;
    case ConfirmationType.
ReplacingExistingSafetyProgramWithNonSafetyProgram:
        messageText = "An existing safety program is about to be replaced by a standard program";
        break;
    case ConfirmationType.LoadingSafetyProgram:
        messageText = "A safety program is about to be loaded for the first time";
        break;
}
//-----
//-----
//-----
Console.WriteLine(messageText + " Y \" \" N \" \"");
String confirmation = Console.ReadLine();
switch (confirmation)
{
    case confirmation == "Y":
        //-----
        //                               SelectedConfirmed
        //-----
        myCPU.SelectedConfirmed = true;
        break;
    case confirmation = "N":
        myCPU.SelectedConfirmed = false;
        break;
}

/*
 *          */
/*
 *          */
#endifregion

```

4.11.4.9 DownloadRecipe 方法

使用 `DownloadRecipe` 方法将配方文件从编程设备下载到 CPU 存储卡。某些 CPU 不支持 (页 105) 远程访问配方。检查 `RemoteRecipesAllowed` 属性以确保当前 CPU 支持此功能。

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

示例：下载配方

```

//-----
//  API 入门指南 (页 34)
//-----
#region
if (myCPU.RemoteRecipesAllowed)
{
    string rcpFile = @"C:\NewRecipe.csv";
    if (myCPU != null)
    {
        IRemoteFolder recipes = myCPU.RecipeFolder;
        recipes.Selected = true;
        recipes.SetRemoteFile(rcpFile);
        retVal = myCPU.DownloadRecipe(rcpFile);
        recipes.Selected = false;
    }
}
#endregion

/*
 */
/*
 */
#endregion

```

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。

4.11.4.10 FormatMemoryCard 方法

使用 FormatMemoryCard 方法格式化插入到 CPU 中的可移动 SIMATIC 存储卡。

方法名称	返回类型	说明
FormatMemoryCard()	Result	格式化 CPU 可移除存储卡。
FormatMemoryCard(ResetOptions options)	Result	可选：通过附加选项格式化 CPU 可移除存储卡。

限制

API 仅支持通过 CPU 网络接口格式化存储卡。无法通过 CM 或 CP 接口格式化存储卡。

预防措施

格式化存储卡将 CPU 置于 STOP 模式。

在标准 CPU 上格式化存储卡

要在标准 CPU 上将存储卡格式化为出厂默认设置，应用程序必须执行以下步骤：

1. 设置 Selected (页 79) 属性以选择 CPU。
2. 如果 CPU 受到保护，通过能提供写访问权限 (页 171) 的密码调用 SetPassword (页 130)。
3. 调用 FormatMemoryCard。
4. 清除 Selected (页 79) 属性。

在 F-CPU 上格式化存储卡

要在 F-CPU 上格式化存储卡，应用程序必须执行以下步骤：

1. 设置 Selected (页 79) 属性以选择 CPU。
2. 如果 CPU 受到保护，通过安全密码 (页 171) 调用 SetPassword (页 130)。
3. 在格式化存储卡之前，确定确认消息 (页 112)（要显示的消息）并获取用户确认。
4. 设置 SelectedConfirmed (页 102) 属性。
5. 调用 FormatMemoryCard。
6. 清除 Selected (页 79) 和 SelectedConfirmed (页 102) 属性。

示例：格式化存储卡

```
//-----
//  API 入门指南 (页 34)
//
//-----
#region
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
if (myCPU != null)
{
    myCPU.Selected = true;
    if (myCPU.Failsafe == false)
    {
        //-----
        //  CPU
        //-----
        if (myCPU.Protected)
        {
            retVal = myCPU.SetPassword(new EncryptedString
("WriteAccessPassword"));
        }
        retVal = myCPU.FormatMemoryCard();
        myCPU.Selected = false;
    }
    else
    {
        //-----
        //  CPU
        //-----
        if (myCPU.Protected)
        {
            retVal = myCPU.SetPassword(new EncryptedString
("SafetyAccessPassword"));
        }
        FailsafeOperation operation = FailsafeOperation.
```

```

FormatMCOperation;
    //
    //      DetermineConfirmationMessage
    //
    //
    //
    //
    //
    retVal = myCPU.FormatMemoryCard();
    //-----
    //
    //
    //-----
    myCPU.Selected = false;
    myCPU.SelectedConfirmed = false;
}
/*
 */
/*
 */
#endifregion

```

4.11.4.11 GetCurrentDateTime 方法

GetCurrentDateTime 方法获取 CPU 的当前日期和时间。

返回类型	方法名称
Result	GetCurrentDateTime

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

示例：获取 CPU 的日期和时间。

```

//-----
//  API 入门指南 (页 34)
//
//-----
#region      CPU
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
if (myCPU != null)
{
    myCPU.SetPassword(new EncryptedString("ReadAccessPassword"));
    myCPU.Selected = true;
    DateTime curTime = new DateTime();
    retVal = myCPU.GetCurrentDateTime(out curTime);
    myCPU.Selected = false;
}

/*
 */
/*
*/

```

```
#endregion
```

4.11.4.12 GetDiagnosticsBuffer 方法

`GetDiagnosticsBuffer` 方法读取 CPU 的当前诊断条目。`GetDiagnosticsBuffer` 方法会返回 `DiagnosticsItem` (页 53) 对象的集合。

使用 `Language enum` (页 168) 参数获取采用特定语言的诊断条目：

返回类型	方法名称
<code>Result</code>	<code>GetDiagnosticsBuffer</code>

参数			
名称	数据类型	参数类型	说明
<code>DiagnosticsItems</code>	<code>List<DiagnosticsItem></code>	<code>Out</code>	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。
<code>Language</code>	<code>Language</code>	<code>In</code>	请求用于诊断缓冲区条目的语言。

示例：获取 CPU 诊断

```
//-----
//    API 入门指南 (页 34)
//-----
#region    CPU
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
List<DiagnosticsItem> aLogs = new List<DiagnosticsItem>();
if (myCPU != null)
{
    myCPU.SetPassword(new EncryptedString("ReadAccessPassword"));
    myCPU.Selected = true;
    retVal = myCPU.GetDiagnosticsBuffer(out aLogs, Language.English);
    if (retVal.Succeeded)
    {
        for (int idxLog = 0; idxLog < aLogs.Count; idxLog++)
        {
            //-----
            //-----
            string descrTimes = aLogs[idxLog].TimeStamp.ToString();
            //-----
            //-----
            string descrBasic = aLogs[idxLog].Description1;
            //-----
            //-----
            string descrDetail = aLogs[idxLog].Description2;
            //-----
            //-----
            string descrState = aLogs[idxLog].State.ToString();
        }
    }
}
```

```

        myCPU.Selected = false;
    }

/*
 */
/*
 */
#endifregion

```

4.11.4.13 MemoryReset 方法

MemoryReset 方法可复位 CPU 存储器。

返回类型	方法名称
Result	MemoryReset

示例：复位 CPU 存储器

```

//-----
//  API 入门指南 (页 34)
//
//-----
#region      CPU
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
if (myCPU != null)
{
    if (myCPU.Protected)
    {
        myCPU.SetPassword(new EncryptedString("WriteAccessPassword"));
    }
    myCPU.Selected = true;
    retVal = myCPU.MemoryReset();
    myCPU.Selected = false;
}

/*
 */
/*
 */
#endifregion

```

4.11.4.14 ProgramUpdate 方法

使用 ProgramUpdate 方法下载新 CPU 程序。

返回类型	方法名称
Result	ProgramUpdate

限制

API 仅支持通过 CPU 网络接口更新程序。无法通过 CM 或 CP 接口更新程序。

预防措施

更新程序将 CPU 置于停止模式。

创建程序更新文件夹

使用 TIA Portal 编程软件创建您的 HMI 运行系统软件。程序完成后，可以使用 ProgramUpdate 方法下载程序。在 TIA Portal 中，使用 TIA Portal 项目树中的读卡器功能创建一个文件夹。此文件夹将包含可供 ProgramUpdate 方法使用的 CPU 程序。创建文件夹后，将 PLC 的全部内容拖放至此新文件夹内。ProgramUpdate 方法仅支持下载完整 CPU 程序（硬件组态 + 软件）。不能只下载部分程序。

标准 CPU 的程序更新

要更新标准 CPU 的程序，应用程序必须执行以下步骤：

1. 设置 Selected (页 79) 属性以选择 CPU。
2. 如果 CPU 受到保护，通过能提供写访问权限 (页 171) 的密码调用 SetPassword (页 130)。
3. 调用 SetProgramFolder (页 131) 以选择包含要下载到 CPU 的新程序的文件夹。
4. 如果新程序设置了 CPU 保护等级，则下载后调用 SetProgramPassword (页 132) 以使用该程序。调用 SetProgramPassword (页 132) 时，必须提供具有写访问权限的密码 (页 171)。
5. 调用 ProgramUpdate。
6. 清除 Selected (页 79) 属性。

F-CPU 的程序更新

要更新 F-CPU 的程序，应用程序必须执行以下步骤：

1. 设置 Selected (页 79) 属性以选择 CPU。
2. 如果 CPU 受到保护，通过安全密码 (页 171) 调用 SetPassword (页 130)。
3. 调用 SetProgramFolder (页 131) 以选择包含要下载到 CPU 的新程序的文件夹。
4. 如果新程序设置了 CPU 保护等级，则下载后调用 SetProgramPassword (页 132) 以使用该程序。如果新程序是故障安全程序，则调用 SetProgramPassword (页 132) 时，必须提供安全密码 (页 171)。如果新程序不是故障安全程序，则调用 SetProgramPassword (页 132) 时，必须提供具有写访问权限的密码 (页 171)。
5. 更新程序前，确定确认消息 (页 112)（要显示的消息）并取得用户确认。
6. 设置 SelectedConfirmed (页 102) 属性。
7. 调用 ProgramUpdate。
8. 清除 Selected (页 79) 和 SelectedConfirmed (页 102) 属性。

示例：程序更新

```
//-----
//  API 入门指南 (页 34)
//
#region
```

```
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
if (myCPU != null)
{
    myCPU.Selected = true;
    if (myCPU.Failsafe == false)
    {
        //-----
        //      CPU
        //-----
        if (myCPU.Protected)
        {
            retVal = myCPU.SetPassword(new EncryptedString
("WriteAccessPassword"));
        }
        FailsafeOperation operation = FailsafeOperation.
ProgramUpdateOperation;
        retVal = myCPU.SetProgramFolder("C:\\MyProgramFolder");
        if (retValFailed && retVal.Error ==
ErrorCode.ProgramPasswordNeeded)
        {
            retVal = myCPU.SetProgramPassword(new
EncryptedString("WriteAccessProgramPassword"));
        }
        retVal = myCPU.ProgramUpdate();
        myCPU.Selected = false;
    }
    else
    {
        //-----
        //      CPU
        //-----
        if (myCPU.Protected)
        {
            retVal = myCPU.SetPassword(new
EncryptedString("SafetyAccessPassword"));
        }
        retVal = myCPU.SetProgramFolder("C:\\MyProgramFolder");
        if (retValFailed && retVal.Error ==
ErrorCode.ProgramPasswordNeeded)
        {
            retVal = myCPU.SetProgramPassword(new
EncryptedString("SafteyAccessProgramPassword"));
        }
    }
    //
    //      DetermineConfirmationMessage
    //
    //
    //
    //
    //
    //-----
    //
    //-----
}

retVal = myCPU.ProgramUpdate();
//-----
//-----
//-----
//-----
```

```

        myCPU.Selected = false;
        myCPU.SelectedConfirmed = false;
    }
}
#endregion

```

4.11.4.15 ResetToFactoryDefaults 方法

该方法用于将 CPU 复位为其出厂默认值。

方法名称	返回类型	说明
ResetToFactoryDefaults()	Result	将 CPU 复位为出厂默认设置。
ResetToFactoryDefaults(ResetOptions options)	Result	可选：通过可选的复位选项将 CPU 复位为出厂默认设置

限制

API 仅支持通过 CPU 网络接口复位为出厂默认设置。不能通过 CM 或 CP 接口将 CPU 复位为出厂默认设置。

预防措施

复位为出厂默认设置将 CPU 置于 STOP 模式。

将标准 CPU 复位为出厂默认设置

要将标准 CPU 复位为出厂默认设置，应用程序必须执行以下步骤：

1. 设置 Selected (页 79) 属性以选择 CPU。
2. 如果 CPU 受到保护，通过能提供写访问权限 (页 171) 的密码调用 SetPassword (页 130)。
3. 调用 ResetToFactoryDefaults。
4. 清除 Selected (页 79) 属性。

将 F-CPU 复位为出厂默认设置

要将 F-CPU 复位为出厂默认设置，应用程序必须执行以下步骤：

1. 设置 Selected (页 79) 属性以选择 CPU。
2. 如果 CPU 受到保护，通过安全密码 (页 171) 调用 SetPassword (页 130)。
3. 在将 F-CPU 复位为出厂默认设置之前，确定确认消息 (页 112)（要显示的消息）并取得用户确认。
4. 调用 ResetToFactoryDefaults。
5. 清除 Selected (页 79) 和 SelectedConfirmed (页 102) 属性。

示例：将 CPU 复位为出厂默认设置

```

//-----
// API 入门指南 (页 34)
//
//-----
#region CPU
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
if (myCPU != null)
{
    myCPU.Selected = true;
    if (myCPU.Failsafe == false)
    {
        //-----
        // CPU
        //-----
        if (myCPU.Protected)
        {
            retVal = myCPU.SetPassword(new EncryptedString
("WriteAccessPassword"));
        }
        retVal = myCPU.ResetToFactoryDefaults();
        myCPU.Selected = false;
    }
    else
    {
        //-----
        // CPU
        //-----
        if (myCPU.Protected)
        {
            retVal = myCPU.SetPassword(new EncryptedString
("SafetyAccessPassword"));
        }
        FailsafeOperation operation = FailsafeOperation.
ResetToFactoryOperation;
        //
        // DetermineConfirmationMessage
        //
        //
        //
        //
        //
        //

        retVal = myCPU.ResetToFactoryDefaults();
        //
        //
        //
        //
        myCPU.Selected = false;
        myCPU.SelectedConfirmed = false;
    }
}
/*
 */
/*
 */
/*
 */
#endregion

```

4.11.4.16 Restore 方法 (ICPU 接口)

使用 Restore 方法将备份文件恢复到 CPU。

返回类型	方法名称
Result	Restore

限制

API 仅支持通过 CPU 网络接口恢复备份文件。无法通过 CM 或 CP 接口恢复备份文件。

预防措施

Restore 方法将 CPU 置于 STOP 模式。

创建备份文件

调用 Backup (页 106) 方法时，CPU 将创建备份文件。Backup (页 106) 方法在您指定的位置存储生成的备份文件。此备份文件可用于 Restore 操作。备份文件包含完整 CPU 程序（硬件组态 + 软件）。不能只备份和恢复部分程序。

从标准 CPU 的备份文件恢复

要恢复标准 CPU 备份文件，应用程序必须执行以下步骤：

1. 设置 Selected (页 79) 属性以选择 CPU。
2. 如果 CPU 受到保护，通过能提供写访问权限 (页 171) 的密码调用 SetPassword (页 130)。
3. 调用 SetBackupFile (页 127) 以选择包含要下载到 CPU 的备份文件。
4. 如果备份程序设置了 CPU 保护等级，则下载后调用 SetBackupFilePassword (页 127) 以使用该程序。调用 SetBackupFilePassword (页 127) 时，必须提供具有写访问权限的密码 (页 171)。
5. 调用 Restore。
6. 清除 Selected (页 79) 属性。

从 F-CPU 的备份文件恢复

要恢复 F-CPU 备份文件，应用程序必须执行以下步骤：

1. 设置 Selected (页 79) 属性以选择 CPU。
2. 如果 CPU 受到保护，通过安全密码 (页 171) 调用 SetPassword (页 130)。
3. 调用 SetBackupFile (页 127) 以选择包含要下载到 CPU 的备份文件。
4. 如果备份程序设置了 CPU 保护等级，则下载后调用 SetBackupFilePassword (页 127) 以使用该程序。调用 SetBackupFilePassword (页 127) 时，必须提供安全密码 (页 171)。
5. 在恢复备份程序之前，确定确认消息 (页 112)（要显示的消息）并取得用户确认。
6. 设置 SelectedConfirmed (页 102) 属性。

7. 调用 `Restore`。
8. 清除 `Selected` (页 79) 和 `SelectedConfirmed` (页 102) 属性。

示例：从备份文件恢复

```

//-----
//  API 入门指南 (页 34)
//-----
#region    CPU
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU; // 192.168.0.1
if (myCPU != null)
{
    myCPU.Selected = true;
    if (myCPU.Failsafe == false)
    {
        //-----
        //      CPU
        //-----
        if (myCPU.Protected)
        {
            retVal = myCPU.SetPassword(new EncryptedString
("WriteAccessPassword"));
        }
        retVal = myCPU.SetBackupFile("C:\\MyBackup.s7pbkp");
        //-----
        //      CPU
        //-----
        retVal = myCPU.SetBackupFilePassword(new
EncryptedString("WriteAccessBackupFilePassword"));
        retVal = myCPU.Restore();
        myCPU.Selected = false;
    }
    else
    {
        //-----
        //      CPU
        //-----
        if (myCPU.Protected)
        {
            retVal = myCPU.SetPassword(new
EncryptedString("SafetyAccessPassword"));
        }
        FailsafeOperation operation =
FailsafeOperation.RestoreOperation;
        retVal = myCPU.SetBackupFile("C:\\MyBackup.s7pbkp");
        //-----
        //      CPU
        //-----
        retVal = myCPU.SetBackupFilePassword(new
EncryptedString("SafteyAccessBackupFilePassword"));

        //
        //      DetermineConfirmationMessage
        //
    }
}

```

```

//  

//  

//  

//  

    retVal = myCPU.Restore();  

//-----  

//  

//  

//-----  

    myCPU.Selected = false;  

    myCPU.SelectedConfirmed = false;  

}  

}  

/* */  

/* */  

/* */  

#endifregion

```

4.11.4.17 SetBackupFile 方法

`SetBackupFile` 方法为通过 `Restore` (页 124) 方法存储的备份文件设置完整的路径和文件名。

`SetBackupFile` 方法设置以下 ICPU 恢复标志 (页 105) :

- `NewRestoreName`
- `NewRestoreFile`
- `NewRestoreNameIsValid`
- `NewRestoreNameIsSafety`
- `NewRestoreNameFSignature`

返回类型	方法名称
<code>Result</code>	<code>SetBackupFile</code>

参数			
名称	数据类型	参数类型	说明
<code>strFile</code>	<code>string</code>	<code>In</code>	备份文件的位置

示例：设置备份文件

`SetBackupFile` 是将备份文件恢复到 CPU 的过程的一个部分。

要了解恢复备份文件的正确、完整的步骤顺序，请参见 `Restore` 方法主题 (页 124) 中的示例和说明。

4.11.4.18 SetBackupFilePassword 方法

恢复备份文件后，应用程序会尝试重新连接至设备。如果您恢复到 CPU 的备份程序受到保护，使用 `SetBackupFilePassword` 方法设置更新后的 CPU 密码。利用更新后的密码，应用程序可重新获得设备的访问权限。

该方法会在 ICPU 对象中设置 NewRestoreNamePassword 标志：

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

示例：设置备份文件的 CPU 密码

SetBackupFilePassword 是将备份文件恢复到 CPU 的过程的一个部分。

要了解恢复备份文件的正确、完整的步骤顺序，请参见 [Restore 方法主题 \(页 124\)](#) 中的示例和说明。

4.11.4.19 SetConfigurationDataProtectionPassword 方法

使用 SetConfigurationDataProtectionPassword 方法设置 CPU 中的组态数据保护密码。

方法名称	返回类型	说明	
SetConfigurationDataProtectionPassword()	Result	设置 CPU 中的组态数据保护密码。	
参数			
名称	数据类型	参数类型	描述
password	EncryptedString	In	新组态数据保护密码。

示例：设置组态数据保护密码

```

//-----
//  API 入门指南 (页 34)
//-----
#region
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
if (myCPU != null)
{
    //-----
    //  CPU
    //-----
    if (myCPU.SecurityAllowed)
    {
        //-----
        // 
        //-----
        retVal = myCPU.SetConfigurationDataProtectionPassword(new
EncryptedString("password"));
    }
}

```

```

}
/*
 */
/*
 */
#endregion

```

4.11.4.20 SetCurrentDateTime 方法

`SetCurrentDateTime` 方法设置 CPU 的当前日期和时间。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 时间而非本地时间。

返回类型	方法名称
Result	<code>SetCurrentDateTime</code>

参数			
名称	数据类型	参数类型	说明
time	<code>System.DateTime</code>	In	新的 CPU 当前时间值

示例：设置 CPU 日期和时间

```

//-----
//      API 入门指南 (页 34)
//-----
#region Set current time and Date
foreach (IProfinetDevice dev in devices)
{
    myCPU = dev as ICPU;
    if (myCPU != null)
    {
        myCPU.SetPassword(new EncryptedString("Password"));
        myCPU.Selected = true;
        etVal = myCPU.SetCurrentDateTime(DateTime.UtcNow);
        myCPU.Selected = false;
    }
}
#endregion
/*
 */
/*
 */

```

4.11.4.21 SetOperatingState 方法

`SetOperatingState` 方法设置 CPU 的操作模式。

返回类型	方法名称
Result	<code>SetOperatingState</code>

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

示例：设置操作模式

```

//-----
// API 入门指南 (页 34)
//-----
#region CPU
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
if (myCPU != null)
{
    myCPU.Selected = true;
    if (myCPU.ChangeModeAllowed)
    {
        retVal = myCPU.SetOperatingState(OperatingStateREQ.Run);
    }
    myCPU.Selected = false;
}
/* */ /* */
/* */ /* */
#endregion

```

4.11.4.22 SetPassword 方法

使用 SetPassword 方法为受保护的 CPU 设置密码。

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	设备的 CPU 密码

使用受保护 CPU

如果 CPU 受到保护，则对于 ICPU 接口，Protected (页 102) 属性为 true。在调用 SetPassword 方法前必须先检查 CPU 是否受到保护。如果 CPU 未受到保护，则不要调用 SetPassword 方法。如果为没有受到保护的 CPU 调用 SetPassword 方法，API 会引发严重错误异常。出现严重错误异常意味着您未正确使用 API。

对于受保护的 CPU，应用程序必须使用能提供充分访问级别 (页 171) 的密码调用 SetPassword 方法。设置有效密码后，可以刷新设备状态。

最佳做法是，在扫描网络后要立即进行通信的所有设备上都调用 SetPassword。将标准 CPU 上的密码设置为具有写访问权限的密码 (页 171)，将 F-CPU 上的密码设置为安全密码 (页 171)，以避免出现 API 错误。

示例：SetPassword

```

//-----
// API 入门指南 (页 34)
//-----

```

```

#region
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
if (myCPU != null)
{
    if (myCPU.Failsafe == false)
    {
        //-----
        // CPU
        //-----
        if (myCPU.Protected)
        {
            retVal = myCPU.SetPassword(new EncryptedString
("WriteAccessPassword"));
        }
    }
    else
    {
        //-----
        // CPU
        //-----
        if (myCPU.Protected)
        {
            retVal = myCPU.SetPassword(new EncryptedString
("SafetyAccessPassword"));
        }
    }
}
/* */ /* */
#endregion

```

4.11.4.23 SetProgramFolder 方法

调用 SetProgramFolder 以选择包含要用 ProgramUpdate (页 120) 方法下载到 CPU 的新程序的文件夹。

返回类型	方法名称
Result	SetProgramFolder

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

SetProgramFolder 方法会在 ICPU 对象中设置以下程序更新标志 (页 103-104) :

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

`SetProgramFolder` 方法还会在 `ICPU` 对象中设置以下程序更新标志 (页 103-104) :

- `NewProgramNameIsSafety`
- `NewProgramNameHasSafetyPassword`

示例 : 为设置用于程序更新的程序文件夹

`SetProgramFolder` 是 CPU 程序更新过程的一个部分。

要了解更新 CPU 程序的正确、完整的步骤顺序, 请参见 `ProgramUpdate` 方法主题 (页 120) 中的示例和说明。

4.11.4.24 SetProgramPassword 方法

返回类型	方法名称
<code>Result</code>	<code>SetProgramPassword</code>

参数			
名称	数据类型	参数类型	描述
<code>password</code>	<code>EncryptedString</code>	<code>In</code>	设置要在以下过程中传递给 CPU 的项目的 CPU 密码 : <code>ProgramUpdate</code> (页 120)

`SetProgramPassword` 方法会在 `ICPU` 对象中设置以下程序更新标志 (页 103-104) :

- `NewProgramNamePasswordIsValid`
- `NewProgramNamePasswordIsSafety`
- `NewProgramNamePasswordLevel`

示例 : 设置用于程序更新的程序密码

`SetProgramPassword` 是 CPU 程序更新过程的一个部分。

要了解更新 CPU 程序的正确、完整的步骤顺序, 请参见 `ProgramUpdate` (页 120) 中的示例和说明。

4.11.4.25 SetTrustCertificateStore 方法

使用 `SetTrustCertificateStore` 方法设置 CPU 中当前证书的信任值。

方法名称	返回类型	说明	
<code>SetTrustCertificateStore()</code>	<code>Result</code>	调用此方法可信任当前证书。	
参数			
名称	数据类型	参数类型	说明
<code>type</code>	<code>TrustCertificateType</code>	<code>in</code>	信任类型 (始终、从不或需要选择)

示例：设置信任证书

```

//-----
//      API 入门指南 (页 34)
//-----
#region
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
if (myCPU != null)
{
    //-----
    //      CPU
    //-----

    if (myCPU.TLSTrustRequired)
    {
        // If on Windows, show certificate chain in windows dialog
        myCPU.CertificateStore.ShowDialog();
        //-----
        //      bTrust
        //-----
        bool bTrust = true;
        if (bTrust)
            retVal =
myCPU.SetTrustCertificateStore(TrustCertificateType.Always);
        else
            retVal =
myCPU.SetTrustCertificateStore(TrustCertificateType.Never);
    }
    /*
    /*
     */
    /*
     */
#endregion

```

4.11.4.26 UploadDataLog 方法

UploadDataLog 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。

返回类型	方法名称
Result	UploadDataLog

参数			
名称	数据类型	参数类型	描述
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

示例：上传数据日志

```

//-----
//      API 入门指南 (页 34)
//
```

```

//-----
#region
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
if (myCPU != null)
{
    //-----
    //          (页 105)
    //-----
    if (myCPU.RemoteDataLogsAllowed)
    {
        //-----
        //-----
        //-----
        if (myCPU.DataLogFolder.Exists)
        {
            //-----
            //-----
            //-----
            foreach (IRemoteFile datalog in myCPU.DataLogFolder.Files)
            {
                datalog.Selected = true;
                //-----
                //-----
                //-----
                retVal = myCPU.UploadDataLog(datalog.Name,
                                              @"C:\MyDataLogs");
                datalog.Selected = false;
            }
        }
    }
}
/*           */
/*           */
/*           */
#endif

```

4.11.4.27 UploadRecipe 方法

使用 UploadRecipe 方法将配方文件副本从 CPU 存储卡上传到编程设备。

返回类型	方法名称
Result	UploadRecipe

参数			
名称	数据类型	参数类型	描述
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

示例：上传配方

```

//-----
//  API 入门指南 (页 34)
//
//-----
#region
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
if (myCPU != null)
{
    //-----
    //      CPU          (页 105)
    //-----
    if (myCPU.RemoteRecipesAllowed)
    {
        //-----
        //
        //
        //-----
        if (myCPU.RecipeFolder.Exists)
        {
            //-----
            //
            //-----
            foreach (IRemoteFile recipe in myCPU.RecipeFolder.Files)
            {
                recipe.Selected = true;
                //-----
                //
                //-----
                retVal = myCPU.UploadRecipe(recipe.Name, @"C:
\MyRecipes");
                recipe.Selected = false;
            }
        }
    }
}
/*
 */
/*
 */
#endregion

```

4.11.4.28 ValidateProgramFolder 方法

使用 ValidateProgramFolder 方法验证 CPU 程序文件夹，并确定此程序是否为故障安全程序，以及是否适合下载到此 CPU。

方法名称	返回类型	说明	
ValidateProgramFolder() ()	Result	验证 CPU 程序文件夹，并确定此程序是否为故障安全程序，以及此程序是否适合下载到此 CPU。	
参数			
名称	数据类型	参数类型	描述
password	String	In	密码
bSafety	bool	Out	如果此程序为安全程序，则为 True。

方法名称	返回类型	说明
bValid	bool	Out 如果此程序适合此 CPU 下载，则为 True。

示例：验证程序文件夹

```

//-----
//  API 入门指南 (页 34)
//
//-----
#region
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
if (myCPU != null)
{
    String strProgramPath = @"C:\MyProgram"; //      TIA Portal

    bool bFailsafeProgram;
    bool bValid;

    //
    retVal = myCPU.ValidateProgramFolder(strProgramPath, out
bFailsafeProgram, out bValid);
}
/*
*/
/*
*/
/*
*/
#endregion

```

4.11.5 RemoteInterfaces 属性**4.11.5.1 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 102) 的 RemoteInterfaces 属性获取有关附加在这些远程接口中的设备的信息。

示例：检查远程接口

```

//-----
//  API 入门指南 (页 34)
//
//-----
#region      CPU
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
if (myCPU != null)
{
    myCPU.Selected = true;
    List<IRemoteInterface> decentralNets =
myCPU.RemoteInterfaces;
    foreach (IRemoteInterface net in decentralNets)
    {
        //-----
        //
        //-----
    }
}

```

```

        myCPU.Selected = false;
    }
    /*
    */
    /*
     */
#endregion

```

4.11.5.2 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	描述
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，如 RemoteInterfaceType 枚举 (页 170) 中的定义
Name	string	远程接口的已组态名称

可以使用 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。该接口具有可用于 IProfinetDevice 的属性集合的一个子集，并提供这些设备可用的有限 API 功能。

IBaseDevice 接口上提供以下属性：

属性名称	返回类型	描述
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。 示例：“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？
Family	DeviceFamily	此设备属于哪个产品系列？更多信息，请参见 DeviceFamily (页 162) 枚举的描述。
FirmwareUpdateAllowed	bool	此设备是否可以进行固件更新？
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

属性名称	返回类型	描述
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 (页 100)。
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
SubSlot	uint	设备的子插槽。这与 SB-1200 等可插拔子模块相关。
Supported	bool	当前的 SIMATIC Automation Tool API 操作是否支持检测到的网络设备？

使用 IRemoteInterface 的设备属性，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 136) 主题中的示例：

此示例遍历 CPU 的所有远程 PROFINET 接口，并创建一个包含所有分散式站的订货号的列表。IBaseDevice 还支持 Modules 属性，因此您可以将示例进一步扩展，不仅能查看分散式站，还能查看每个站中的所有本地模块。

示例：检查远程接口上站的属性

```

//-----
// API 入门指南 (页 34)
//-----
#endregion
#region CPU
ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
// 192.168.0.1
if (myCPU != null)
{
    myCPU.Selected = true;
    List<IRemoteInterface> decentralNets = myCPU.RemoteInterfaces;
    List<string> orderNumbers = new List<string>();
    foreach (IRemoteInterface net in decentralNets)
    {
        //-----
        //
        //-----
    }
}
//-----
// -----
//-----

```

```

        if (net.InterfaceType == RemoteInterfaceType.Profinet)
    {
        //-----
        //
        //-----
        List<IBaseDevice> stations = net.Devices;

        //-----
        //
        //-----
        foreach (IBaseDevice station in stations)
        {
            orderNumbers.Add(station.ArticleNumber);
        }
    }
    myCPU.Selected = false;
}
/*
 */
/*
 */
/*
 */
#endifregion

```

4.12 ICPUCClassic 接口

4.12.1 识别 IProfinetDeviceCollection 中的经典 CPU 设备

该 ScanNetworkDevices 方法 ([页 65](#)) 会生成一个 IProfinetDeviceCollection ([页 67](#))。在该集合中，PROFINET 网络上的每台可访问设备都有一个对应的项。这些设备可以包括 S7-300 和 S7-400 (经典) CPU。

IProfinetDevice ([页 79](#)) 提供适用于各类设备的属性和方法。ICPUCClassic 接口提供了一种特定于经典 CPU 设备的方法，即 GetDiagnosticsBuffer ([页 140](#))。

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 ICPUCClassic。若转换成功，则表明该网络设备是经典的 CPU，并且可以使用 ICPUCClassic 接口的方法。

示例：确定 PROFINET 设备是否为经典 CPU

```

//-----
//  API 入门指南 (页 34)
//
//-----
#region          CPU
ICPUCClassic myCPUClassic = scannedDevices.FindDeviceByIP(0xC0A80001)
as ICPUCClassic;
// 192.168.0.1
if (myCPUClassic != null)
{
    //-----
    //      CPU
    //      ICPUCClassic
    //-----
}

```

```
#endregion
```

4.12.2 GetDiagnosticsBuffer 方法

ICPUClassic 接口的 GetDiagnosticsBuffer 方法从经典 CPU 读取当前的诊断条目。GetDiagnosticsBuffer 方法会返回 DiagnosticsItem (页 53) 对象的集合。以下示例将在 IProfinetDeviceCollection 中搜索位于特定 IP 地址的经典 CPU。成功搜索到后，将从该经典 CPU 读取诊断信息。使用 Language enum (页 168) 参数获取采用特定语言的诊断条目。

返回类型	方法名称
Result	GetDiagnosticsBuffer

参数			
名称	数据类型	参数类型	描述
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。

```
/*
// API 入门指南 (页 34)
*/
/*
#region      CPU
ICPUClassic myCPUClassic = scannedDevices.FindDeviceByIP(0xC0A80001)
as ICPUClassic;
// 192.168.0.1
List<DiagnosticsItem> aLogs = new List<DiagnosticsItem>();
if (myCPUClassic != null)
{
    myCPUClassic.Selected = true;
    retVal = myCPUClassic.GetDiagnosticsBuffer(out aLogs,
Language.English);
    if (retVal.Succeeded)
    {
        for (int idxLog = 0; idxLog < aLogs.Count; idxLog++)
        {
            //get information time stamp
            string descrTimes = aLogs[idxLog].TimeStamp.ToString();
            //get information basic description
            string descrBasic = aLogs[idxLog].Description1;
            //get information detailed description
            string descrDetail = aLogs[idxLog].Description2;
            //get information state (incoming/outgoing)
            string descrState = aLogs[idxLog].State.ToString();
        }
    }
    myCPUClassic.Selected = false;
}

/*
 */
/*
 */
#endregion
```

4.13 IHMI 接口

4.13.1 IHMI 接口

`ScanNetworkDevices` 方法 (页 65) 会生成一个 `IProfinetDeviceCollection` (页 67)。在该集合中，网络接口上的每台可访问设备都有一个对应的项。这些设备可包括 CPU、HMI 和其它设备。`IProfinetDevice` 接口提供适用于全部设备类别的属性和方法。`IHMI` 接口继承自 `IProfinetDevice`，因而支持所有 `IProfinetDevice` 属性和方法 (页 79)。`IHMI` 接口提供仅适用于 HMI 设备的属性和方法。

要确定指定的 `IProfinetDevice` 接口是否表示一个 HMI 设备，可将其转换为 `IHMIDevice`。若转换成功，则表明该网络设备是 HMI 设备，并且可以使用 `IHMIDevice` 接口上的方法。

示例：确定 PROFINET 设备是否为 HMI

```
//--  
// API 入门指南 (页 34)  
//  
//  
//--  
#region      HMI PROFINET  
IHMI devAsHmi = dev as IHMI;  
if (devAsHmi != null)  
{  
    //--  
    //      HMI  
    //  IHMI  
    //--  
}  
#endregion  
/*           */  
/*           */  
/*           */
```

4.13.2 IHMI 属性和标志

4.13.2.1 IHMI 属性

IHM1 接口包含下列属性：

属性名称	返回类型	描述
DeviceType	string	对象所代表的 HMI 类型
FirmwareDeviceVersion	string	HMI 当前的固件版本
RuntimeDeviceVersion	string	HMI 当前的运行系统版本
TransferChannel	HMITransferChannel (页 168)	用于与 HMI 设备通信的协议。 默认值 : PN_IE

4.13.2.2 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

4.13.2.3 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

4.13.2.4 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
BackupSupported	bool	设备支持备份时为真
ProgramUpdateAllowed	bool	设备允许程序更新时为真
ProgramUpdateSupported	bool	设备支持程序更新时为真
RestoreAllowed	bool	设备允许恢复时为真
RestoreSupported	bool	设备支持恢复时为真
ScheduleFullBackup	bool	SIMATIC Automation Tool

4.13.3 IHMI 方法

4.13.3.1 Backup 方法 (IHMI 接口)

使用 IHMI 接口的 Backup 方法备份 HMI 的数据。

返回类型	方法名称
Result	Backup

参数			
名称	数据类型	参数类型	描述
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（默认） • 配方 • 用户管理数据

备份 HMI

要备份 HMI，应用程序必须执行以下步骤：

1. 设置 Selected (页 79) 属性以选择 HMI。
2. 使用一个对应于待执行备份的类型的可选参数调用 Backup。
3. 清除 Selected (页 79) 属性。

示例：备份 HMI

```

// -----
// API 入门指南 (页 34)
// -----
#region HMI
IHMI myHMI = scannedDevices.FindDeviceByIP(0xC0A80001) as IHMI;
// 192.168.0.1
if (myHMI != null)
{
    myHMI.Selected = true;
    //
    retVal = myHMI.Backup("C:\\MyHMIBackup.s7pbkp",
    BackupType.Recipes);

    // HMI
    retVal = myHMI.Backup("C:\\MyHMIBackup.s7pbkp");
    myHMI.Selected = false;
}
/*
 */
/*
 */
#endif

```

4.13.3.2 FirmwareUpdate 方法

使用 IHMI 接口的 FirmwareUpdate 方法执行 HMI 固件更新。

方法名称	返回类型	描述
FirmwareUpdate()	Result	执行 HMI 固件更新。

示例：更新 HMI 固件

```
//-----
//  API 入门指南 (页 34)
//
//-----
#region HMI
IHMI myHMI = scannedDevices.FindDeviceByIP(0xC0A80001) as IHMI;
// 192.168.0.1
if (myHMI != null)
{
    myHMI.Selected = true;
    myHMI.SetFirmwareFile(@"c:\myFolder\Firmware\Simatic.HMI\
KTP900_V16_00_00_03.fwf");
    retVal = myHMI.FirmwareUpdate();
    myHMI.Selected = false;
}
/* */ /* */
/* */ /* */
#endregion
```

4.13.3.3 ProgramUpdate 方法 (IHMI 接口)

IHMI 接口的 ProgramUpdate 方法用于更新 HMI 设备的操作系统和运行系统软件。

返回类型	方法名称
Result	ProgramUpdate

创建程序更新文件夹

使用 TIA Portal 编程软件创建您的 HMI 运行系统软件。在 TIA Portal 中，使用 TIA Portal 项目树中的读卡器功能创建一个文件夹。此文件夹将包含可供 ProgramUpdate 方法使用的 HMI 操作系统和运行系统软件。创建文件夹后，将 HMI 设备的全部内容拖放至这个新文件夹中。ProgramUpdate 方法会下载 HMI 操作系统和运行系统软件。不能只更新部分程序。

要成功进行程序更新，新的程序文件夹必须包含以下文件：

DownloadTask.xml
ProjectCharacteristics.rdf

在 TIA Portal 中创建的文件夹通常具有以下格式：

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

文件夹名称示例：

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

HMI 的程序更新

要更新 HMI 程序，应用程序必须执行以下步骤：

1. 设置 Selected (页 79) 属性以选择 HMI。
2. 调用 SetProgramFolder (页 146) 以选择包含要下载到 HMI 的新程序的文件夹。
3. 调用 ProgramUpdate。
4. 清除 Selected (页 79) 属性。

示例：更新 HMI 程序

```
//-----
//  API 入门指南 (页 34)
//
//-----
#region    HMI
IHMI myHMI = scannedDevices.FindDeviceByIP(0xC0A80001) as IHMI;
// 192.168.0.1
if (myHMI != null)
{
    myHMI.Selected = true;
    myHMI.SetProgramFolder(
        @"c:\myFolder\ProgramUpdate\Simatic.HMI\RT_Projects\Project1");
    //-----
    //          HMI
    //
    //-----
    retVal = myHMI.ProgramUpdate();
    myHMI.Selected = false;
}
/*           */
/*           */
/*           */
#endregion
```

4.13.3.4 Restore 方法 (IHMI 接口)

使用 IHMI 接口的 Restore 方法从之前的 HMI 设备备份恢复 HMI 设备数据。

返回类型	方法名称
Result	Restore

从 HMI 备份文件恢复

要恢复 HMI 备份文件，应用程序必须执行以下步骤：

1. 设置 Selected (页 79) 属性以选择 HMI。
2. 调用 SetBackupFile (页 146) 以选择包含要下载到 HMI 的备份文件。
3. 调用 Restore。
4. 清除 Selected (页 79) 属性。

示例：恢复 HMI 备份文件

```

//-----
// API 入门指南 (页 34)
//-----
#endregion HMI
IHMI myHMI = scannedDevices.FindDeviceByIP(0xC0A80001) as IHMI; // 192.168.0.1
if (myHMI != null)
{
    myHMI.Selected = true;
    retVal = myHMI.SetBackupFile(@"C:\MyFolder\Backup.s7pbkp");
    retVal = myHMI.Restore();
    myHMI.Selected = false;
}
/*
/*
/*
#endregion

```

4.13.3.5 SetBackupFile 方法

使用 IHMI 接口的 SetBackupFile 方法选择要恢复到 HMI 的备份文件。

返回类型	方法名称		
Result	SetBackupFile		
参数			
名称	数据类型	参数类型	描述
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

示例：设置 HMI 备份文件

要了解如何设置要恢复到 HMI 的备份文件，请参见 [Restore 方法主题 \(页 145\)](#) 中的示例。

4.13.3.6 SetProgramFolder 方法

使用 SetProgramFolder 方法为 HMI 设备设置用于程序更新 ([页 144](#)) 的文件夹。

返回类型	方法名称		
Result	SetProgramFolder		
参数			
名称	数据类型	参数类型	描述
strPath	string	in	设置存储程序更新文件夹的位置

参数			
名称	数据类型	参数类型	说明
strFolder	string	in	文件夹源位置

示例：设置要更新的程序文件

要了解如何设置用于更新 HMI 的程序文件，请参见 [ProgramUpdate 方法主题 \(页 144\)](#) 中的示例。

4.13.3.7 SetTransferChannel 方法

HMI 设备支持使用多种协议与 API 交换数据。协议即为传输通道。SetTransferChannel 方法设置 HMI 传输通道 [\(页 168\)](#)：

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	识别所选传输通道

默认传输通道为 HMITransferChannel.PN_IE。

示例：设置 HMI 传输通道

```

//-----
//      API 入门指南 \(页 34\)
//-----
//-----
#region    HMI
IHMI myHMI = scannedDevices.FindDeviceByIP(0xC0A80001) as IHMI;
// 192.168.0.1
if (myHMI != null)
{
    //      PN/IE          HMI
    retVal = myHMI.SetTransferChannel(HMITransferChannel.PN_IE);
}
/*           */
/*           */
/*           */
#endregion

```

4.14 IScalance 接口

4.14.1 IScalance 接口

IScalance 接口提供仅适用于 SCALANCE 设备的属性和方法。

ScanNetworkDevices 方法 [\(页 65\)](#) 会生成一个 IProfinetDeviceCollection [\(页 67\)](#)。在该集合中，网络接口上的每台可访问设备都有一个对应的项。这些设备可包括 CPU、HMI、

4.14 IScalance 接口

SCALANCE 和其它设备。IProfinetDevice 接口提供适用于全部设备类别的属性和方法。IScalance 接口继承自 IProfinetDevice，因而支持所有 IProfinetDevice 属性和方法 (页 79)。

要确定设备接口是否表示 SCALANCE 设备，可将 IProfinetDevice 接口 (页 79) 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 148) 和属性 (页 148)。.

示例：确定 PROFINET 设备是否为 SCALANCE 设备

```
//-----
//  API 入门指南 (页 34)
//-----
#region          SCALANCE
IScalance myScalance = scannedDevices.FindDeviceByIP(0xC0A80001) as
IScalance;
// 192.168.0.1
if (myScalance != null)
{
    //-----
    //      SCALANCE .
    //      IScalance
    //-----
}
#endregion
```

4.14.2 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

4.14.3 IScalance 方法

4.14.3.1 SetProfile 方法

调用 SetProfile 方法以建立 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	描述
Profile	ISNMPProfile	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

示例：设置 SNMP 配置文件

可通过以下任意示例查看 SetProfile 方法调用示例：

- 示例：SNMP 版本 1 组态 (页 155)
- 示例：SNMP 版本 2 组态 (页 156)
- 示例：SNMP 版本 3 组态 (页 158)

在这些示例中，SetProfile 调用在接近示例代码末尾的位置。

4.14.3.2 FirmwareUpdate 方法

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	描述
type	FirmwareUpdateType (页 167)	In	可选输入参数，指定要执行的固件更新操作的类型

示例：更新 SCALANCE 设备的固件

可通过以下任意示例查看 FirmwareUpdate 方法调用示例：

- 示例：SNMP 版本 1 组态 (页 155)
- 示例：SNMP 版本 2 组态 (页 156)
- 示例：SNMP 版本 3 组态 (页 158)

这些示例显示了固件更新的两种类型：

- 通过单独的 FirmwareUpdate 方法调用下载并激活固件
- 第一步是通过 FirmwareUpdate 下载固件，第二步是通过 FirmwareActivate 激活固件。

4.14.3.3 FirmwareActivate 方法

FirmwareActivate 方法将激活为硬件标识符为 hardwareID 的指定 Scalance 设备所下载的固件更新文件

返回类型	方法名称
Result	FirmwareActivate

参数			
名称	数据类型	参数类型	描述
hardwareID	uint32	In	SCALANCE 设备的硬件标识符

示例：在 SCALANCE 设备上激活下载的固件

以下示例显示了在下载固件而不激活固件的 FirmwareUpdate (页 149) 后，如何执行 FirmwareActivate。这三个示例分别显示了三种 SNMP 版本类型。

- 示例：SNMP 版本 1 组态 (页 155)
- 示例：SNMP 版本 2 组态 (页 156)
- 示例：SNMP 版本 3 组态 (页 158)

4.15 ISNMPProfile 接口

4.15.1 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。为更新 SCALANCE 设备的固件，必须设置一个 TFTP 服务器。

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)。描述中不含版本号的属性适用于所有 SNMP 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/V3 读取权限 ID 字符串 (PW)
WriteCommunity	string	V1/V2/V3 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

属性名称	返回类型	说明
AuthKey	byte[]	V3 身份认证密钥
PrivAlgorithm	SNMPPrivAlgorithm	V3 隐私算法
PrivKey	byte[]	V3 隐私密钥

4.15.2 ISNMPProfile 方法

4.15.2.1 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

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 [\(页 155\)](#)
- 示例：SNMP 版本 2 组态 [\(页 156\)](#)
- 示例：SNMP 版本 3 组态 [\(页 158\)](#)

在这些示例中，Validate 调用在设置配置文件参数后进行。

4.15.3 SNMPProfile 类

4.15.3.1 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	读取权限 ID 字符串
WriteCommunity	string	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

属性名称	返回类型	说明
PrivKey	byte[]	V3 隐私密钥

4.15.3.2 SNMPProfile 类方法

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 155](#))
- 示例：SNMP 版本 2 组态 ([页 156](#))
- 示例：SNMP 版本 3 组态 ([页 158](#))

在这些示例中，SetProfileName 调用都位于示例代码的开头。

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 171)	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 155](#))
- 示例：SNMP 版本 2 组态 ([页 156](#))
- 示例：SNMP 版本 3 组态 ([页 158](#))

在这些示例中，SetSNMPVersion 调用都位于示例代码的开头。

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 155](#))
- 示例：SNMP 版本 2 组态 ([页 156](#))
- 示例：SNMP 版本 3 组态 ([页 158](#))

在这些示例中，SetServerIP 调用都位于示例代码的开头。

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 155](#))
- 示例：SNMP 版本 2 组态 ([页 156](#))
- 示例：SNMP 版本 3 组态 ([页 158](#))

在这些示例中，SetServerPort 调用在 SetServerIP 调用后立即进行。

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。

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 155](#))
- 示例：SNMP 版本 2 组态 ([页 156](#))
- 示例：SNMP 版本 3 组态 ([页 158](#))

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。

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 155](#))
- 示例：SNMP 版本 2 组态 ([页 156](#))
- 示例：SNMP 版本 3 组态 ([页 158](#))

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 \(页 158\)](#) 代码示例的中间位置查看 SetUserName 方法调用示例。

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 \(页 158\)](#) 代码示例的中间位置查看 SetContextName 方法调用示例。

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 171)	in	SNMP V3 安全等级

可以在 [示例：SNMP 版本 3 组态 \(页 158\)](#) 代码示例的中间位置查看 SetSecurityLevel 方法调用示例。

`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 154](#)) 要求“身份认证”时，身份验证算法适用。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	<code>SetAuthAlgorithm</code>

参数			
名称	数据类型	参数类型	说明
algorithm	SNMPAuthAlgorithm (页 170)	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 \(页 158\)](#) 代码示例的中间位置查看 `SetAuthAlgorithm` 方法调用示例。

`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 154](#)) 要求“隐私”时，隐私算法适用。

返回类型	方法名称
Result	<code>SetPrivAlgorithm</code>

参数			
名称	数据类型	参数类型	说明
algorithm	SNMPPrivAlgorithm (页 170)	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 \(页 158\)](#) 代码示例的中间位置查看 `SetPrivAlgorithm` 方法调用示例。

4.16 IScalance 和 ISNMP 固件更新代码示例

4.16.1 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。

该示例显示了固件更新的两种类型：

- 通过单独的 `FirmwareUpdate` ([页 149](#)) 方法调用下载并激活固件
- 第一步是通过 `FirmwareUpdate` 下载固件，第二步是通过 `FirmwareActivate` ([页 149](#)) 激活固件。

示例：SNMP 版本 1 固件更新

```
//-----
// API 入门指南 (页 34)
```

4.16 IScalance 和 ISNMP 固件更新代码示例

```
//  
//-----  
#region SCALANCE SNMP 1  
  
SNMPProfile profileV1 = new SNMPProfile();  
// SNMP 1  
retval = profileV1.SetSNMPVersion(SNMPVersion.Version1);  
//  
retval = profileV1.SetProfileName("Profile_1");  
// TFTP IP  
retval = profileV1.SetServerIP("192.168.0.1");  
// TFTP 69  
retval = profileV1.SetServerPort(69);  
// SCALANCE "  
retval = profileV1.SetReadCommunity("public");  
// SCALANCE "  
retval = profileV1.SetWriteCommunity("private");  
//  
retval = profileV1.Validate();  
  
// SCALANCE  
IScalance myScalance = scannedDevices.FindDeviceByIP(0xC0A80001) as  
IScalance;  
if (myScalance != null)  
{  
    myScalance.Selected = true;  
    // SCALANCE SNMP  
    retval = myScalance.SetProfile(profileV1);  
    // SCALANCE  
    retval =  
    myScalance.SetFirmwareFile(@"C:\Firmware\FirmwareFile.lad");  
    // -----  
    // SCALANCE  
    //  
    // -----  
    retval = myScalance.FirmwareUpdate();  
    // -----  
    //  
    // SCALANCE  
    //  
    //  
    // -----  
    retval =  
    myScalance.FirmwareUpdate(FirmwareUpdateType.  
DownloadWithoutActivation);  
    // -----  
    // SCALANCE  
    //  
    //  
    // -----  
    retval = myScalance.FirmwareActivate();  
    myScalance.Selected = false;  
}  
/* */  
/* */  
/* */  
#endregion
```

4.16.2 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：

该示例显示了固件更新的两种类型：

- 通过单独的 `FirmwareUpdate` (页 149) 方法调用下载并激活固件
- 第一步是通过 `FirmwareUpdate` 下载固件，第二步是通过 `FirmwareActivate` (页 149) 激活固件。

示例：SNMP 版本 2 固件更新

```

// -----
// API 入门指南 (页 34)
//
// -----
#region    SCALANCE          SNMP      2

SNMPProfile profileV2 = new SNMPProfile();
//      SNMP      2
retval = profileV2.SetSNMPVersion(SNMPVersion.Version2);
//
retval = profileV2.SetProfileName("Profile_2");
//      TFTP      IP
retval = profileV2.SetServerIP("192.168.0.1");
//      TFTP      69
retval = profileV2.SetServerPort(69);
//      SCALANCE      " "
retval = profileV2.SetReadCommunity("public");
//      SCALANCE      " "
retval = profileV2.SetWriteCommunity("private");
//
retval = profileV2.Validate();
//      SCALANCE
IScalance myScalance = scannedDevices.FindDeviceByIP(0xC0A80001) as
IScalance;
if (myScalance != null)
{
    myScalance.Selected = true;
    //      SCALANCE      SNMP
    retval = myScalance.SetProfile(profileV2);
    //      SCALANCE
    retval =
myScalance.SetFirmwareFile(@"C:\Firmware\FirmwareFile.lad");
    // -----
    //      SCALANCE
    //
    // -----
    retval = myScalance.FirmwareUpdate();
    // -----
    //      SCALANCE
    //
    // -----
    retval =
myScalance.FirmwareUpdate(FirmwareUpdateType.
DownloadWithoutActivation);
    // -----
    //      SCALANCE
    //
}

```

```

//  

//  

// -----  

    retVal = myScalance.FirmwareActivate();  

    myScalance.Selected = false;  

}  

/*           */  

/*           */  

/*           */  

#endifregion

```

4.16.3 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：

该示例显示了固件更新的两种类型：

- 通过单独的 FirmwareUpdate (页 149) 方法调用下载并激活固件
- 第一步是通过 FirmwareUpdate 下载固件，第二步是通过 FirmwareActivate (页 149) 激活固件。

示例：带有身份验证和隐私的 SNMP 版本 3 固件更新

```

//-----  

//      API 入门指南 (页 34)  

//  

//-----  

#region      SCALANCE          SNMP      3  
  

SNMPProfile profileV3 = new SNMPProfile();  

//      SNMP      3  

retVal = profileV3.SetSNMPVersion(SNMPVersion.Version3);  

//  

retVal = profileV3.SetProfileName("Profile_3.1");  

//      TFTP      IP  

retVal = profileV3.SetServerIP("192.168.0.1");  

//      TFTP      69  

retVal = profileV3.SetServerPort(69);  

//      SCALANCE      "  

retVal = profileV3.SetReadCommunity("public");  

//      SCALANCE      "  

retVal = profileV3.SetWriteCommunity("private");  
  

//  

retVal = profileV3.SetUserName("User1");  

//  

retVal = profileV3.SetContextName("Context1");  

//      SCALANCE      AuthPriv  

retVal = profileV3.SetSecurityLevel(SNMPSecurityLevel.AuthPriv);  

//      MD5  

retVal =  

profileV3.SetAuthAlgorithm(SNMPAuthAlgorithm.MD5, "Password1");  
  

//      DES  

retVal =  

profileV3.SetPrivAlgorithm(SNMPPrivAlgorithm.DES, "Password2");  

//  

retVal = profileV3.Validate();

```

```

//      SCALANCE
IScalance myScalance = scannedDevices.FindDeviceByIP(0xC0A80001) as
IScalance;
if (myScalance != null)
{
    myScalance.Selected = true;
    //      SCALANCE          SNMP
    retVal = myScalance.SetProfile(profileV3);
    //      SCALANCE
    retVal =
myScalance.SetFirmwareFile(@"C:\Firmware\FirmwareFile.lad");
    // -----
    //      SCALANCE
    //
    // -----
    retVal = myScalance.FirmwareUpdate();
    // -----
    //
    //      SCALANCE
    //
    //
    // -----
    retVal =
myScalance.FirmwareUpdate(FirmwareUpdateType.
DownloadWithoutActivation);
    // -----
    -
    //      SCALANCE
    //
    //
    //
    // -----
    retVal = myScalance.FirmwareActivate();
    myScalance.Selected = false;
}
/*
 */
/*
 */
#endregion

```

不带身份验证和隐私的 SNMP 版本 3 固件更新

要将示例修改为不使用身份验证和隐私，移除用于设置安全等级、身份验证算法和权限算法的代码。将删除的代码替换为以下代码：

```
//      SCALANCE          NoAuthNoPriv
retVal = profileV3.SetSecurityLevel(SNMPSecurityLevel.NoAuthNoPriv);
```

带身份验证但不带隐私的 SNMP 版本 3 固件更新

要将示例修改为使用身份验证但不使用隐私，移除用于设置安全等级、身份验证算法和权限算法的代码。将删除的代码替换为以下代码：

```
//      SCALANCE          AuthNoPriv
retVal = profileV3.SetSecurityLevel(SNMPSecurityLevel.AuthNoPriv);
//          MD5
```

```
        retVal =
profileV3.SetAuthAlgorithm(SNMPAuthAlgorithm.MD5, "Password1");
```

4.17 异常

4.17.1 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。

```
#region Critical internal error exception
try
{
    //-----
    // API 入门指南 (页 34)
    //
    //-----
    ICPU myCPU = scannedDevices.FindDeviceByIP(0xC0A80001) as ICPU;
    // 192.168.0.1
    if (myCPU != null)
    {
        myCPU.SetPassword(new EncryptedString("Password"));
        myCPU.Selected = true;
        if (myCPU.Failsafe)
            myCPU.SelectedConfirmed = true;
        retVal = myCPU.ResetToFactoryDefaults();
    }
}
catch (CriticalInternalErrorException e)
{
    // API
    //
    myCPU.Selected = false;
    myCPU.SelectedConfirmed = false;
}
catch (Exception e)
{
    // API
    myCPU.Selected = false;
    myCPU.SelectedConfirmed = false;
}
/*
 */
/*
 */
/*
 */
#endregion
```

4.18 API 枚举

4.18.1 BackupOptions

BackupOptions 枚举定义用于创建和覆盖文件的选项：

```
OverWriteFile  
DoNotOverWriteFile  
CreateUniqueFile
```

4.18.2 BackupType

BackupType 枚举定义用户可执行的备份类型：

```
Invalid  
FullBackup  
Recipes  
UserAdministration
```

4.18.3 CertificateStatusError

CertificateStatusError 枚举包含可能的证书错误类型：

```
None  
CertStatusAuthorityInvalid  
CertStatusRevoked  
CertStatusDateInvalid  
CertStatusIPAddress  
CertStatusInvalid
```

4.18.4 ConfigurationDataProtection

ConfigurationDataProtection 枚举描述了 CPU 中数据保护组态的状态。其中包含以下值：

```
ProtectionUnknown  
NoProtection_CPUProgramMatch  
NoProtection_CPUProgramMismatch  
NoProtection_NoCPUProgram  
ProtectionEnabled_CPUProgramMatch  
ProtectionEnabled_CPUProgramMismatch  
ProtectionEnabled_NoCPUProgram
```

4.18.5 ConfigurationStatus

ConfigurationStatus 枚举描述 CPU 的组态状态。其中包含以下值：

```
NotAvailable  
DiagnosticError  
NotConfigured  
Match  
Mismatch
```

4.18.6 ConfirmationType

此枚举用于指示故障安全 CPU 的状态：

```
Invalid  
SafetyPasswordIsBeingUsed  
DeletingExistingSafetyProgram  
ReplacingExistingSafetyProgram  
ReplacingExistingSafetyProgramWithNonSafetyProgram  
LoadingSafetyProgram
```

4.18.7 DataChangedType

此枚举定义了 DataChangedEventHandler (页 98) 的可能参数值：

```
Invalid  
OperatingState  
RackInformation  
Folders  
File  
ProfinetName  
IPAddress  
Password  
FileSystem  
StopModeTransition  
RefreshStatus
```

4.18.8 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  
SinamicsDrive  
SoftStarter  
CommunicationsModule  
OpticalReader  
RFID
```

4.18.9 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
NotChangeableInErrorState
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
ParameterOutOfRange
FailedToZipFolderContents
ErrorWritingToFile
ErrorCreatingFile
ErrorCreatingFolder
InvalidTimeoutValue
NoDataToBackup
ErrorWritingToStream
ErrorReadingFromStream
InvalidProjectPath
ProjectNotCompatibleWithDevice
FailedToSetProfinetName
FailedToSetIPAddress
DownloadInvalidRecipe
IdentityFailure
DeviceMismatch
InvalidInterface
DeviceNotSelected
FailsafeAccessRequired
InternalApplicationError
InvalidPassword
DuplicateIPAddress
DuplicateProfinetName
```

```
SafetyDeviceMustBeConfirmed
NoSDCardPresent
InvalidProgramFolder
FSignaturesDoesNotMatch
FSignaturesMatch
DeviceDoesNotSupportProject
ProjectsUpdateIPNotReachable
RestoreIPNotReachable
ProjectIPNotUnique
SafetyProjectDownloadedToStandardNotAllowed
PasswordDiversityFailed
InvalidBackupFile
InvalidBackupFileExtension
IncompatibleBackupFile
InvalidFirmwareFile
OperationWasNotSuccessful
CouldNotValidatePassword
IPAddressAlreadyExistsOnNetwork
MissingProgramFilePassword
InvalidProgramFilePassword
OperationCanceledByUser
InvalidProgramForDevice
InvalidProgramFilePasswordLegitimizationLevel
DeviceNotFound
DeviceAlreadyExists
IPAddressAlreadyOnNetwork
ProfinetNameAlreadyOnNetwork
FailedToConnect
DeviceNotInitialized
CPUNewerVersionNotSupported
IPSuitNotValid
IPAddressChanged
ScanNoDevicesFound
DeviceCannotBeInserted
InsertDeviceDuplicateIP
IPNotReachable
CouldNotReadFSignature
InvalidNetworkInterface
InsufficientLegitimizationLevel
NoProgramPassword
ProjectVersionV1NotSupported
ProjectOpenCanceled
ProgramPasswordNeeded
RestoreError
IncompatibleProgramFile
UnsupportedProgramFile
ProgramFileFamilyMismatch
DuplicateNewIPAddress
SNMPErrorNoAccess
SNMPErrorReadOnly
SNMPErrorNotWritable
SNMPErrorAuthorizationError
SNMPError
InvalidProfileName
InvalidSNMPVersion
InvalidServerIP
InvalidServerPort
InvalidReadCommunity
InvalidWriteCommunity
InvalidUserName
InvalidContextName
InvalidSecurityLevel
InvalidAuthAlgorithm
```

```
InvalidAuthPassword
InvalidPrivAlgorithm
InvalidPrivPassword
InvalidProfile
ProfileNameAlreadyExists
ObsoleteMethod
FailedToInitiateFirmwareTransfer
DeviceDefinedError
ErrorDeletingFile
ErrorDeletingFolder
ErrorInvalidMAC
FailedToUpdateDuplicates
DuplicateNewProfinetName
ErrorBackingupData
FailsafeControlObjectIncorrectType
InvalidIPAddressOrUsedByNIC
FirmwareIntegrityFailed
ExportDiagnosticsBufferError
AlmFailedInitialize
AlmFailedCleanup
AlmFailedSessionInitialize
AlmFailedSessionCleanup
AlmSessionIDMissing
AlmSessionIDUnknown
AlmCouldNotConnect
AlmOutOfMemory
AlmOperationTimeout
AlmFunctionNotFound
AlmAborted
AlmBadfunctionArgument
AlmUnkownOption
AlmSendError
AlmReceiveError
AlmNoConnectionAvailable
AlmOpenSession
AlmResources
AlmService
AlmCryptography
AlmTaskAlreadyRunning
AlmInvalidPointer
AlmResultMismatch
AlmBadResult
AlmBatchWrongArgumentNumber
AlmBatchWrongArgument
AlmBatchWrongInputFile
AlmBatchWrongInputStream
AlmBatchAPILoadFailed
AlmBatchOutputfileExists
AlmBatchWrongOutputParmater
AlmBatchOutputCreationFailure
AlmBatchAccessDenied
AlmUnknownError
PowerCycleRequired
DuplicateFolders
DuplicateFiles
FeatureNotSupportedInTrial
APIFeatureNotSupported
CPUProtectionLevelWeaker
CouldNotReadProtectionLevel
ChangingIPNotAllowedNatRouter
IPAndRouterIPCannotBeTheSame
OperationNotAllowedThroughCPCM
FolderOrFileDoesNotExist
```

```
DCPOperationNotSupportedBehindNATRouters
OperationNotSupportedThroughCPCM
CPUFailedToEnterStopMode
NoFirmwareToActivate
FailedToActivateFirmware
InvalidOperationException
ErrorRefreshingFolder
ErrorCannotRenameFileAlreadyExists
FileWriteProtected
FolderWriteProtected
MemoryCardNotPresent
OperationRequiresStopMode
ErrorWritingFileMemoryCard
ErrorCreatingFolderMemoryCard
ErrorReadingServiceData
NoFilesInListToDelete
InsufficientLegitimizationLevelRefreshStatus
APIFeatureAdvancedNotSupported
TLSConnectionTrustRequired
TLSCertificateDisabled
TLSCertificateEnabled
TLSCertificateEnabledBootstrap
ConfigurationDataProtectionDoesNotMatch
ErrorInvalidMemoryCard
ErrorInvalidProgramFolder
InvalidScanFilter
CPUGeneratedCertificate
MultipleInterfacesWarning
DeviceDoesNotSupportDeleteOption
OnlineOfflineCompareFailed
PreconditionsFAddressAssignmentFailed
FAddressParameterizationCRCFailed
FailedEstablishConnection
FailedEstablishConnectionSNMP
NotConnectedToFCPU
OperationNotSupportedInRunRedundant
FailedComparingActualToConfiguredHardware
IPAndRouterIPAreTheSame
DeviceCannotBeReplaced
SameDeviceCannotBeReplaced
APIAvailableForSATOnly
FailedToIdentifyDevice
BackupFileAlreadyExists
FirmwareVersionNotSupported
RouterIPAddress IsNotValid
DeviceNotProtected
IPAndRouterIPMustBeDifferent
UnexpectedOperatingSystemError
ServiceActive
RemoteTransferDisabled
HardwareSoftwareNotComplete
LogicalVolumeMissing
LogicalVolumeOutOfSpace
Abort
FirmwareTypeNotSupported
FirmwareTypeNotInstalled
StoreReadFailed
StoreWriteFailed
RescueBackupNotPossible
RescueRestoreNotPossible
ConnectionRequired
ObjectNotFound
BufferTooSmall
```

```
InvalidArguemnts
AttributeNotFound
InvalidPath
TypeConversionFailed
FileReadFailed
FileWriteFailed
OutOfResources
OutOfSpace
UnknownAddon
IncompatibleAddon
AddonsUnsupported
UnknownApp
UnknownAppAddon
UnknownReferenceApp
RuntimeMissing
RuntimeBroken
SignatureRequired
SignatureInvalid
SignatureFailure
CertificateInvalid
CertificateFailure
CertificateNotReady
CertificateExpired
CertificateRevoked
SecurityLib
WrongRuntimeVersion
MajorRuntimeDowngrade
MajorRuntimeUpgrade
MajorImageDowngrade
MajorImageUpgrade
WrongRuntime
NotEnoughMemory
ProjectCharacteristicsMissing
ProjectCharacteristicsInvalid
PanelOrientationIsPortrait
PanelOrientationIsLandscape
WrongDevicetype
NoRuntimeInstalled
RuntimeCorrupt
InvalidTransferChannel
InvalidBackupType
HMIUnknownError
```

4.18.10 FailsafeOperation

FailsafeOperation 枚举表示与安全相关的操作：

```
Invalid
ResetToFactoryOperation
FormatMCOperation
ProgramUpdateOperation
RestoreOperation
```

4.18.11 FirmwareUpdateType

FirmwareUpdateType 指定要执行的固件更新操作的类型：标准固件更新或两步固件更新。第一步是下载固件更新文件。第二步是激活设备上的固件更新。

```
Invalid  
DownloadWithActivation  
DownloadWithoutActivation  
DownloadWithoutActivationInStopMode
```

4.18.12 HMIBackupType

HMIBackupType 枚举定义用户可执行的 HMI 备份类型：

```
FullBackup  
Recipes  
UserAdministration
```

4.18.13 HMITransferChannel

HMITransferChannel 枚举指示用于与 HMI 设备通信的协议：

```
Invalid  
PN_IE  
Ethernet
```

4.18.14 Language

Language 枚举允许用户为返回的字符串数据指定语言：

```
German  
English  
Spanish  
French  
Italian  
Chinese
```

4.18.15 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
```

4.18.16 OperatingStateREQ

此枚举定义了在调用 SetOperatingState ([页 129](#)) 方法时可请求的可能状态转换：

Stop
Run

4.18.17 ProgressAction

此枚举定义了可被发送至 ProgressChangedEventHandler (页 99) 的可能参数值：

Invalid
Connecting
Reconnecting
Disconnecting
Initializing
Updating
Processing
Downloading
Uploading
Deleting
Resetting
Rebooting
Verifying
Formatting
Refreshing
Finished
UpdatingFirmware
InstallingRuntime
InstallingAddOns
UninstallingAddOns
UpdatingProgram
Renaming
Creating
FileReplaceOrSkip
FileWriteProtected
FolderWriteProtected
ChangePassword

4.18.18 ProtectionLevel

ProtectionLevel 枚举列出了 CPU 密码的保护等级：

Unknown
Failsafe
Full
Read
HMI
NoAccess
NoPassword

4.18.19 RedundancyRole

此枚举定义了 RedundancyRole 属性的可能状态：

NotValid
Primary
Backup

4.18.20 RemoteFolderType

RemoteFolderType 枚举表示远程文件夹类型：

- None
- Recipe
- Datalog
- Files

4.18.21 RemoteInterfaceType

此枚举定义了在 IRemoteInterfaces ([页 136](#)) 接口上调用 InterfaceType 属性时可以返回的可能状态：

- None
- Profinet
- Profibus
- Asi

4.18.22 ResetOptions

ResetOptions 枚举包含用于复位出厂默认设置以及格式化存储卡的选项：

- None
- DeleteIPAddress
- DeleteConfigurationDataProtectionPassword
- FormatMemoryCard

4.18.23 ScanErrorType

ScanErrorType 枚举指示设备扫描返回的错误类型：

- Invalid
- Success
- Error
- Warning
- Information

4.18.24 SDCardType

SDCardType 枚举包含 SD 卡的特性：

- Unknown
- NotPresent
- ReadOnly
- ReadWrite

4.18.25 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：

- NotSupported
- MD5
- SHA

4.18.26 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：

- NotSupported
- DES
- AES

4.18.27 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：

- NotSupported
- NoAuthNoPriv
- AuthNoPriv
- AuthPriv

4.18.28 SNMPVersion

SNMPVersion 枚举表示 SCALANCE 设备的 SNMP 版本号：

- NotSupported
- Version1
- Version2
- Version3

4.18.29 TimeFormat

TimeFormat 用于指定时间字符串的时间格式：

- Local
- UTC

说明

UTC 表示协调世界时。

4.18.30 TrustCertificateType

TrustCertificateType 枚举定义证书信任类型：

- “从不”(Never)
- “始终”(Always)
- “需要选择”(SelectionNeeded)

4.19 CPU 密码访问级别

标准 CPU 有四个密码访问级别。故障安全 CPU 有五个密码访问级别。

在 TIA Portal 中，可在 CPU 设备“属性”(Properties) 的“保护和安全”(Protection & Security) 部分查看 CPU 访问级别：

选择该 PLC 的存取等级。

访问级别	访问				访问权限
	HMI	读取	写入	故障安	
<input checked="" type="radio"/> 完全访问权限，包括故障安全 (无...)	✓	✓	✓	✓	*****
<input type="radio"/> 完全访问权限 (无任何保护)	✓	✓	✓		*****
<input type="radio"/> 读访问权限	✓	✓			*****
<input checked="" type="radio"/> HMI 访问权限	✓				
<input type="radio"/> 不能访问 (完全保护)					

HMI 访问权限：
TIA Portal 用户将无法访问标准功能和故障安全功能。
而 HMI 应用则可以访问所有功能（故障安全和标准功能）。

必填密码：
要额外具有读 / 写访问权限和访问故障安全功能的权限。TIA Portal 用户需要输入“所有权限，包括故障安全”的密码。

可选密码：
要额外具有对标准功能（但不包含故障安全功能）的读/写访问权限。
需要定义一个“读/写访问权限”或者“读访问权限”的密码。

需要读访问权限的设备操作需要能提供“读”访问权限的任何访问级别。

需要写访问权限的设备操作需要能提供“写”访问权限的任何访问级别。

有关访问级别和密码的其它信息，请参见 STEP 7 信息系统 (TIA Portal 的在线帮助)。

F-CPU 上与安全相关的操作的密码要求

您可以使用读或写访问权限在故障安全 CPU 上执行某些操作。与安全相关的操作需要提供安全密码，其对应于“包括故障安全在内的完全访问权限（无保护）”访问级别。

与安全相关的操作包括：

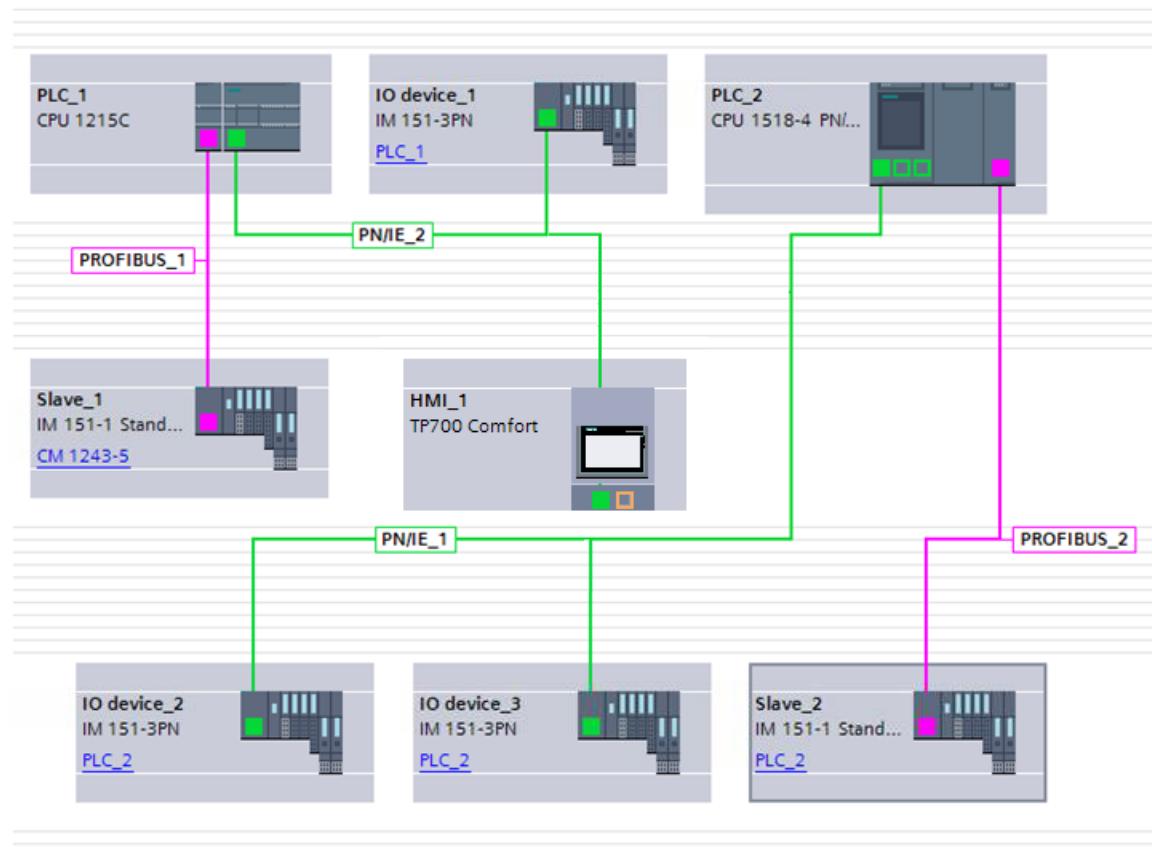
- 程序更新
- 从备份中恢复
- 复位为出厂默认值
- 格式化存储卡

如果故障安全 CPU 不使用密码保护，则与安全相关的操作不需要使用安全密码来启动操作。

在本用户指南中，“安全密码”一词指的是具有“包括故障安全在内的完全访问权限（无保护）”访问级别的密码。

4.20 网络示例

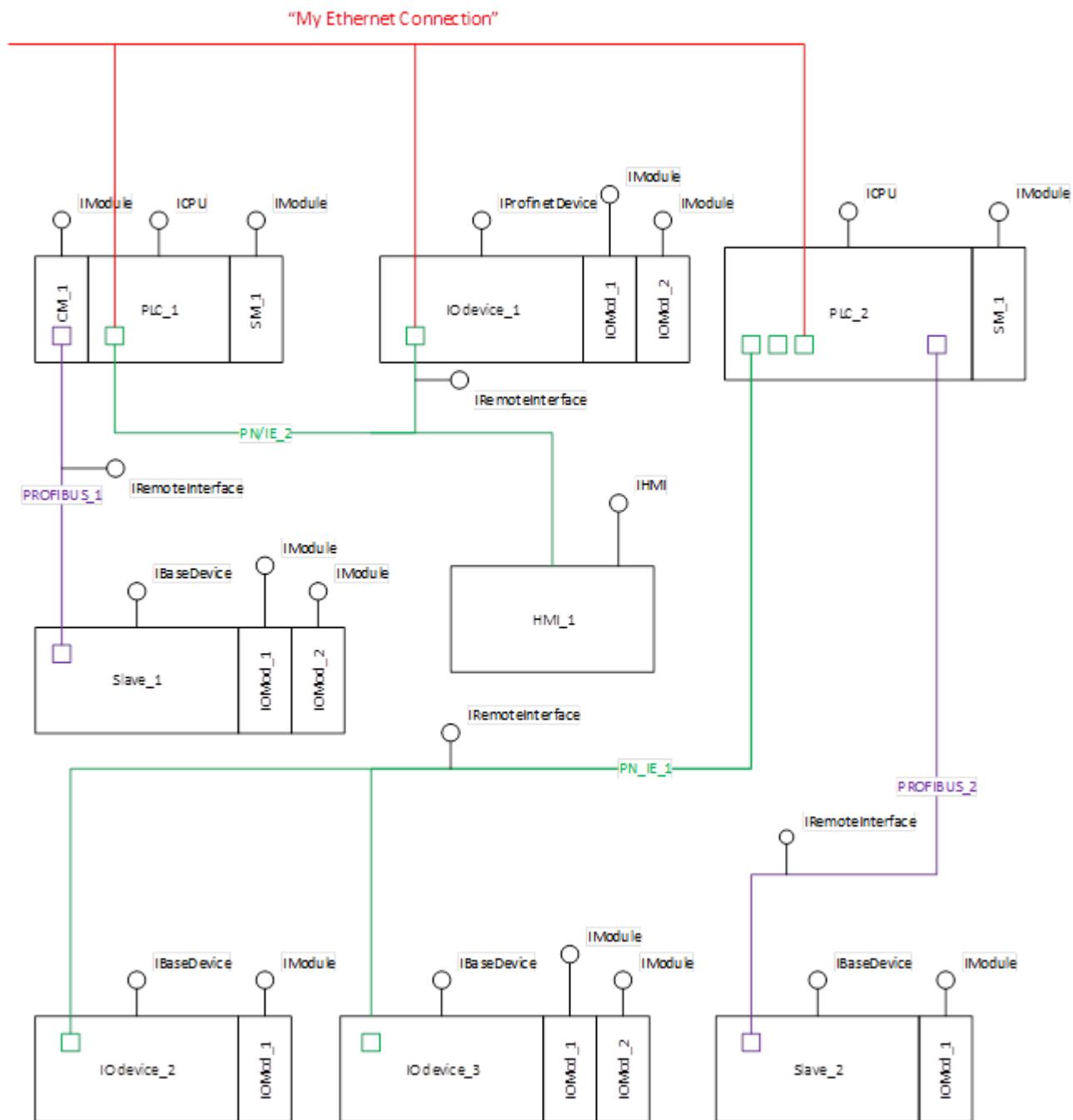
此示例显示了 TIA Portal 网络组态和表示已联网设备的 API 接口：



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 PROFINET 网络（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 (页 101) 表示。
- 直接与外部网络连接的 HMI 由 `IHMI` 接口 (页 140) 表示。
- 直接与外部网络连接的 I/O 站由 `IProfinetDevice` 接口 (页 79) 表示。
- 源自 CPU 的子网由 `IRemoteInterface` 接口 (页 56) 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 (页 58) 表示。
- 连接至 CPU 或 I/O 站的 I/O 或通信模块由 `IModule` 接口 (页 100) 表示。

4.21 参考信息

4.21.1 关于路由器后面的设备

网络扫描找不到路由器后面的设备。要将设备插入路由器后面，可通过输入设备的 IP 地址将其插入 (页 76)。

NAT 路由器打开端口的要求

要使用 NAT 路由器后面的设备，必须打开以下端口（取决于具体设备）。

设备类型	需要打开的端口
CPU	102 161（可选）
具有以太网传输通道 (页 147) 的 HMI	5001 161
具有 PN/IE 传输通道 (页 147) 的 HMI	102 161
SCALANCE	161

IP 和 NAT 路由器后面的设备的限制

路由器后面的设备不支持以下 DCP（发现和组态协议）操作：

- 设置 IP 地址（包括子网和网关） (页 94) (Set IP Address (including Subnet and Gateway))
- 设置 PROFINET 名称 (页 95) (Set PROFINET Name)
- 识别设备 (页 88) (Identify device)
- 复位通信参数 (页 93) (Reset Communication Parameters)

如果路由器后面的设备通过 CM（通信模块）或 CP（通信处理器）连接到网络，则存在额外的限制 (页 175)。

程序更新 (页 120) 和恢复 (页 124) 操作对 NAT 路由器后面的设备有额外的限制。

4.21.2 关于连接到 CM 或 CP 的设备

API 可以通过 CM (通信模块) 或 CP (通信处理器) 连接与 CPU 通信。要启用此类型通信, 请按以下步骤操作 :

1. 从 TIA Portal 中, 在 STEP 7 项目中打开 CM 或 CP 的设备组态。
2. 在“属性”(Properties) 的“通信类型”(Communication types) 设置中, 选择“启用在线功能”(Enable online functions)。
3. 在属性的“SNMP”设置中, 选择“启用 SNMP”(Enable SNMP)。
4. 将 STEP 7 项目下载到 CPU 中。

您可以将 PROFINET 网络连接到 CM 或 CP 的以太网接口。API 将通过网络扫描 ([页 65](#)) 找到所连接的 CPU。

通过 CM 或 CP 连接的 CPU 的属性

通过 CM 或 CP 的以太网接口与 CPU 通信时, IProfinetDevice 属性 ([页 79](#)) 中的 MAC 地址和 IP 地址是 CM 或 CP 的 MAC 地址和 IP 地址。IProfinetDevice 属性 ([页 79](#)) 中的序列号和所有其它数据均为 CPU 中的数据。

限制

API 无法对通过 CM 或 CP 连接的 CPU 执行以下设备操作 :

- 程序更新 ([页 120](#))
- 固件更新 ([页 82](#))
- 备份 ([页 106](#))
- 还原 ([页 124](#))
- 复位为出厂默认值 ([页 123](#))
- 格式化存储卡 ([页 116](#))

索引

A

API (应用程序编程接口)

- 示例程序, 11
- 创建和组态项目, 11
- 选择并设置网络接口, 20
- 连接到网络上的 CPU, 21
- 切换 CPU 的操作状态, 27
- 设置或更改 CPU 的 IP 地址、子网和网关, 28
- 读取和写入 PROFINET 名称, 29
- 更新 CPU 固件, 30
- 入门指南, 34
- 架构概述, 39
- 文件和安装, 40
- 版本兼容性, 41
- S7 通信, 41
- 提供的安全功能, 42
- 设计 UI, 43

B

BackupOptions (API 枚举), 160
BackupType (API 枚举), 161
Backup (API 方法, ICPU 接口), 106
Backup (API 方法, IHMI 接口), 143

C

CertificateStatusError (API 枚举), 161
Clear (API 方法), 78
ConfigurationDataProtection (API 枚举), 161
ConfigurationStatus (API 枚举), 161
ConfirmationType (API 枚举), 161
CopyUserData (API 方法), 78
CPU
 密码访问级别, 171
CreateConfigurationDataProtectionCard (API 方法), 107
CreateFirmwareUpdateCard (API 方法), 108
CreateProgramUpdateCard (API 方法), 109
CriticalInternalErrorException, 160

CurrentNetworkInterface (API 属性), 65

D

DataChangedEventArgs (API 类), 53
DataChangedType (API 枚举), 162
DataChanged (API 事件), 98
DeleteConfigurationDataProtectionPassword (API 方法), 110
DeleteDataLog (API 方法), 110
DeleteRecipe (API 方法), 111
DetermineConfirmationMessage (API 方法), 112
DeviceFamily (API 枚举), 162
DiagnosticsItem (API 类), 53
DownloadRecipe (API 方法), 115

E

EncryptedString (API 类), 50
ErrorCode (API 枚举), 162
ExportDeviceDiagnostics (API 方法), 74
ExportDeviceInformation (API 方法), 74
ExportProgressEventArgs (API 类), 54

F

FailSafeOperation (API 枚举), 167
FilterByDeviceFamily (API 方法), 71
FilterOnlyCPUs (API 方法), 71
FindDeviceByIP (API 方法), 71
FindDeviceByMAC (API 方法), 72
FindDevicesBySerialNumber (API 方法), 69
FirmwareActivate (API 方法, IProfinetDevice 接口), 85
FirmwareActivate (API 方法, IScalance 接口), 150
FirmwareUpdateType (API 枚举), 167

- FirmwareUpdate (API 方法, IHMI 接口) , 144
 FirmwareUpdate (API 方法, IProfinetDevice 接口) , 82
 FirmwareUpdate (API 方法, IScalance 接口) , 149
 FormatMemoryCard (API 方法) , 116
- G**
- GetCommunicationsTimeout (API 方法) , 66
 GetCurrentDateTime (API 方法) , 118
 GetDiagnosticsBuffer (API 方法, ICPUClassic 接口) , 140
 GetDiagnosticsBuffer (API 方法, ICPU 接口) , 119
 GetEmptyCollection (API 方法) , 67
 GetEnumerator (API 方法) , 69
- H**
- HMIBackupType (API 枚举) , 168
 HMITransferChannel (API 枚举) , 168
- I**
- IBaseDevice (API 接口) , 58
 ICertificateStore (API 接口) , 63
 ICertificate (API 接口) , 63
 ICPUClassic (API 接口) , 139
 ICPU (API 接口) , 102
 ICPU (API 接口)
 属性, 102
 程序更新标志, 104
 恢复标志, 105
 功能标志, 105
 Identify (API 方法) , 88
 IDiagnosticBuffer (API 接口) , 62
 IHardwareCollection (API 接口) , 58
 IHardware (API 接口) , 57
 IHMI (API 接口) , 141
 属性, 141
 程序更新标志, 142
 恢复标志, 142
 功能标志, 142
 IModuleCollection (API 接口) , 59
 IModuleCollection (API 接口) , 使用, 100
- IModule (API 接口) , 101
 IModule (API 接口)
 Modules 属性和 IModuleCollection 类, 100
 InsertDeviceByIP (API 方法) , 76
 InsertDeviceByMAC (API 方法) , 77
 IProfinetDevice (API)
 集合项, 70
 属性, 79
 IProfinetDeviceCollection 类别 (API)
 迭代集合项目, 68
 IProfinetDeviceCollection (API 类)
 FindDevicesBySerialNumber 方法, 69
 GetEnumerator 方法, 69
 SortByIP 方法, 69
 IProfinetDeviceCollection (API 类)
 [] 特性, 70
 IRemoteFile (API 接口) , 56
 IRemoteFolder (API 接口) , 56
 IRemoteInterface (API), 属性, 137
 IRemoteInterface (API 接口) , 56
 IResult (API 接口) , 62
 IScalance (API 接口) , 147-148
 属性, 148
 方法, 148
 IScanErrorCollection (API 类) , 59
 IScanErrorEvent (API 类) , 61
 IsFirmwareUpdate2StepAllowed (API 方法, IProfinetDevice 接口) , 89
 IsFirmwareUpdateAllowed (API 方法, IProfinetDevice 接口) , 88
 IsIPAddressOnNetwork (API 方法, IProfinetDevice 接口) , 90
 ISNMPProfile (API 接口)
 属性, 150
 方法, 151
 IsProfinetNameOnNetwork (API 方法, IProfinetDevice 接口) , 90
 IsUploadServiceDataAllowed (API 方法, IProfinetDevice 接口) , 91

L

Language (API 枚举) , 168

M

MemoryReset (API 方法) , 120

N

Network 构造函数 (API) , 64

O

OperatingStateReq (API 枚举) , 168

OperatingState (API 枚举) , 168

P

PROFINET 名称, 读取和写入, 29

ProgramUpdate (API 方法, ICPU 接口) , 120

ProgramUpdate (API 方法, IHMI 接口) , 144

ProgressAction (API 枚举) , 169

ProgressChangedEventArgs (API 类) , 54, 55

ProgressChanged (API 事件) , 99

ProtectionLevel (API 枚举) , 169

Q

QueryNetworkInterfaceCards (API 方法) , 64

R

ReadFromStream (API 方法) , 73

RedundancyRole (API 枚举) , 169

RefreshStatus (API 方法) , 92

RemoteFolderType (API 枚举) , 170

RemoteInterfaces (API) , 136

RemoteInterfaceType (API 枚举) , 170

Remove (API 方法) , 78

ResetCommunicationParameters (API 方法, IProfinetDevice 接口) , 93

ResetOptions (API 枚举) , 170

ResetToFactoryDefaults (API 方法) , 123

Restore (API 方法, ICPU 接口) , 125

Restore (API 方法, IHMI 接口) , 145

Result (API 类) , 51

S

S7-300 和 S7-400, 通过 API 访问, 139

ScanErrorType (API 枚举) , 170

ScanNetworkDevices (API 方法) , 65

SDCardType (API 枚举) , 170

SetAuthAlgorithm (API 方法、SNMPPProfile 类) , 155

SetBackupFilePassword (API 方法) , 128

SetBackupFile (API 方法, ICPU 接口) , 127

SetBackupFile (API 方法, IHMI 接口) , 146

SetCommunicationsTimeout (API 方法) , 66

SetConfigurationDataProtectionPassword (API 方法) , 128

SetContextName (API 方法、SNMPPProfile 类) , 154

SetCurrentDateTime (API 方法) , 129

SetCurrentNetworkInterface (API 方法) , 64

SetIP (API 方法) , 94

SetOperatingState (API 方法) , 129

SetPassword (API 方法) , 130

SetPrivAlgorithm (API 方法、SNMPPProfile 类) , 155

SetProfileName (API 方法、SNMPPProfile 类) , 152

SetProfile (API 方法, IScalance 接口) , 148

SetProfinetName (API 方法) , 95

SetProgramFolder (API 方法, ICPU 接口) , 131

SetProgramFolder (API 方法, IHMI 接口) , 146

SetProgramPassword (API 方法) , 132

SetReadCommunity (API 方法、SNMPPProfile 类) , 153

SetSecurityLevel (API 方法、SNMPPProfile 类) , 154

SetServerIP (API 方法、SNMPPProfile 类) , 152

SetServerPort (API 方法、SNMPPProfile 类) , 153

SetSNMPVersion (API 方法、SNMPPProfile 类) , 152

SetTransferChannel (API 方法, IHMI 接口) , 147

SetTrustCertificateStore (API 方法) , 132

SetUserName (API 方法、SNMPProfile 类) , 154
SetWriteCommunity (API 方法、SNMPProfile 类) , 153
SNMPAuthAlgorithm (API 枚举) , 170
SNMPPrivAlgorithm (API 枚举) , 171
SNMPProfile (API 类)
 属性, 151
 方法, 152
SNMPSecurityLevel (API 枚举) , 171
SNMPVersion (API 枚举) , 171
SNMP 版本 1, API 示例, 155
SNMP 版本 2, API 示例, 157
SNMP 版本 3, API 示例, 158
SortByIP (API 方法) , 69

T

TimeFormat (API 枚举) , 171
TrustCertificateType (API 枚举) , 171

U

UI 开发中的颜色编码安全域, 44
CPU 设备图标, 45
设备数据, 46
CPU 密码, 47
程序文件夹, 48
程序密码, 49
UploadDataLog (API 方法) , 133
UploadRecipe (API 方法) , 134
UploadServiceData (API 方法) , 96

V

ValidateIPAddressSubnet (API 方法) , 97
ValidateNetworkInterface (API 方法) , 67
ValidatePROFINETName (API 方法) , 97
ValidateProgramFolder (API 方法) , 135
Validate (API 方法, ISNMPProfile 接口) , 151

W

WriteToStream (API 方法) , 73

创

创建自定义应用程序, 11

第

第三方软件许可条件与版权, 31

分

分散式模块, 通过 API 访问, 136

固

固件更新, 30

汉

汉明码, 49-50

将

将应用程序分发给您的最终用户, 31

将项目插入到集合中 (API) , 76

将项目添加到集合中 (API) , 76

经

经典 CPU, 通过 API 访问, 139

连

连接到网络上的 CPU, 21

密

密码, CPU 访问级别, 171

识

识别 CPU、API, [102](#)

使

使用 API 的编程指南, [43](#)

使用 API 进行 UI 设计中的安全相关操作, [42](#), [43](#)

示

示例程序

打开并运行, [11](#)

创建和组态项目, [11](#)

设置代码, [12](#)

辅助方法, [13](#)

选择语言, [18](#)

选择并设置网络接口, [20](#)

连接到网络上的 CPU, [21](#)

显示 CPU 和模块信息, [25](#)

切换 CPU 的操作状态, [27](#)

设置或更改 CPU 的 IP 地址、子网和网关, [28](#)

读取和写入 PROFINET 名称, [29](#)

更新 CPU 固件, [30](#)

示例网络, [173](#)

示例, SNMP 配置文件

版本 1, [155](#)

版本 2, [157](#)

版本 3, [158](#)

网

网络接口, 选择, [20](#)

网络示例, [173](#)

许

许可条款和条件, [41](#)