SIEMENS

SINUMERIK 840D sl / One / MCU1720

Programming Manual **02/2024**

SINUMERIK Integrate Create MyHMI /3GL (.NET Api)

Manufacturer Documentation

SIEMENS

SINUMERIK 840D sl / One / MCU1720

SINUMERIK Integrate Create MyHMI /3GL (.NET Api)

Programming Manual

Valid for

Software Version 4.95 SP4

02/2024

Introduction 1

Getting started 2

Communication with the NC/PLC 3

Program instance services (PI services) 4

Access to alarms and events 5

Directory and file services 6

Infrastructure services 7

Archive service 8

Drive machine data service 9

TraceDataRecorder 10

ToolManagement service 11

Action log service 12

COM clients 13

Display Manager 14

Index I

SINUMERIK® documentation

Edition history

Brief details of this edition and previous editions are listed below.

The status of each edition is shown by the code in the "Remarks" columns.

Status code in the "Remarks" column:

A .... New documentation.
B .... Unrevised reprint with new order number.
C .... Revised edition with new revision level.

Edition	Order No.	Remark
02/2024	-----	**A**

Trademarks

All names identified by the trademark symbol ® are registered trademarks of Siemens AG. Other product names used in this documentation may be trademarks which, if used by third parties, could infringe the rights of their owners.

Disclaimer of liability

We have checked that the contents of this document correspond to the hardware and software described. Nonetheless, differences might exist and therefore we cannot guarantee that they are completely identical. Nevertheless, the information contained in this document is reviewed regularly and any necessary changes will be included in subsequent editions.

Copyright © Siemens AG 1995 - 2024.
Order No. -----

Siemens AG 2024.
Subject to change without prior notice

Content

1 Introduction 1-13

1.1 System and software architecture 1-13

1.2 Security information 1-15

2 Getting started 2-17

2.1 New project 2-17
2.1.1 Creating a new project 2-17
2.1.2 Error evaluation 2-21

2.2 Integrating the application into Sinumerik Operate 2-22
2.2.1 Integrating a .NET application in Sinumerik Operate 2-22
2.2.2 Parameterizing the OEMFrame application (oemframe.ini) 2-28
2.2.3 OEM subdirectories and versioning 2-36

2.3 Debugging a project 2-39
2.3.1 Overview 2-39
2.3.2 Starting VisualStudio 2-39
2.3.3 Debugging 2-39

2.4 Installation on PCU50 2-40
2.4.1 Requirements 2-40
2.4.2 Installation 2-41
2.4.3 Certificates 2-42

2.5 FAQs 2-44
2.5.1 FAQs regarding the integration in Sinumerik Operate 2-44
2.5.2 FAQs on debugging 2-45

3 Communication with the NC/PLC 3-47

3.1 Introduction 3-48
3.1.1 Class model 3-48
3.1.2 Explanation of terms 3-48

3.2 Interactive test interface 3-52

3.3 Step-by-step examples 3-55
3.3.1 Preparation 3-55
3.3.2 Synchronous reading/writing of a variable 3-55
3.3.3 Asynchronous reading/writing of a variable 3-57
3.3.4 Hotlink to a variable 3-60
3.3.5 Reading/writing multiple variables – "array access" 3-61
3.3.6 Reading/writing multiple variables – "multi-access" 3-63

3.4 Reference DataSvc 3-65
3.4.1 Definitions 3-65

Unrestricted © Siemens AG 2024 All Rights Reserved

3.4.2 Creating a DataSvc object ... 3-65
3.4.3 Reading variables ... 3-66
3.4.4 Writing variables ... 3-67
3.4.5 Notification regarding a variable change (hotlink) ... 3-69
3.4.6 Further properties of the DataSvc class ... 3-71

3.5 Reference Item ... 3-75
3.5.1 Definitions ... 3-75
3.5.2 Variable paths ... 3-75
3.5.3 Creating an Item object ... 3-85
3.5.4 Properties ... 3-85

3.6 Reference DataSvcStatus ... 3-86
3.6.1 Definitions ... 3-86
3.6.2 Properties ... 3-86

3.7 FAQs ... 3-88
3.7.1 General FAQs ... 3-88
3.7.2 FAQs regarding hotlink ... 3-88
3.7.3 FAQs regarding machine data / GUDs ... 3-89

4 Program instance services (PI services) ... 4-91

4.1 Introduction ... 4-92
4.1.1 Class model ... 4-92

4.2 Step-by-step examples ... 4-93
4.2.1 Preparation ... 4-93
4.2.2 Sending a general PI service ... 4-93

4.3 Reference PiSvc ... 4-95
4.3.1 Definitions ... 4-95
4.3.2 Creating a PiSvc object ... 4-95
4.3.3 Issuing a PI service ... 4-96
4.3.4 Properties of the PiSvc class ... 4-97

5 Access to alarms and events ... 5-99

5.1 Overview ... 5-100
5.1.1 Introduction and explanation of terms ... 5-100
5.1.2 Class model ... 5-101

5.2 Step-by-step examples ... 5-103
5.2.1 Preparation ... 5-103
5.2.2 Accessing the alarm list ... 5-103
5.2.3 Access to the alarm events ... 5-105
5.2.4 Creating and deleting HMI alarms ... 5-107

5.3 Reference AlarmSvc ... 5-113
5.3.1 Definitions ... 5-113
5.3.2 Creating am AlarmSvc object ... 5-113
5.3.3 Notification about a change in the current alarm list ... 5-114
5.3.4 Notification about a change in the event list ... 5-115
5.3.5 Notification about a change in an individual event ... 5-115
5.3.6 Creating, acknowledging and canceling alarms ... 5-116

© Siemens AG 2024 All Rights Reserved

5.3.7 Further properties 5-119

5.4 Reference Alarm 5-120
5.4.1 Definitions 5-120
5.4.2 Properties 5-120

5.5 Reference AlarmSource 5-127
5.5.1 Definitions 5-127
5.5.2 Properties 5-127

5.6 FAQs 5-128
5.6.1 FAQs regarding alarm texts 5-128

6 Directory and file services 6-129

6.1 Introduction 6-130
6.1.1 Class model 6-130
6.1.2 Explanation of terms 6-131

6.2 Step-by-step examples 6-134
6.2.1 Preparation 6-134
6.2.2 Listing the contents of a directory (synchronous) 6-134
6.2.3 Listing the contents of a directory (asynchronous) 6-136
6.2.4 Generating, copying, and deleting a file (synchronous) 6-138
6.2.5 Generating, copying and deleting a file (asynchronous) 6-140
6.2.6 Determining file or directory attributes 6-143
6.2.7 Block-by-block reading and writing of data 6-144

6.3 Reference Node 6-147
6.3.1 Definitions 6-147
6.3.2 Creating a node object 6-148
6.3.3 Properties 6-148

6.4 Reference FileSvc 6-150
6.4.1 Definitions 6-150
6.4.2 Creating a FileSvc object 6-150
6.4.3 Management functions 6-150
6.4.4 Specifications for implementing delegates 6-160

6.5 Reference ControlAccessRights 6-163
6.5.1 Definitions 6-163
6.5.2 Properties 6-163

6.6 Reference NodeAttributes 6-165
6.6.1 Definitions 6-165
6.6.2 Properties 6-165

6.7 Reference device 6-166
6.7.1 Definitions 6-166
6.7.2 Creating a device object 6-166
6.7.3 Notification for drives 6-166
6.7.4 Properties 6-167

6.8 Reference FileSvcStatus 6-168
6.8.1 Definitions 6-168

Unrestricted © Siemens AG 2024 All Rights Reserved

6.8.2 Properties ... 6-168

6.9 FAQs ... 6-169
6.9.1 FAQs regarding "FileSvc" ... 6-169

7 Infrastructure services ... 7-171

7.1 Introduction ... 7-172
7.1.1 Class model ... 7-172
7.1.2 Explanation of terms ... 7-172

7.2 Step-by-step examples ... 7-173
7.2.1 Preparation ... 7-173
7.2.2 Current language - read, changeover, notify for change ... 7-173
7.2.3 Operating area selection, NCU and channel change ... 7-175

7.3 Reference infrastructure ... 7-178
7.3.1 Definitions ... 7-178
7.3.2 Notifications ... 7-178
7.3.3 Unsubscribing notifications ... 7-181
7.3.4 Properties of the class infrastructure ... 7-182
7.3.5 Setting the active NCU ... 7-183
7.3.6 Selecting the operating area ... 7-184
7.3.7 Setting the active channel ... 7-184
7.3.8 Shutting down the HMI ... 7-184
7.3.9 Is HMI Operate active? ... 7-184

8 Archive service ... 8-185

8.1 Introduction ... 8-186
8.1.1 Class model ... 8-186
8.1.2 Explanation of terms ... 8-186

8.2 Step-by-step examples ... 8-188
8.2.1 Preparation ... 8-188
8.2.2 Creating a series commissioning archive ... 8-188
8.2.3 Creating a user archive ... 8-192
8.2.4 Read-in an archive ... 8-195
8.2.5 Monitoring the functions of the archive service ... 8-198

8.3 Reference ArchiveSvc ... 8-200
8.3.1 Definitions ... 8-200
8.3.2 Creating an ArchiveSvc object ... 8-200
8.3.3 Creating an archive (SINUMERIK 840D sl) ... 8-200
8.3.4 Creating an archive (SINUMERIK ONE) ... 8-204
8.3.5 Reading-in an archive ... 8-205
8.3.6 Further functions ... 8-206
8.3.7 Specifications for implementing delegates ... 8-207

8.4 Reference ArchiveSvcStatus ... 8-209
8.4.1 Properties ... 8-209

8.5 Reference ArchiveSvcNotifier ... 8-211
8.5.1 Definitions ... 8-211
8.5.2 Creating an ArchiveSvcNotifier object ... 8-211

© Siemens AG 2024 All Rights Reserved

8.5.3 Activating the monitoring function ... 8-211
8.5.4 Deactivating the monitoring function ... 8-211
8.5.5 Specifications for implementing delegates ... 8-212

9 Drive machine data service ... 9-213

9.1 Introduction ... 9-214
9.1.1 Class model ... 9-214

9.2 Step-by-step examples ... 9-215
9.2.1 Preparation ... 9-215
9.2.2 Asynchronous listing of the drive objects of a control ... 9-215
9.2.3 Reading and writing of drive parameters ... 9-217
9.2.4 Asynchronous saving of drive objects ... 9-219

9.3 Reference Drive ... 9-222
9.3.1 Definitions ... 9-222
9.3.2 Creating a Drive object ... 9-222
9.3.3 Releasing assigned resources ... 9-222
9.3.4 Reading drive parameters ... 9-222
9.3.5 Writing drive parameters ... 9-224
9.3.6 Saving a drive object ... 9-224
9.3.7 Reset the drive object ... 9-225
9.3.8 Canceling an asynchronous function ... 9-225
9.3.9 Properties ... 9-225
9.3.10 Specifications for implementing delegates ... 9-227

9.4 Reference DrivesSvc ... 9-228
9.4.1 Definitions ... 9-228
9.4.2 Creating a DrivesSvc object ... 9-228
9.4.4 Creating a list of Drive objects ... 9-229
9.4.5 Accessing an individual drive object ... 9-229
9.4.6 Backing up drive objects ... 9-230
9.4.7 Triggering resets ... 9-230
9.4.8 Canceling asynchronous functions ... 9-231
9.4.9 Specifications for implementing delegates ... 9-231

9.5 Reference DriveSvcStatus ... 9-233
9.5.1 Definitions ... 9-233
9.5.2 Properties ... 9-233

10 TraceDataRecorder service ... 10-235

10.1 Introduction ... 10-236
10.1.1 Class model ... 10-236

10.2 Step-by-step examples ... 10-238
10.2.1 Preparation ... 10-238
10.2.2 Example 1 ... 10-240
10.2.3 Example 2 ... 10-245

10.3 Reference TraceDataRecorder ... 10-249
10.3.1 Creating an instance of TraceDataRecorder ... 10-249
10.3.2 Releasing assigned resources ... 10-249
10.3.3 Resetting to initial state ... 10-249

Unrestricted © Siemens AG 2024 All Rights Reserved

10.3.4 Transferring the parameterization 10-249
10.3.5 Starting the recording 10-250
10.3.6 Stopping the recording 10-250
10.3.7 Subscribing to events 10-250
10.3.8 Unsubscribing events 10-250
10.3.9 Specifications for implementing delegates 10-251
10.3.10 Properties 10-252

10.4 Reference TraceDataRecorderBinData 10-254
10.4.1 Creating a TraceDataRecorderBinData object 10-254
10.4.2 Array with recorded values 10-254
10.4.3 Array with errors that have occurred 10-254

10.5 Reference TraceDataRecorderDataset 10-255

10.6 Reference TraceDataRecorderErrorInterval 10-256

10.7 Reference TraceDataRecorderError 10-257

10.8 Reference TraceDataRecorderException 10-258

11 ToolManagement service 11-259

11.1 Introduction 11-260
11.1.1 Class model 11-260

11.2 Step-by-step examples 11-261
11.2.1 Preparation 11-261
11.2.2 Tracking changes 11-261

11.3 Reference ToolMngmntSvc 11-263
11.3.1 Creating a ToolMngmntSvc object 11-263
11.3.2 Releasing assigned resources 11-263
11.3.3 Functions 11-263
11.3.4 Specifications for implementing delegates 11-264

12 Action log service 12-265

12.1 Introduction 12-266
12.1.1 Class model 12-266
12.1.2 Explanation of terms 12-266

12.2 Step-by-step examples 12-268
12.2.1 Preparation 12-268
12.2.2 Synchronously reading out the action log 12-268
12.2.3 Asynchronously reading out the action log 12-272

12.3 Reference TripRecorder 12-275
12.3.1 Definitions 12-275
12.3.2 Creating a TripRecorder object 12-275
12.3.3 Releasing assigned resources 12-275
12.3.4 Starting and stopping the recording 12-276
12.3.5 Generation of log entries 12-276
12.3.6 Reading out the log 12-276

© Siemens AG 2024 All Rights Reserved

12.3.7 Status of the recording 12-278
12.3.8 Canceling a request that is being executed 12-278
12.3.9 Specifications for implementing delegates 12-278
12.3.10 Events 12-279

13 COM clients and .NetApi 13-281

13.1 Introduction 13-282
13.1.1 Overview 13-282
13.1.2 Links 13-282
13.1.3 Disclaimer of liability 13-282

13.2 Creating an interface DLL 13-283

13.3 Using the interface DLL by the COM client 13-285

14 Display Manager 14-287

14.1 Introduction 14-288
14.1.1 Class model 14-288

14.2 Reference DisplayManager 14-289
14.2.1 Definitions 14-289
14.2.2 Creating a SlDisplayManager object 14-289
14.2.3 API functions 14-289
14.2.4 Notifications 14-292

I Index I-293

I.1 Keyword index I-293

Unrestricted © Siemens AG 2024 All Rights Reserved

© Siemens AG 2024 All Rights Reserved

1 Introduction

1

1.1 System and software architecture

Operate .net has a component-oriented architecture. The scope of these components includes SINUMERIK-specific functions. Examples included are the service to read and write control variables (DataService), the AlarmService, which provides Operate with all currently available alarm and part program messages or the file service to simply handle files and directories in the NC, on the hard disk and on other drives, e.g. network drives or USB flash drives.

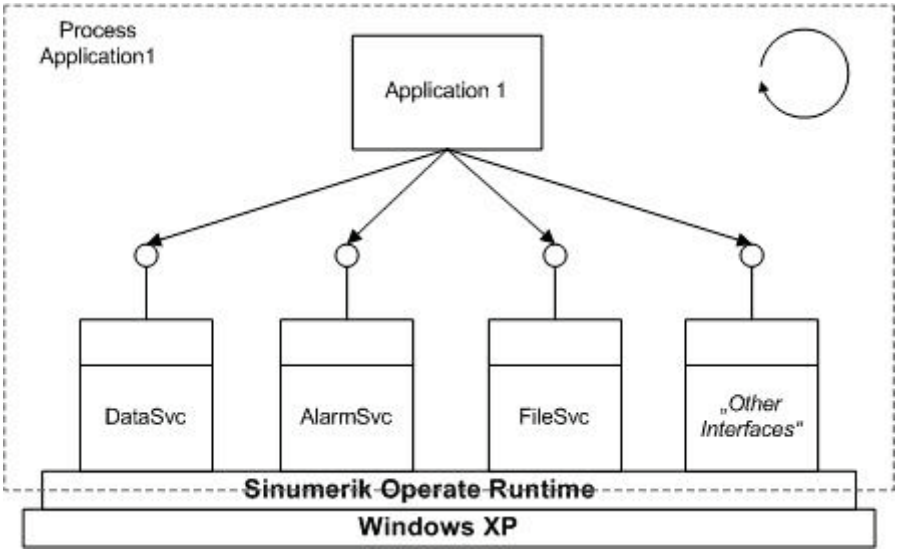


Fig. 1-1: Architecture

SINUMERIK Operate .net only includes interfaces to access the control. There are no components to lay out the interface. This means that the interface is generated as a standard Windows application with C# or VB.NET. Examples are supplied for both programming languages. However, the step-by-step examples in the documentation are restricted to the C# programming language.
The applications generated can be integrated in SINUMERIK Operate on a PCU50/IPC.

Unrestricted © Siemens AG 2024 All Rights Reserved

Important

Machine-critical OA functions must be secured using suitable real-time applications. Under no circumstances may the implementation of such machine-critical functions just be based on Operate functions. It is especially important to note that the "Hotlink mechanism" (notification when values change) must not be used as the sole basis for implementing machine-critical functions.

© Siemens AG 2024 All Rights Reserved

1.2 Security information

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

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

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

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

Siemens products and solutions undergo continuous development to make them more secure. Siemens strongly recommends that product updates are applied as soon as they are available and that the latest product versions are used. The use of outdated or no longer supported versions can increase the risk of cyber threats.

To stay informed about product updates, subscribe to the Siemens Industrial Security RSS Feed at:
https://www.siemens.com/industrialsecurity
(https://new.siemens.com/global/en/products/services/cert.html#Subscriptions)

You can find more information on the internet:
Configuration Manual Industrial Security
(https://support.industry.siemens.com/cs/ww/de/view/108862708)

Unrestricted © Siemens AG 2024 All Rights Reserved

⚠WARNING

Unsafe operating states resulting from software manipulation

Software manipulation, e.g. viruses, Trojans or worms, can cause unsafe operating states in your system that may lead to death, serious injury, and property damage.

- Keep the software up to date.
- Integrate the automation and drive components into a holistic, state-of-the-art industrial security concept for the plant or machine.
- For your holistic industrial security concept, take into account all products used.
- Protect files stored on interchangeable storage media from malicious software by taking suitable security measures, e.g. virus scanners.
- Carefully check all security-related settings once commissioning has been completed.

© Siemens AG 2024 All Rights Reserved

2

2 Getting started

2.1 New project

2.1.1 Creating a new project

Overview

In order that the interfaces of SINUMERIK Operate .NET can be used, the following steps must be performed.

Supported programming languages

SINUMERIK Operate .NET supports the following programming languages:

- Microsoft C#
- Microsoft VB.NET

Note

The documentation is restricted to the programming language C#. However, examples in VB.NET are supplied for every interface.

Opening Visual Studio

Visual Studio must be started via a start menu entry that was also installed by Siemens. Only then are settings made to the environment, which are required to access the interfaces:

Start → Siemens Automation → Visual Studio Professional 2022

Create a project

Operate must have a window to be able to link the application later. This is the reason why a "Windows Forms Application" should be generated. If the application requires multiple windows, it should be designed as an MDI application (Multiple Document Interface application). You will find more detailed information on MDI in the Microsoft documentation.

Unrestricted © Siemens AG 2024 All Rights Reserved

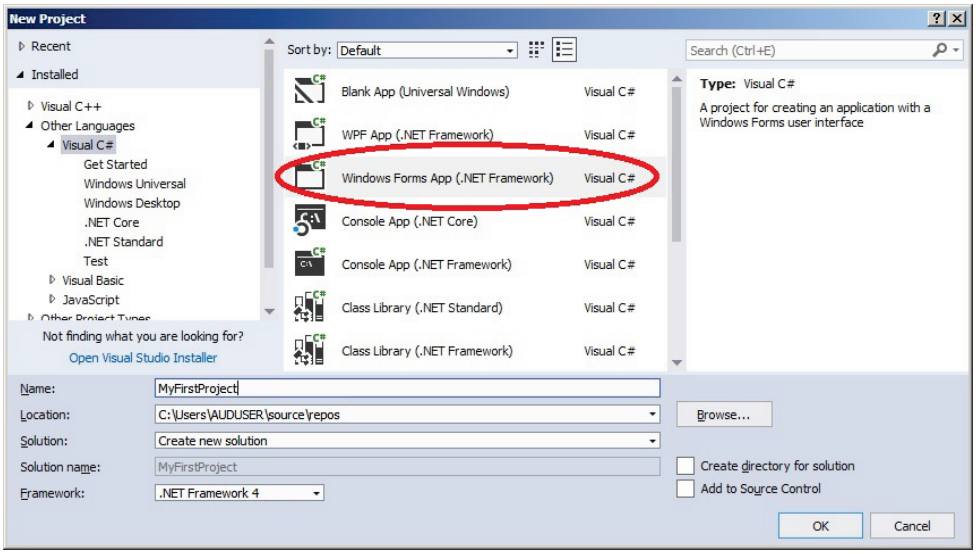


Fig. 2-1: Creating a new project (Windows Forms Application)

.NET-Framework

When creating a new project, it should be ensured that the version of .NET-Framework, on which the project is based, is not higher than the version of the .NET-Framework on the target system. The currently installed version first can be viewed on the PCU in the Service mode under "Control Panel" ➔ "Add or Remove Programs".

When generating the project, the version of the .NET-Framework used can be set:

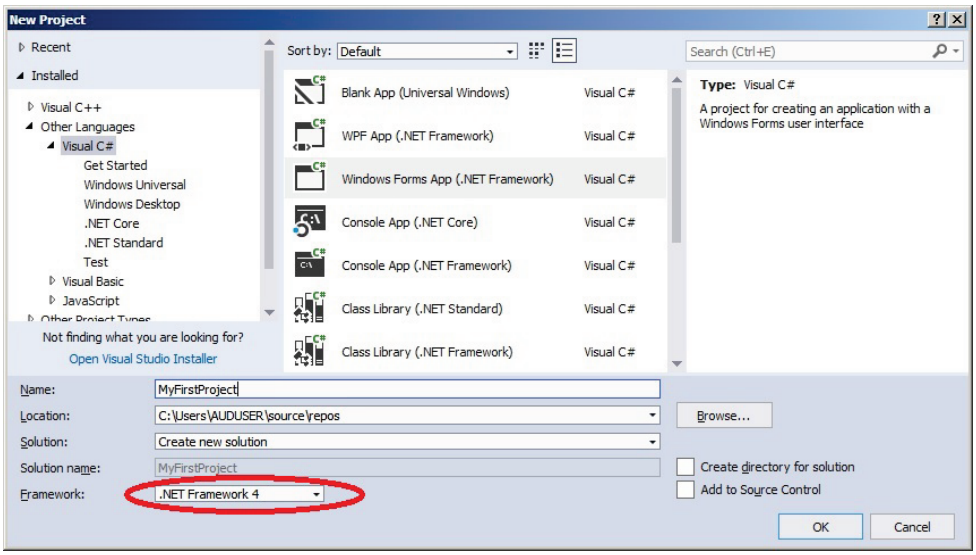


Fig. 2-2: : Creating a new project – setting the .NET-Framework used

© Siemens AG 2024 All Rights Reserved

The version used can also be changed at a later time:

1. "Project" menu
2. "[Project name] properties..." menu command
3. "Application" tab
4. The version of .NET-Framework used can be selected in the selection box "Target Framework".

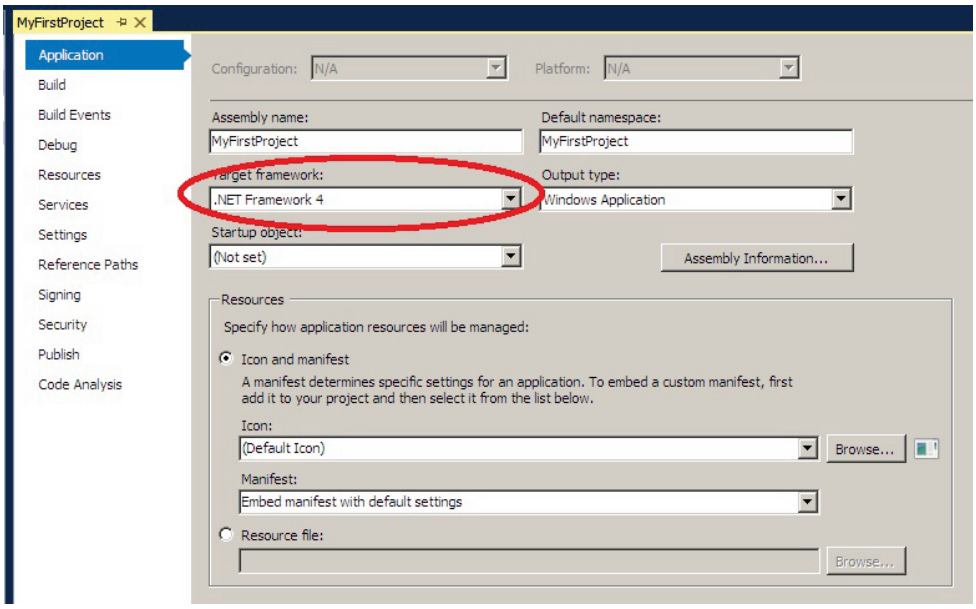


Fig. 2-3: Changing the .NET-Framework used

Note

If the version of the .NET-Framework is reduced at a later date, then this can result in a compatibility problem, if functions of a higher version were used.

Configuring the platform target

Configuration: Set "All Configurations" and platform target to "x86".

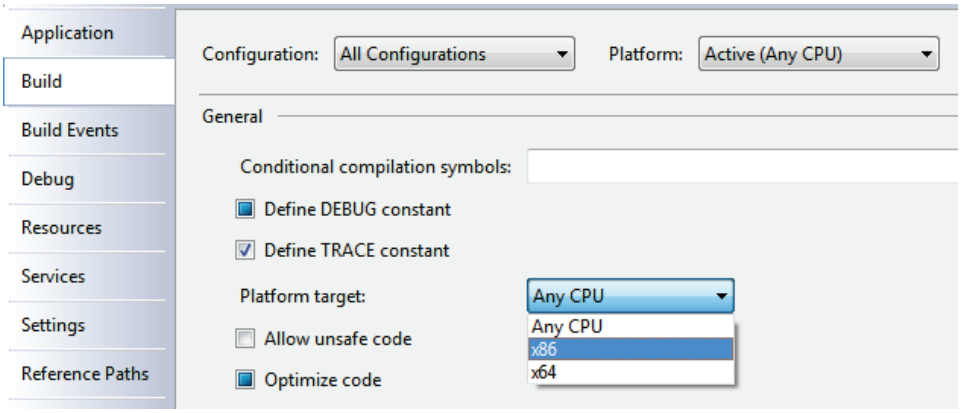


Fig. 2-4: Platform target

Unrestricted © Siemens AG 2024 All Rights Reserved

Integrating the reference

In order that the interfaces can be used, the next step is to integrate the reference of the interface into the project:

1. "Project" menu
2. "Add Reference" menu command
3. Select "Siemens.Sinumerik.Operate.Services4"
4. Press "OK"

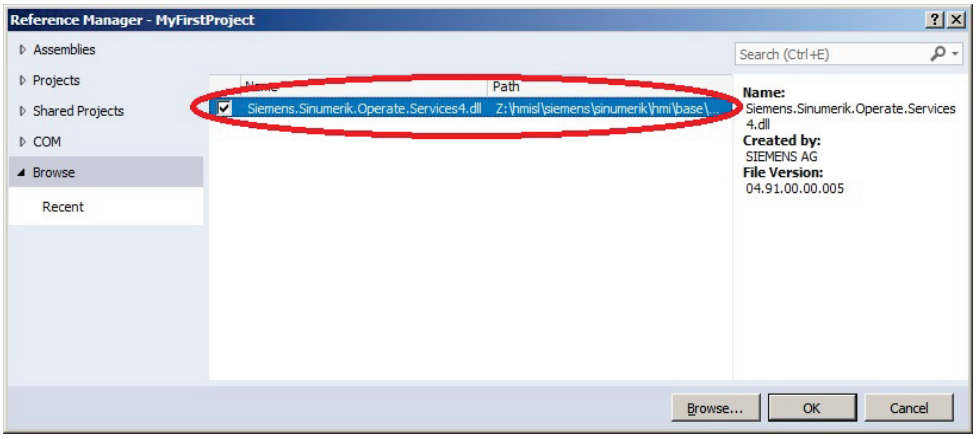


Fig. 2-5: Inserting the reference

Namespace

All classes of the SINUMERIK Operate .NET are defined in the following namespace:

Siemens.Sinumerik.Operate.Services

It makes sense to make this namespace available in the C# project.
For this purpose, at the start of each source code file, the using-directive can be used in the C# project:

```
using Siemens.Sinumerik.Operate.Services;
```


© Siemens AG 2024 All Rights Reserved

2.1.2 Error evaluation

There is no error evaluation in the individual step-by-step examples. However, this must remain the exception. As all functions of the Operate .net issue exceptions in the case of an error, all function calls must be detected in try-catch blocks. Which exceptions the individual functions issue are located in the reference of the particular class.

```
try
{
    // Function call(s)
}
catch (DataSvcException ex)
{
    // Error evaluation, for DataSvcExceptions
}
catch (Exception ex)
{
    // Error evaluation, for all other exceptions
}
```

The error evaluation is shown precisely in the source code of the individual examples.

Information about error numbers

More detailed information about error numbers, which are supplied in exceptions, can be determined using the application "errorLookup".

Note

In this regard, we recommend that you include the following EventHandler:

WinForms:
Application.ThreadException
See:
http://msdn.microsoft.com/de-de/library/system.windows.forms.application.threadexception(v=vs.110).aspx

WPF:
Application.Current.DispatcherUnhandledException
See:
http://msdn.microsoft.com/de-de/library/system.windows.application.dispatcherunhandledexception(v=vs.110).aspx

---

Unrestricted © Siemens AG 2024 All Rights Reserved

2.2 Integrating the application into Sinumerik Operate

2.2.1 Integrating a .NET application in Sinumerik Operate

.NET applications must be integrated into SINUMERIK Operate using the "OEMFrame". Generally, these applications are selected using a softkey on the area switchover bar. The required settings are described in this chapter.

SINUMERIK Operate is started and controlled from an application called System Manager. This System Manager also handles the control of the OEMFrame applications.

The System Manager is configured using the systemconfiguration.ini file.

System configuration

In the systemconfiguration.ini configuration file, is used to define from which operating areas the HMI is configured.

To integrate an OEMFrame application into the system, create the systemconfiguration.ini file in one of the following two directories:

- **<Installation path>/user/sinumerik/hmi/cfg**
- **<Installation path>/oem/sinumerik/hmi/cfg**

The sections of the systemconfiguration.ini file that are relevant for linking OEMFrame applications are described below.

Section [processes]
This section specifies the processes to be managed by the System Manager. The applications to be linked as OEMFrame applications must be specified here. The following entries are relevant for OEMFrame applications:

Table 4-1: Entries in the [processes] section that are relevant for OEMFrame applications

Value	Meaning
process	Symbolic name of the OEMFrame application. This is required for the operating area configuration.
cmdline	Command line, passed to the oemframe.exe process on starting. "\\" must be used as path separator. (the path does not have to be specified if the OEMFrame application is in the directory "\oem\sinumerik\hmi\appl")
oemframe	Should always be set to "true" for OEMFrame applications.
windowname	Window name of the OEMFrame application - determined using the supplied Findwindow tool (Start → Siemens Automation → Find Window) Maximum 255 characters
classname	Class name of the OEMFrame application - determined using the supplied Findwindow tool (start menu entry, see "windowname"). Maximum 127 characters
deferred	true: OEMFrame application is not started when the HMI starts up, but the first time it is selected.

© Siemens AG 2024 All Rights Reserved

Value	Meaning
startupTime	The linked process starts as follows: immediately ➔ (default) afterServices ➔ After all services have started. afterGuis ➔ After all GUI components have started. When shutting down SINUMERIK Operate, the order is inverted: immediately ➔ (default) afterServices ➔ After all services have shut down. afterGuis ➔ After all GUI components have shut down.
gimmekeys	Release mask for keys of the system configuration that are to be handled by the OEMFrame application. The parameterization is performed in the form of a bit mask. (See example 2)
disablekeys	Parameterization for the behavior of the keyboard filter. The parameterization is performed in the form of a bit mask. (See example 2)
menuselectkey	Changes the key that displays the operating area menu (default: F10). The value is a logically ORed combination of the modifiers Key_Shift, Key_Alt, Key_Ctrl, and the virtual key code, as defined by Microsoft. (See example 3)
timeout	Maximum duration for the search for the OEMFrame application in milliseconds. If the OEMFrame application was not found within this time, it can be managed by the System Manager. (Default: systemconfiguration.ini / [miscellaneous] / startTimeoutDefault)
shutdowntime	Maximum duration for downloading the OEMFrame application in milliseconds. If the OEMFrame application was not found within this time, the process is canceled. (Default: systemconfiguration.ini / [miscellaneous] / shutdownTimeoutDefault)
processaffinitymask	Bit mask which can be used to specify on how many processor cores the OEMFrame application runs. (Default: systemconfiguration.ini / [miscellaneous] / ProcessAffinityMask)

The Gimmekeys bit mask can be parameterized as follows:

Table 4-2: Gimmekey bit mask

Bit	Keys	Meaning
0	F1 – F8	Horizontal softkeys (upper bar, HU)
1	Shift+F1 – Shift+F8	Vertical softkeys (right bar, VR)
2	Ctrl+F1 – Ctrl+F8	Horizontal softkeys (lower bar, HL)
3	Shift+Ctrl+F1 – Shift+Ctrl+F8	Vertical softkeys (left bar, VL)
4	F9	Recall
5	Shift+F9	ETC switchover
6	F10	Operating area menu
7	Shift+F10	M key
8	F11	Channel switchover key
9	Shift+F11	M key (hard key)
10	F12	Info/help
11	Shift+F12	Custom key (hard key)
12	ESC	Alarm cancel
13	HOME	Window switchover key
14	END	PROGRAM (hard key)
15	PAGE UP	ALARM (hard key)

Unrestricted © Siemens AG 2024 All Rights Reserved

Bit	Keys	Meaning
16	PAGE DOWN	TOOL OFFSET (hard key)
17	HOME (NUMPAD)	PROGRAM MANAGER (hard key)
18	F13 – F20	Extended horizontal softkeys (upper bar, HU)
19	Shift+F13 – Shift+F20	Extended, vertical softkeys (right bar, VR) and right direct keys HT8
20	Ctrl+F13 – Ctrl+F20	Extended horizontal softkeys (lower bar, HL)
21	Shift+Ctrl+F13 – Shift+Ctrl+F20	Extended, vertical softkeys (left bar, VL) and left direct keys HT8

The Gimmekey bit mask for an OEMFrame application is set to 0xF by default. This means that all key combinations of F1 through F8 are available to the OEMFrame application. The OEMFrame application itself can handle a specific key (or key combination) by setting additional bits. Otherwise, the system configuration will evaluate the key (or key combination) and it will not even reach the OEMFrame application.

The Disablekeys bit mask can be parameterized as follows:

Table 4-3: Disablekeys bit mask

Bit	Keys	Meaning
0-7	Reserved	
8	(Shift)+Ctrl+F1	Lower and left-hand softkey bar (HL, VL)
9	(Shift)+Ctrl+F2	Lower and left-hand softkey bar (HL, VL)
10	(Shift)+Ctrl+F3	Lower and left-hand softkey bar (HL, VL)
11	(Shift)+Ctrl+F4	Lower and left-hand softkey bar (HL, VL)
12	(Shift)+Ctrl+F5	Lower and left-hand softkey bar (HL, VL)
13	(Shift)+Ctrl+F6	Lower and left-hand softkey bar (HL, VL)
14	(Shift)+Ctrl+F7	Lower and left-hand softkey bar (HL, VL)
15	(Shift)+Ctrl+F8	Lower and left-hand softkey bar (HL, VL)
16	Reserved	
17	Reserved	

The Disablekeys bit mask for an OEMFrame application is set to 0x3FFFF by default. This means that all keyboard sequences are filtered out and will not reach the OEMFrame application. If a bit is set to 0, the keyboard filter for the corresponding keyboard sequence will be deactivated and the OEMFrame application will be unable to receive it. If you want an OEMFrame application to receive all softkeys of the lower and left-hand softkey bar, for example, set the Disablekeys bit mask to 0x300FF.

The ProcessAffinityMask bit mask can be parameterized as follows:

Table 4-4: ProcessAffinityMask bit mask

Bit	Core	Meaning
0	Core0	OEMFrame application is running in the 1st core
1	Core1	OEMFrame application is running in the 2nd core
2	Core2	OEMFrame application is running in the 3rd core
X	CoreX	etc.

© Siemens AG 2024 All Rights Reserved

Per default, the setting from the systemconfiguration.ini at [miscellaneous] / ProcessAffinityMask applies. If the OEMFrame application is to run in several processor cores, the appropriate bits are set. The entry 0xFFFFFFFF means that the application is running simultaneously on all processor cores.

The Gimmekeys bit mask, the Disablekeys bit mask and the ProcessAffinityMask bit mask can be specified with a decimal code (e.g. 31) or a hexadecimal code (e.g. 0x1F).

Example 1:

```
[processes]
PROC500=process:=notepadOEM, cmdline:="C:\\WINDOWS\\system32\\notepad.exe",
oemframe:=true, deferred:=true, windowname:="Untitled - Notepad",
classname:="Notepad"

PROC501=process:=calcOEM, cmdline:="C:\\WINDOWS\\system32\\calc.exe",
oemframe:=true, deferred:=true, windowname:="Calculator",
classname:="SciCalc"

[areas]
AREA500=name:=AreaNote, process:=notepadOEM
AREA501=name:=AreaCalc, process:=calcOEM
```

In this example, the two Windows applications "notepad.exe" and "calc.exe" are configured as OEMFrame applications.

Example 2:

```
[processes]
PROC500= process:=keycatcherOEM, cmdline:="keycatcher.exe", oemframe:=true,
deferred:=true, windowname:="keycatcher", classname:="QWidget",
gimmekeys:=0x1F, disablekeys:=0x300FF

[areas]
AREA500=name:=AreaKeyCatcher, process:= keycatcherOEM
```

The Windows "keycatcher.exe" application is linked in this example. This is located in directory "\oem\sinumerik\hmi\appl"; this is the reason that the path does not have to be specified. In this case, all 4 softkey bars and the Recall key are made available to the Windows application. The keyboard filter for the lower and the left-hand softkey bar is deactivated.

Example 3:

```
[processes]
PROC500= process:=keycatcherOEM, cmdline:="keycatcher.exe", oemframe:=true,
deferred:=true, windowname:="keycatcher", classname:="QWidget",
gimmekeys:=0x4F, disablekeys:=0x300FF, menuselectkey:=Key_Control|0x7B

[areas]
AREA500=name:=AreaKeyCatcher, process:= keycatcherOEM
```

Unrestricted © Siemens AG 2024 All Rights Reserved

The Windows "keycatcher.exe" application is linked in this example. In this case, all four softkey bars and the F10 key will reach the Windows application. Press Ctrl+F12 to show the operating area menu when a Windows application is displaying (the system configuration no longer evaluates F10).

Example 4:

```
[processes]
PROC500= process:=keycatcherOEM, cmdline:="keycatcher.exe", oemframe:=true,
deferred:=true, windowname:="keycatcher", classname:="QWidget",
gimmekeys:=0xF, disablekeys:=0x300FF

[areas]
AREA500=name:=AreaKeyCatcher, process:= keycatcherOEM
```

The Windows "keycatcher.exe" application is linked in this example. In this case, all four softkey bars are fed to the Windows application. Key sequences Ctrl-F1 to Ctrl-F8 can be mapped onto key sequences Ctrl-F13 to Ctrl-F20.

Section [areas]
The HMI operating areas are configured in this section. The following entries are relevant for OEMFrame applications:

Table 4-5: Entries of the [areas] section that are relevant for OEMFrame applications

Value	Meaning
name	Symbolic name for the operating area
process	Name of the OEMFrame application as set in the [processes] section.
panel	Name of the panel (header) to be used. Only "SlHdStdHeaderPanel" is currently available for OEM applications.
plcid	ID to identify the operating area using the Operate HMI monitor (display MD9032, only values in the range of 150-199 are permissible)

Example:

```
[areas]
AREA600= name:=AreaOEM, process:=notepadOEM
AREA601= name:=AreaCalc, process:=calcOEM, panel:=SlHdStdHeaderPanel
```

Important

In the "processes" and "areas" sections, the number range 500-999 is reserved for OEM customers. If numbers less than 500 are used, basic Siemens components may be overwritten.

Important

OEMFrame applications that use the programming interfaces of HMI-Advanced are not supported.

Important

To avoid write errors, the entries under "Section [processes]" and "Section [areas]" should only be determined using the "Find Window" tool provided (start → Siemens Automation → Find Window).

Additional information is provided in document "FindWindow.pdf" in the SINUMERIK Integrate Create MyHMI /3GL documentation.

© Siemens AG 2024 All Rights Reserved

Section [miscellaneous]
Various settings can be made in this section. However, only the Start operating area is usually changed here.

Table 4-6: Entries for the [miscellaneous] section

Key	Value
startuparea	Name of the start operating area.

Example:

```
[miscellaneous]
startuparea = AreaOEM
```

Operating area menu configuration

The operating area menu is used to switch between the operating areas configured in "systemconfiguration.ini". A softkey for selecting the appropriate operating area is provided on the horizontal softkey bar for each operating area that is configured.

The operating area displays the configured names of the operating areas from "systemconfiguration.ini" as the text on the operating area softkeys. The menu automatically searches for a free softkey on the horizontal softkey bar for each operating area. If you want to assign the operating area to a specific softkey in the operating area menu or if you want to display a foreign-language text or an icon for the operating area on the softkey, further configuration is possible and is required to achieve this.

This configuration is performed in the "slamconfig.ini" configuration file. This file is stored in the same directory as the "systemconfiguration.ini" file.

A section can be created for each operating area that has been configured in systemconfiguration.ini. The section must have the name configured for the operating area in question, e.g. AreaOEM.

The following values can be set for an operating area:

Table 2-7: Values for configuring the operating area in slamconfig.ini

Value	Meaning
TextId	Text-ID for a foreign-language text that is to be displayed as the softkey label.
TextContext	Context of the foreign-language text.
TextFile	Name of the text file which includes the context and the foreign-language text.
Picture	Name of an image file that is to be used as an icon for the softkey.
SoftkeyPosition	Fixed softkey position of the area softkey. Positions 1 through 8 are the first ETC level; 9 through 16, the second ETC level, and so on.
AccessLevel	Access level as from which the softkey will appear. If this value is not specified, the access level will be set to 7 = keyswitch position 0.

Unrestricted © Siemens AG 2024 All Rights Reserved

Example:

```
[AreaOEM]
; Text-ID of a language dependent text
TextId = MY AREA
; File name of the text file which contains the Text-ID
TextFile = mytextfile
; Context in the text file to which the Text-ID is assigned to
TextContext = mycontext
; File name of an icon shown on the area softkey
Picture = mypicture.png
; Position of the area softkey on area menu,
; If no position is specified, an empty position is searched
SoftkeyPosition = 7
; Access level of the area softkey
AccessLevel = 5
```

The area softkey for the "AreaOEM" operating area is configured in this example. The softkey is at position 7 in the operating area menu and is displayed with access level "Keyswitch position 2" and higher. This softkey displays the text stored in the mytextfile_xxx.ts text file in the context "mycontext" with the Text-ID "MY_AREA". The mypicture.png icon is also displayed.

Example of a text file (mytextfile_eng.ts)
stored under "\oem\sinumerik\hmi\lng":

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE TS>
<TS>
    <context>
        <name>mycontext</name>
        <message>
            <source>MY_AREA</source>
            <translation>AppName</translation>
        </message>
    </context>
</TS>
```

! Important

Operating area position 7 is reserved for OEM customers.

Note

If an OEMFrame application comprises several forms, then these should be combined under one MDI frame. Only then can it be guaranteed that the System Manager correctly executes the area switchover.

2.2.2 Parameterizing the OEMFrame application (oemframe.ini)

Additional parameters can be assigned for the OEMFrame applications using file "oemframe.ini".

A dedicated section is created for each OEMFrame application for which parameters are to be stored. This section has the same name as the corresponding program file but without the file name extension.

The "oemframe.ini" is located in directory "..\compat\oem". If "oemframe.ini" does not yet exist, it should be created at that location.

© Siemens AG 2024 All Rights Reserved

Parameter overview

Table 2-8: "oemframe.ini" parameters

Entry	Meaning	Default
WindowStyle_On	Defines properties to be assigned to the window.	0
WindowStyte_Off	Defines properties that the window must not have.	0
x	Horizontal starting coordinate of the OEMFrame application (Unit: pixels)	0
y	Vertical starting coordinate of the OEMFrame application (Unit: pixels)	0
Width	Width of the OEMFrame application (Unit: pixels)	Width of the desktop
Height	Height of the OEMFrame application (Unit: pixels)	Height of the desktop
nDelayInitComplete	Delays feedback to the System Manager.	0
fSearchOnlyForTask-Window	States whether the window specified in the "systemconfiguration.ini" belongs to the task also specified in that file.	1
fRestoreTaskWindow	Defines the behavior when exiting an application that was started from the OEMFrame application.	0
fKeepPlacement	Deactivates the size adaptation.	0
fForceTaskFocus fSearchForPopUps	This parameter defines which window of the OEMFrame application will be displayed when starting.	0 1
nInitShowMode	State in which the window of the OEMFrame application is displayed when the application is started.	SW_SHOWMIN-NOACTIVE
nShowMode	State in which the window of the OEMFrame application is displayed when it is shown.	SW_SHOW-NORMAL
nUnShowMode	State into which the window of the OEMFrame application is put when it is hidden.	SW_SHOWMIN-NOACTIVE
fWinForms	Must be set if the application is a "Windows Forms Application".	0
nSwitchToTaskAfter-Termination	Controls the behavior when the OEMFrame application is exited.	-1
fFindWindowWithWildcards	Should be set when using "wildcards" (?,*) for the window name.	0
nPlacementMode	Behavior when using with SideScreen or DisplayManager	0

Unrestricted © Siemens AG 2024 All Rights Reserved

WindowStyle_On / WindowStyle_Off

The appearance of a Windows application is defined in part using the SetWindowLong Windows API functions. When the SetWindowLong function is called, the appearance of the application is controlled using an 8-byte word. Two bytes of this word can be changed using the WindowStyle_On and WindowStyle_Off attributes. These are explained in the following table:

Table 2-9: Control options using the WindowStyle attributes

0000	0000	xxxx	xxxx	0000	0000	0000	0000
		1010		Header			
		1000		Border			
		0100		Type of window frame on a dialog box			
		0010		Vertical scroll bar			
		0001		Horizontal scroll bar			
			1000	System menu			
			0100	Frame (Thickframe)			
			0010	Minimize box			
			0001	Maximize box			

The values shown in binary notation here are assigned to the WindowStyle attributes as a decimal number. You can use the pocket calculator of Windows to convert from binary to decimal, and vice versa. The calculator is usually found in the Accessories program group.

Example:
The properties of the system menu as well as the horizontal and vertical scroll bar should be defined. According to the table, these are:

```
0000 0000 0011 1000 0000 0000 0000 0000 binary or 0038 0000 Hex.
Now, call the pocket calculator and
- Click the Hex button
- Enter the string of digits 380000 (leading zeros can be omitted)
- Click the Dec selection button
- Copy result 3670016, using the Edit menu
- Insert the result at Attribute
```

The "WindowStyle_On" parameter defines the properties that will be assigned to the window, and the "WindowStyle_Off" parameter defines the properties that the window will not have.

Example 1:
For the Notepad editor, the system menu as well as the horizontal and vertical scroll bar should be activated:

```
[notepad]
WindowStyle_On = 3670016
```

Example 2:
No minimize box and no maximize box should be displayed for the Notepad editor:

```
[notepad]
WindowStyle_Off = 196608
```


© Siemens AG 2024 All Rights Reserved

X/Y

The X and Y attributes define the starting coordinates of the window of the Windows application to be linked, measured from an origin in the upper left corner of the screen.

X is the horizontal coordinate; Y is the vertical coordinate pointing downward. The unit of measurement is pixels.

Width

This attributes defines the width of the window for the Windows application, measured in pixels from the window origin according to attribute X.

Height

This attribute defines the height of the window for the Windows application, measured in pixels from the window origin according to attribute Y.

nDelayInitComplete

As soon as the window of the Windows application has been found, a message indicating this is sent to the System Manager. The Windows application can then be selected using the System Manager. This message can be delayed using the "nDelayInitComplete" parameter. This is necessary if the Windows application still has to execute actions that will take a long time after its window has been created, which could result in the window being incorrectly displayed if it were activated too early by the System Manager.
.
(Unit: milliseconds, default value: 0)

Example:
After creating its window, a Windows application "app.exe" reads additional status data from a database that is absolutely necessary for the Windows application to run correctly. The window of the Windows application may only be displayed after all status data has been read. On the average, this read operation should take approx. 1 sec. The following parameter assignment is required:

```
[app]
;worst case
nDelayInitComplete = 2000
```

fSearchOnlyForTaskWindow

This parameter specifies whether or not the window specified in the file "systemconfiguration.ini" using ClassName/WindowName belongs to the task that is also specified there. If the window belongs to the specified task, the parameter "fSearchOnlyForTaskWindow" is assigned the value 1. If the window does not belong to the specified task, "fSearchOnlyForTaskWindow" should be set to 0.

In this case, not just the windows of the task configured in "systemconfiguration.ini" are considered in the search for the specified window, but all of the windows that exist in the system at the time of the search.
(Flag, default value: 1)

Unrestricted © Siemens AG 2024 All Rights Reserved

Example:
The Windows application comprises more than one process, e.g. a "startup.exe" and a "user.exe". In the "systemconfiguration.ini" file (only) "startup.exe" is entered, from which "user.exe" is then started. The application window belongs to "user.exe" and is therefore not found when searching is restricted to the windows of "startup.exe". The following parameter assignment is required:

```
[startup]
fSearchOnlyForTaskWindow = 0
```

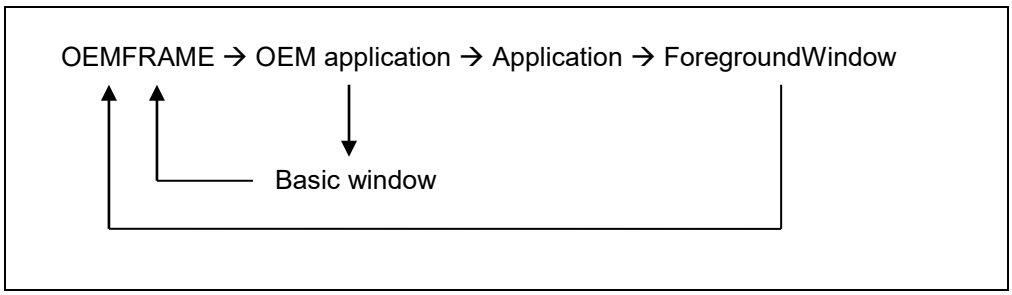
fRestoreTaskWindow

This parameter defines the behavior when exiting a Windows application that was started from the OEMFrame application (second task level).

Note

This situation cannot be completely controlled and should therefore be avoided if at all possible!

When the OEMFrame application is selected, the last window that was active (ForegroundWindow) is saved by default. This window is reactivated when the OEMFrame application is reselected. If another application has been started from the OEMFrame application, the active window will usually belong to this Windows application.



The proxy application (OEMFRAME.EXE) cannot detect when the Windows application ends. It is therefore unable to put the window of the OEMFrame application into the foreground in this situation, which may result in incorrect scenarios being displayed when the second task level is ended.

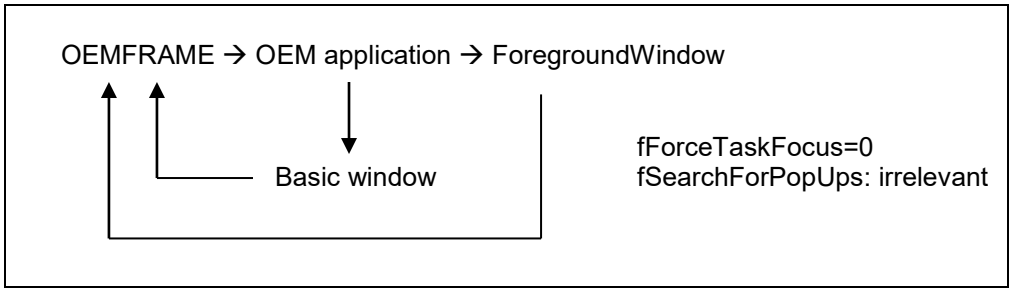
The "fRestoreTaskWindow" parameter provides a partial remedy. If the value 1 is assigned to this parameter, two windows are always activated when the OEMFrame application is selected or an application is started from the OEMFrame application. The basic window of the OEMFrame application is placed in the foreground first and then the "ForegroundWindow" is placed over this window. This ensures that the basic window of the OEMFrame application will appear when the second task level is ended.

Restrictions:
If a modal window (dialog box) is active in the OEMFrame application when the second application is started, this modal window will lose the input focus when the second application is ended. In this case, the operator must explicitly set the focus in the dialog box (e.g. using the mouse).

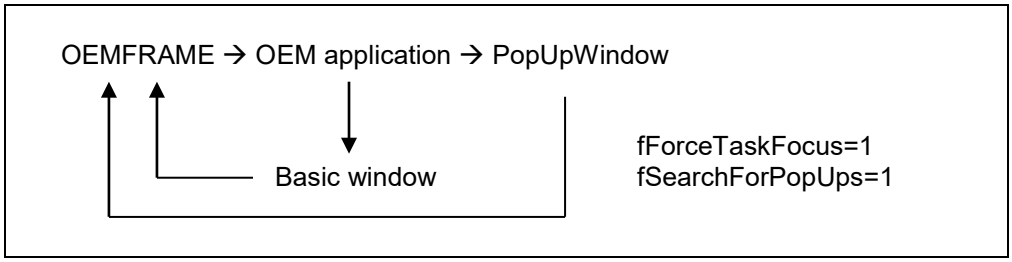
© Siemens AG 2024 All Rights Reserved

fForceTaskFocus / fSearchForPopUps

These two parameters define which window the OEMFrame application will activate after it has been deselected and reselected again. On switching from one operating area to another, the default setting is that the last active window of the OEMFrame application is saved. When the application is selected again, this window is activated again. In the Windows API, this window is called the ForegroundWindow.



This (default) behavior makes sense for the majority of applications. However, there are exceptions for which this behavior can be changed. If the two parameters "fForceTaskFocus" and "fSearchForPopUps" are set to the value 1, when the OEMFrame application is deselected, a search is not made for the ForegroundWindow but rather for an active popup window that belongs to the basic window of the application. If a popup window is found, it will be displayed when the OEMFrame application is selected again. If there is no popup window, the basic window of the application will be displayed when the application is reselected.



A search for an active popup window is not made if parameter "fSearchForPopUps" is set to the value 0. When an application is deselected and then selected again, only the basic window of the OEMFrame application is considered. The basic window is specified in ClassName/WindowName in the "systemconfiguration.ini" file.

fKeepPlacement

This function deactivates the resizing (zooming) for the basic window of the OEM application. The application is normally zoomed to the screen size before it is displayed. In the case of applications that do not allow their windows to be zoomed, the resizing can result in display problems. In such cases, the zoom function must be deactivated.

Example:
An application "fixres.exe" should be displayed in its programmed window size. size. The following parameter assignment is required:

```
[fixres]
fKeepPlacement = 1
```

Unrestricted © Siemens AG 2024 All Rights Reserved

nInitShowMode / nShowMode / nUnShowMode

The three parameters define how the application window will be displayed when the application (nInitShowMode) is started, hidden, and shown. The "nShowMode" parameter refers to showing (area is activated); the "nUnShowMode" parameter, to hiding. The following value range applies to both parameters:

0: The application window is hidden (SW_HIDE).

1: The application window is displayed in its original form (position, size) and has the input focus (SW_SHOWNORMAL, SW_NORMAL).

2: The application is minimized and has the input focus (SW_SHOWMINIMIZED).

3: The application window is maximized (SW_SHOWMAXIMIZED).

4: The application window is displayed without having the input focus (SW_SHOWNOACTIVATE).

5: The application is displayed and has the input focus (SW_SHOW).

6: The application window is minimized and loses the input focus (SW_MINIMIZE).

7: The application window is minimized without having the input focus (SW_SHOWMINNOACTIVE).

8: The application window is displayed without having the input focus (SW_SHOWNA).

9: The application window is displayed in its original form (position, size) (SW_RESTORE).

10: The application window is displayed in the same way as when the application was started (SW_SHOWDEFAULT).

Note

The default settings are suitable for most applications. However, in some cases, display problems can occur (window not maximized, etc.). This can be remedied by setting parameter nShowMode=3.

fWinForms

If the OEMFRAME application is a "Windows Forms Application" with the name "myApp.exe", then the following parameterization is required:

```
[myApp]
fWinForms = 1
```

Note

If the parameter is not set, the OEMFrame application will not be started as a maximized window and any sizing (X, Y, Width, Height) will have no effect.

© Siemens AG 2024 All Rights Reserved

nSwitchToTaskAfterTermination

This parameter controls the behavior when the OEMFrame application is exited. The System Manager normally displays the area switch menu in this situation and the user must explicitly switch to another area. This parameter can be used to initiate an automatic switch to the previously active area.

Value range of the parameter:
–1 Area switch menu display
–2 Switch to the previously active area
(Default value: –1)

Example:
When the "closeapp.exe" Windows application is exited, a switch should be made to the previous area. The following parameter assignment is required:

```
[closeapp]
nSwitchToTaskAfterTermination = -2
```

fFindWindowWithWildcards

This parameter activates the wildcard search for the window name. ? (an arbitrary character) and * (arbitrary number of characters) can be used. In this way it is possible to identify applications that do not have the same window name at every start (e.g. access level, time in the window name, etc.).

Value range of the parameter:
0 ➔ Wildcard search is deactivated
1 ➔ Wildcard search is activated

Example:
An application is to be integrated that contains the start time in the window name.

oemframe.ini:

```
[MyQtTest]
fFindWindowWithWildcards=1
```

systemconfiguration.ini:

```
[processes]
PROC500=process:=ProcessOEM, cmdline:="C:\\Program Files
(x86)\\Siemens\\MotionControl\\oem\\sinumerik\\hmi\\appl\\MyQtTest.exe",
oemframe:=true, windowname:="MyQtTest - ??:??:??", classname:="QWidget",
deferred:=true

[areas]
AREA500=name:=Test, process:=ProcessOEM
```

Unrestricted © Siemens AG 2024 All Rights Reserved

nPlacementMode

This parameter influences the behavior in conjunction with the Siemens Sidescreen and the Display Manager

Value range of the parameter:
0 ➔ The OEMFRAME application maximizes itself over the entire screen that is available. It is not permissible that this setting is used in conjunction with the Siemens Sidescreen and the DisplayManager.

2 ➔ The OEMFRAME application no longer maximizes itself over the entire available screen so that Siemens Sidescreen or DisplayManager will not be superimposed.

3 ➔ Should be set for OEMFRAME applications, which should be displayed within the DisplayManager.

2.2.3 OEM subdirectories and versioning

OEM subdirectories

If multiple OEM applications (Operate dialog boxes or OEMFrame applications) are to be linked, these can be distributed across the OEM's own subdirectories. The advantage is the clear assignment of the files to a particular OEM application. This is especially true for the "systemconfiguration.ini" as there is a risk of it being accidentally overwritten when copying, which may hide other OEM applications.

OEM subdirectories can be created with any directory name under "/oem/sinumerik/hmi". These subdirectories are communicated in the file "systemconfiguration.ini" in the cfg directory ("/oem/sinumerik/hmi/cfg"). They are created in section "oem_dirs" with the following syntax:

OEM_<Number>=<OEM subdirectory name>

Example:

```
[oem_dirs]
OEM_1=oem_example
OEM_2=oem_new
```

When required, the appl, cfg, lng, hlp, etc. directories can be created again below these two OEM subdirectories ("/oem/sinumerik/hmi/oem_example" and "/oem/sinumerik/hmi/oem_new").

Versioning of OEM applications

To provide OEM applications with their own versioning, which can then be viewed in the "diagnostics" area under the "Version" softkey, appropriate version files should be generated (versions.xml). These must be provided by the OEM application and saved in the OEM directory ("/oem/sinumerik/hmi") or in the corresponding OEM subdirectory (e.g. "/oem/sinumerik/hmi/oem_new").

A version file must have the name "versions.xml" and is generated in a format similar to XML. As a minimum, a name and a version are required:

Example (minimum):

```
<info>
    <Name>OEM1</Name>
    <Version>1.1</Version>
</info>
```


© Siemens AG 2024 All Rights Reserved

Beyond this, there can also be an internal version and subcomponents or references to additional version files using the tag <Path>.

Example (subcomponents, internal version):

```
<info>
    <Name>OEM1</Name>
    <Version>1.1</Version>
    <InternalVersion>999</InternalVersion>
    <Component>
        <Name>OEM1 componentX</Name>
        <Version>6.1</Version>
    </Component>
    <Component>
        <Name>OEM1 componentY</Name>
        <Version>00.00</Version>
        <InternalVersion>L00.00.06.00</InternalVersion>
    </Component>
</info>
```

Example (path):

```
<info>
    <Name>OEM1</Name>
    <Version>1.1</Version>
    <Component>
        <Name>OEM1 componentX</Name>
        <Version>6.1</Version>
        <Path>appl</Path>
    </Component>
</info>
```

During startup, the OEM directory as well as possibly existing OEM subdirectories are scanned and a higher-level version file is created in the "/oem" directory.

Note

The higher-level version file is only created if such a version file does not exist under "/oem". This means that it must be deleted when a new OEM application is installed.

References to the OEM applications from the OEM directory or the OEM subdirectories are entered into this higher-level version file. If OEM subdirectories and the OEM directory are used simultaneously, it should be ensured that the entries of the version files do not overwrite one another.

Versioning of INI files

To be able to also display the versions of INI files, the following two lines are required at the start of the INI file:

```
;VERSION: <Version> ;DATE: YYYY-MM-DD
;CHANGE: <Version> ;DATE: YYYY-MM-DD
```

Example :

```
;VERSION: 06.02.08 ;DATE: 2002-01-07
;CHANGE:  06.02.04 ;DATE: 2001-07-23
```

Unrestricted © Siemens AG 2024 All Rights Reserved

In addition, a "versions.xml" with the following contents should be saved in the cfg directory of the corresponding INI file.

```
<info defaultFileType=" *.ini"
      defaultFileVersion="06.02.08" LinkName="OEM">
    <Name>ini files</Name>
</info>
```

By specifying the "defaultFileType", the version information is directly read out from all INI files and displayed. The "defaultFileVersion" specifies the target version. If the reference and actual version do not match, this is clearly indicated in the version display by a red attention sign.

Note

The version file in the "cfg" directory is not automatically considered. Therefore, a link to it must be created in the higher-level OEM directory (or OEM subdirectory) by specifying "Path":

```
<Component>
    <Name>OEM1 componentX</Name>
    <Version>6.1</Version>
    <Path>cfg</Path>
</Component>
```


© Siemens AG 2024 All Rights Reserved

2.3 Debugging a project

2.3.1 Overview

If a project that uses the reference "Siemens.Sinumerik.Operate.Services4" is started under debugger control, the following issues must be observed.

2.3.2 Starting VisualStudio

After installation, the start menu will contain the following menu item, which is then used to start Visual Studio:

Start → Siemens Automation → Visual Studio Professional 2022

This link can also be copied to the desktop.

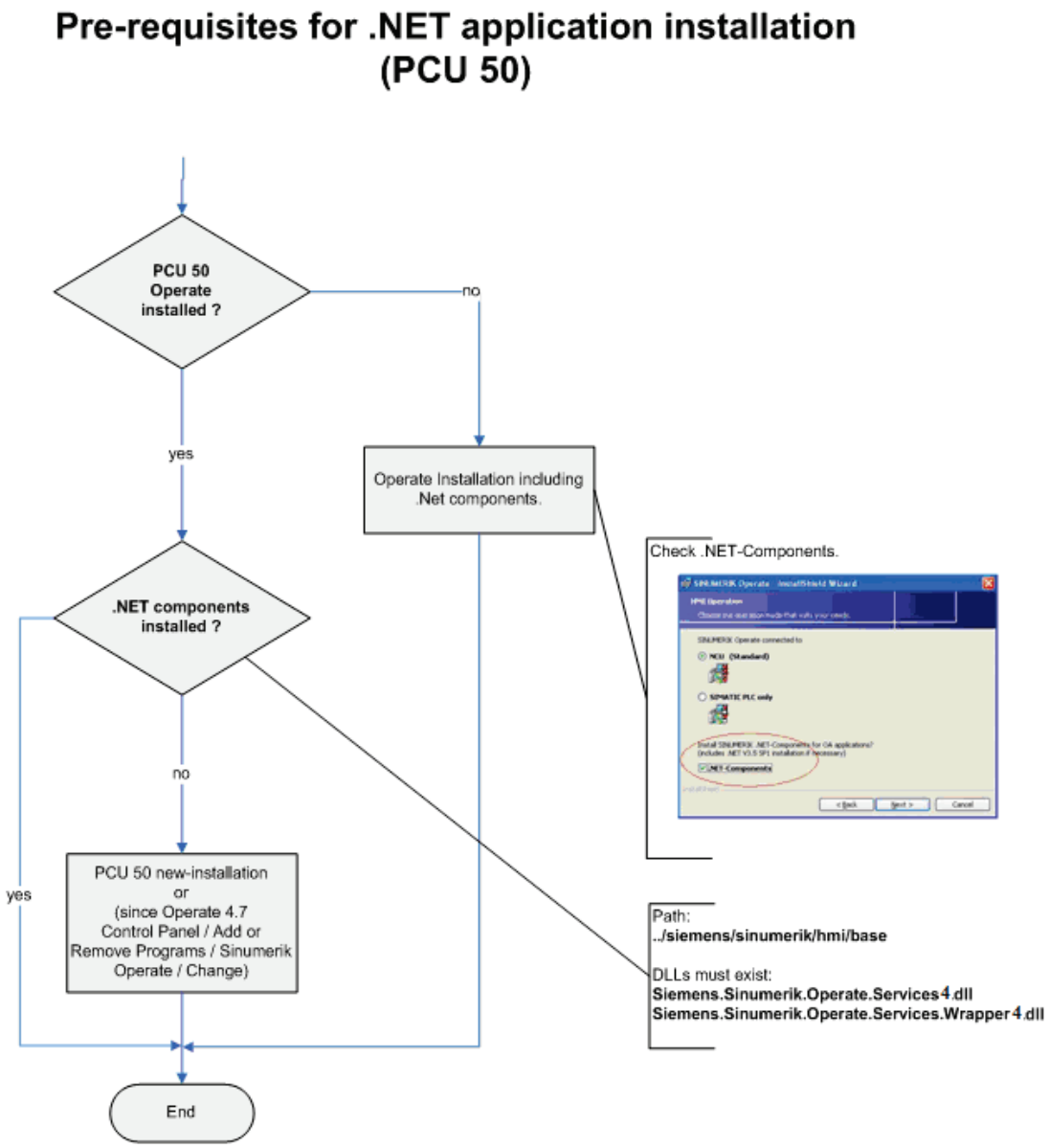
2.3.3 Debugging

Visual Studio must be started as described above. Then SINUMERIK Operate must be started. As soon as a connection has been made to the control, the application can be started in Visual Studio under debugger control.

Unrestricted © Siemens AG 2024 All Rights Reserved

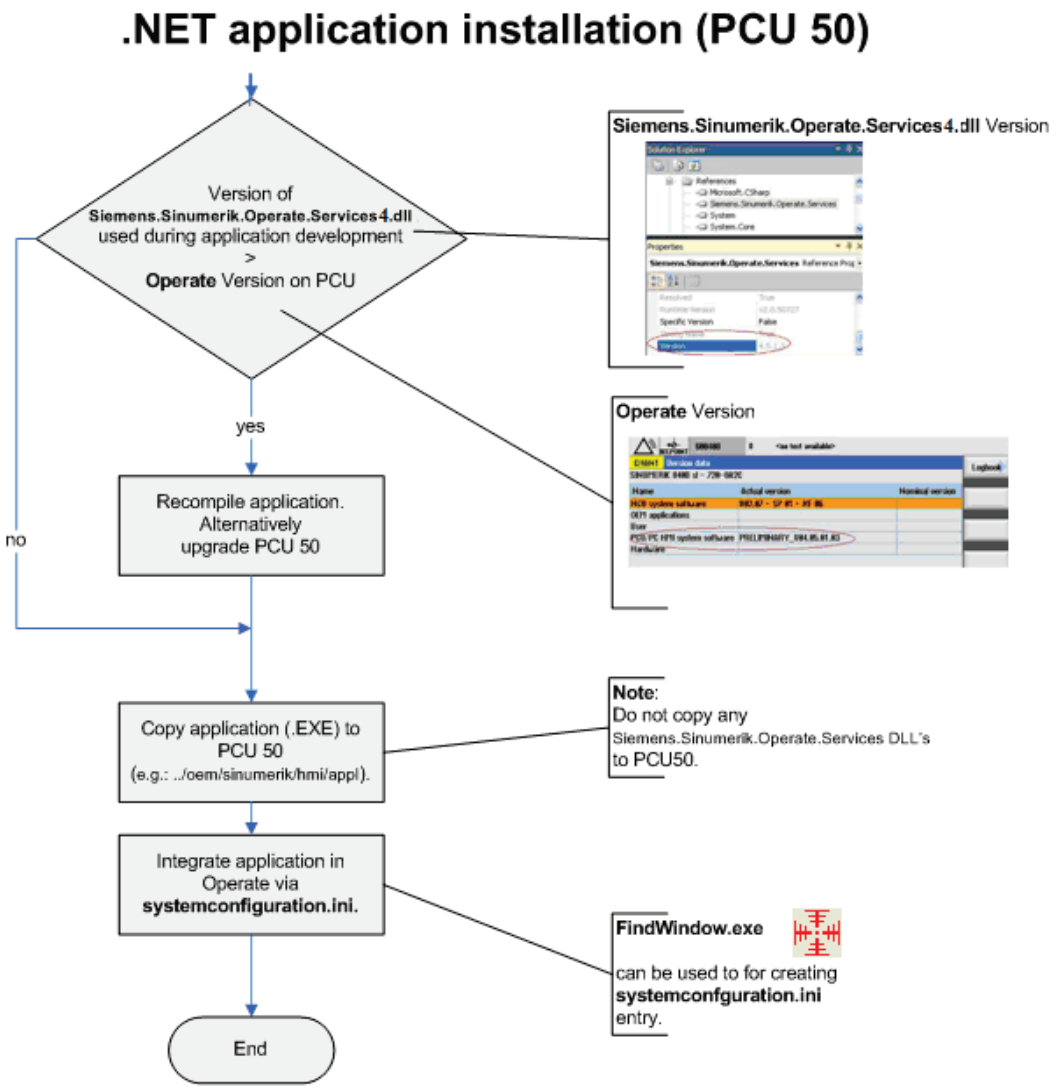
2.4 Installation on PCU50

2.4.1 Requirements



© Siemens AG 2024 All Rights Reserved

2.4.2 Installation



Unrestricted © Siemens AG 2024 All Rights Reserved

2.4.3 Certificates

Note

Concerns all PCU50/IPC with Windows 7 without Internet access.

All of the libraries supplied with SINUMERIK Operate contain a certificate. .NET runtime attempts to check the validity of the certificate, and to do this wants to establish a connection to the Internet. If there is no Internet access, then .NET runtime waits to the end of the timeout (approx. 15-20 seconds). The included application consequently starts displayed.

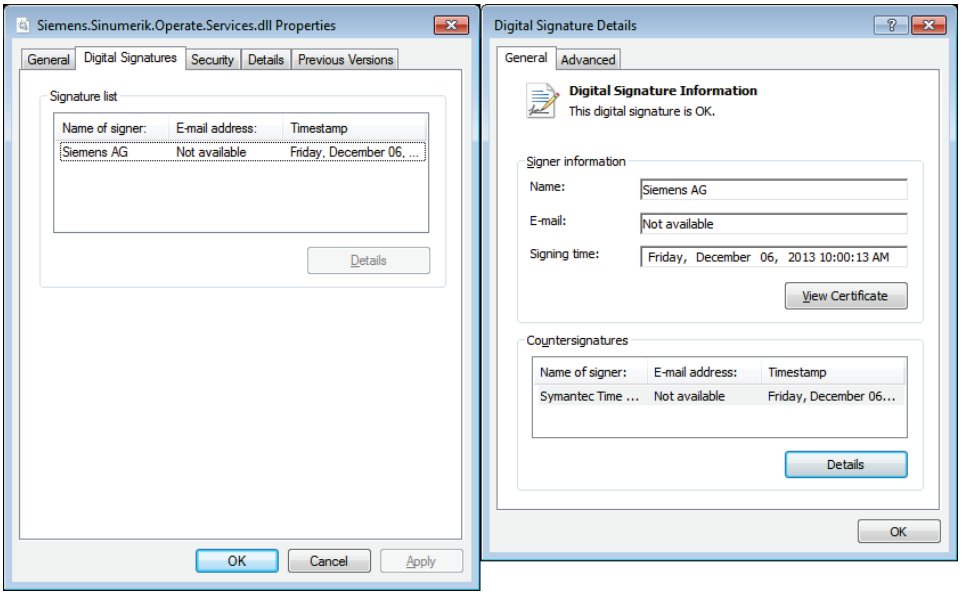


Fig. 2-6: SINUMERIK Operate .NET certificates

The following two possibilities can be used to avoid delaying the application start:

a) Workaround with configuration file
The following entry in the App.config (e.g. "MyApp.exe.config") disables the certificate test.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <runtime>
    <generatePublisherEvidence enabled="false"/>
  </runtime>
</configuration>
```

See also:
http://msdn.microsoft.com/en-us/library/cc656914(v=vs.110).aspx
(Optimize Authenticode)

b) Workaround with Internet Explorer setting
Remove the option "Check for publisher's certificate revocation" in the Internet options.

© Siemens AG 2024 All Rights Reserved

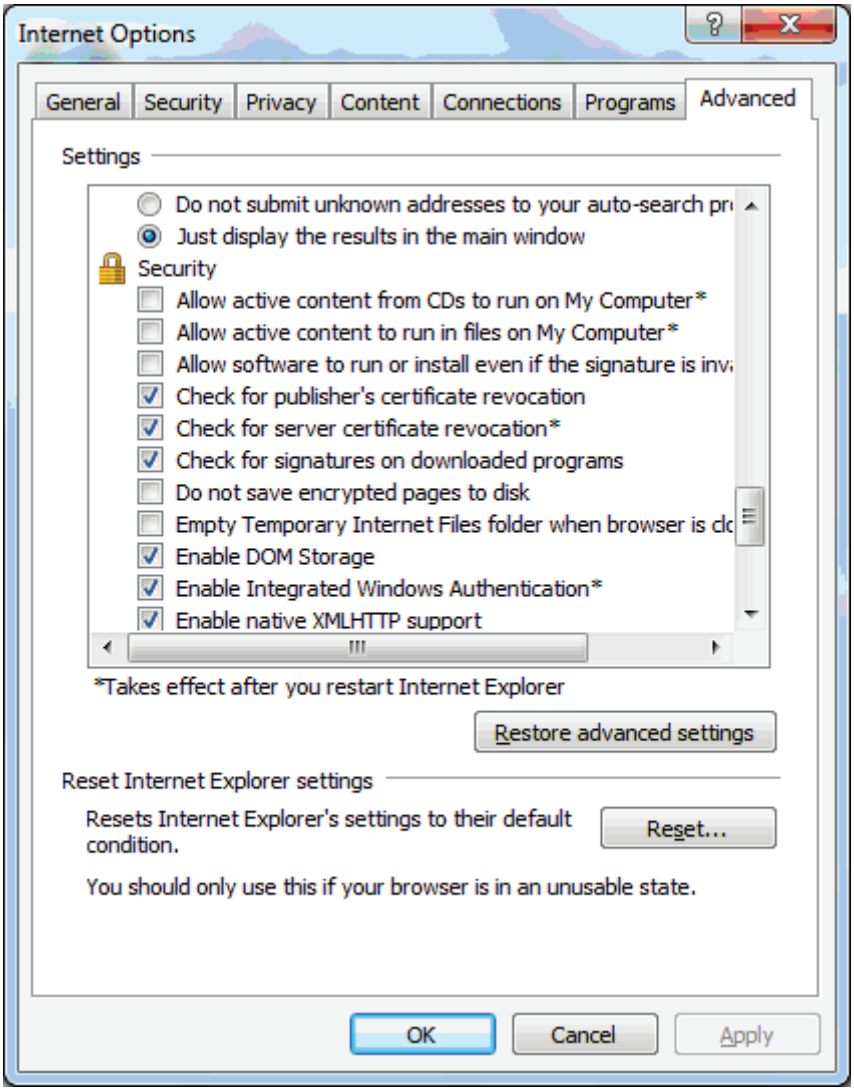


Fig. 2-7: Internet options

Unrestricted © Siemens AG 2024 All Rights Reserved

2.5 FAQs

2.5.1 FAQs regarding the integration in Sinumerik Operate

Why is my application not displayed in a full screen?

In this case, the window of the application does not accept a maximized start. One of the following points could help rectify the situation:

1. In the file oemframe.ini, for the application involved, set parameter nShowMode=3 (maximized start – SW_SHOWMAXIMIZED).
2. In Visual Studio, for the form involved, set the property "StartPosition" to the value "Manual".

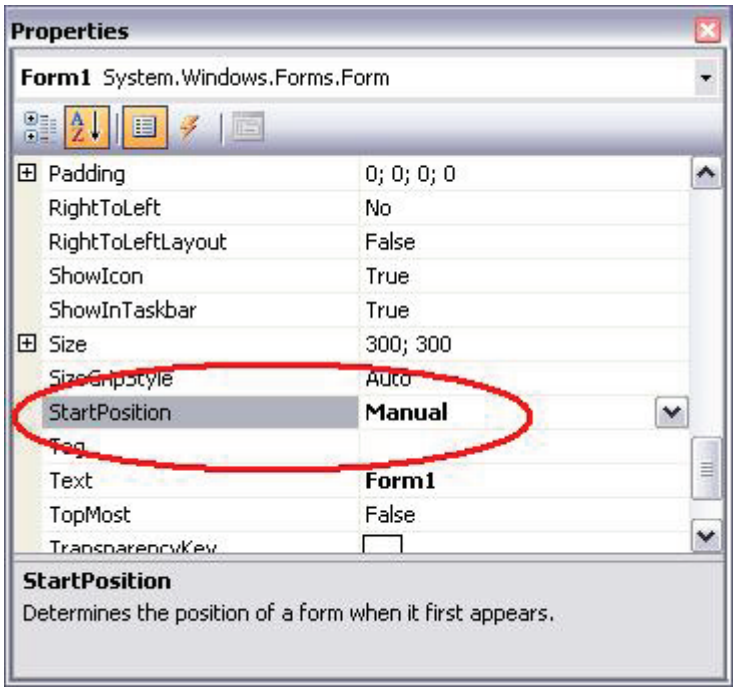


Fig. 2-8: Setting the start position for the form

© Siemens AG 2024 All Rights Reserved

2.5.2 FAQs on debugging

When starting my application, why do I always get a FileNotFoundException?

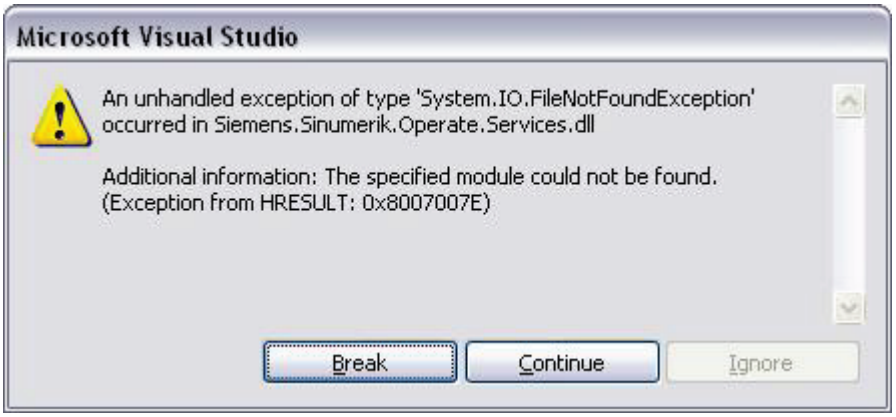


Fig. 2-9: FileNotFoundException

Visual Studio was not started via the link provided. However, once Visual Studio is started via the supplied link, then the exception is no longer displayed.

The error message also occurs when the .NET application is to be started directly via the EXE file. Note that the .NET application has to be integrated in SINUMERIK Operate by means of the systemcofiguration.ini. It cannot be started directly by double-clicking even when SINUMERIK Operate is running in the background.

Unrestricted © Siemens AG 2024 All Rights Reserved

© Siemens AG 2024 All Rights Reserved

3

3 Communication with the NC/PLC

Objective of the chapter

This chapter describes the DataService interface. This is used for communication between applications and the NC/PLC.

Note

The communication time is not guaranteed. Real-time tasks therefore cannot be handled with SINUMERIK Operate .NET.

Unrestricted © Siemens AG 2024 All Rights Reserved

3.1 Introduction

3.1.1 Class model

Overview

The class model of the DataService essentially comprises the following classes:

- DataSvc
- Item
- DataSvcStatus

Namespace

All classes are defined in the "Siemens.Sinumerik.Operate.Services" namespace. As a consequence, this namespace can be integrated into the C# project at the start of the particular file:

```
using Siemens.Sinumerik.Operate.Services;
```

Class DataSvc

Data accesses to the NC and PLC are implemented using DataSvc objects. Especially variables can be read and written, or you can be notified if a value changes.

Class Item

The item object is used for the individual calls of the DataSvc object to identify variables. The value to be written or read is also included in this object.

Class DataSvcStatus

Additional data regarding the currently executed DataSvc request can be queried using the DataSvcStatus object. These include, for example, the time stamp or the status of the current hotlink connection.

3.1.2 Explanation of terms

Synchronous calls

Synchronous calls only return after the request has been executed, i.e. the calling thread is blocked during execution. This can interfere with event processing. For example, operator inputs and displays may be withheld during a synchronous call. This is the reason why calls that could take a long time should be executed asynchronously.

Asynchronous calls

Asynchronous calls and hotlinks return as soon as the request has been sent to the DataService. The returned error code cannot therefore indicate whether the request has been completed successfully; it can only indicate whether the request has been sent successfully. For example, an error occurs if call parameters are not correctly assigned. The actual request status is provided in the callbacks of the DataService.

© Siemens AG 2024 All Rights Reserved

Single call

Single calls access a variable with read or write access.
The following single calls are available:

Table 3-1: Single calls

DataSvc call	Description
Read	Synchronous reading of a variable
ReadAsync	Asynchronous reading of a variable
Write	Synchronous writing to a variable
WriteAsync	Asynchronous writing to a variable

Multi-variable call

In a multi-variable call, multiple variables are read or written. In this case, the variables of a request can address completely different data. The following multi-variable calls are available:

Table 3-2: Multi-variable calls

DataSvc call	Description
Read	Synchronous reading of multiple variables
ReadAsync	Asynchronous reading of multiple variables
Write	Synchronous writing to multiple variables
WriteAsync	Asynchronous writing to multiple variables

Note

The function names of the multi-variable calls do not differ from those of the single calls. Only their call parameters are different.

Hotlink

A hotlink is a notification of a change to one or more variables. Notification is realized by calling a delegate. The caller must implement this delegate.
The first notification provides the value at the time the hotlink was set up.

DataService sends notifications as soon as it has identified a value change. In this case, intermediate values may be lost, but not the final value.
The following calls are available:

Table 3-3: Hotlink calls

DataSvc call	Description
Subscribe	Set up notification when one or several variables change

Array access

For some variables, the option exists in the variable path to specify several in the variable name. This is known as an array access.
Array accesses accelerate data access, therefore improving the speed of the overall system, as the communication resources required are significantly reduced.
For example, an array access to three consecutive R-parameters (R7, R8, R9) would be

```
"/channel/parameter/r[u1,7,#3]"
```

Unrestricted © Siemens AG 2024 All Rights Reserved

Connection monitoring

The following variables should be used to monitor the connection. The return value 30 indicates that a connection exists.

Table 3-4: Items for connection monitoring

Variable	Description
/nc/connect_state	Connection monitoring to the NC
/plc/connect_state	Connection monitoring to the PLC

In addition, for the NCK, variable "nck/state/anpowerOnState" can be monitored. The NCK is ready for communication as soon as 16777223 (decimal) is returned here.

Note

Previously, the variable "/bag/state/opmode" or "/Nck/Configuration/maxNumChannels" was used to monitor the connection to the NCK and variable "ib0" to the PLC. However, this is no longer recommended as under certain circumstances this procedure is no longer sufficient.

Control type

Using machine data 19010 "/NC/_N_NC_TEA_ACX/$ON_SYSTEM_INFO[1]", you can determine with which control type you are connected:
5 ➔ SINUMERIK 828D
6 ➔ SINUMERIK 840D sl
8 ➔ SINUMERIK ONE
32 ➔ SINUMERIK motion control

Using machine data 18965 "/NC/_N_NC_TEA_ACX/$MN_SIM_ENVIRONMENT", you can determine whether it involves a virtual environment.

A detailed description of these two machine data is provided in the Machine Data Manual for the corresponding control.

Completeness of signaled value changes

Setting up a hotlink (subscribe calls) does not guarantee that all changes are signaled. The reasons for this are as follows:

1. The values are read cyclically by the NCK. Value changes continue to be signaled if the read value differs from the previously signaled value. Value changes are not detected by this mechanism if a value changes and then changes back to the starting value between two consecutive sampling points.
2. The sampling times do not have to be isochronous with respect to time. NCK/PLC may suppress the continued signaling of value changes if the real-time area would otherwise be disturbed or if the communication connection is overloaded.
3. An attempt is made to combine the largest possible number of values in one communication protocol unit (PDU) in order to reduce the data volume for the applications.
4. Changes can be suppressed if a processing backup is detected on the client side.

© Siemens AG 2024 All Rights Reserved

In summary, the following is applicable:
The DataService is not in a position to detect all value changes of a variable. In addition, the required sampling rates are not guaranteed; rather, they should be viewed as a target specification. However, the DataService will attempt to continue sending the client all value changes it has detected provided the response capability of the client is sufficient. In this case, the client has no real-time requirements, but it must prevent an ever-increasing backup from occurring. The DataService guarantees that the final value of a series of value changes is always signaled after a certain time delay.

Unrestricted © Siemens AG 2024 All Rights Reserved

3.2 Interactive test interface

Overview

The interactive "slcaptest.exe" test program can be used to check data access operations.
This enables application programmers to monitor variables or to check for their existence, for example

Application

The test program is started using the following start menu entry:

```
Start → Siemens Automation → slCapTest (-colocated)
```

The parameter `-colocated` also loads and starts a CAP service, which is required for communication. Further, the following should still be started:

```
Start → Siemens Automation → cp_840Di
```

After the test program is started the first time, the root screen displays a page with several lines, each of which can perform one access to the CAP service. This page is named "DefaultTab" and belongs to the "DefaultGroup". The "Add Tab..." button can be used to add additional pages as well as additional groups. In this way, various test scenarios can be separated from one another more effectively.

If the test program is ended, the current parameter settings are automatically saved.

Buttons

The buttons at the bottom of the screen perform the following actions:

1. "AddTab ..."
   Generates a new page (tab) in a group. If the entered group does not yet exist, a new group is generated. The names are limited to alphanumeric characters and "_".

2. "DelTab"
   Deletes the currently displayed page with its entry lines. If the deleted page is the last page in a group, the group is also deleted.

3. "Clear"
   Deletes all input fields of the displayed page and brings the buttons to the initial state.

4. "DoitAll"
   Activates all buttons on the displayed page once, provided the variable path is not empty. The buttons are activated from top to bottom.

The list box at the bottom of the screen enables the selection of groups.

© Siemens AG 2024 All Rights Reserved

Operations

An operation can be selected by pressing the first button on a line multiple times. The following options exist:

1. "Hotlink"
   Enter a variable path into the left entry field of the line. Then, press the "Start" button to set up the hotlink. Press the "Stop" button to end this. If the hotlink is active, the current value is displayed in the right entry field.

2. "Read"
   Enter a variable path into the left entry field of the line. Then, press the "Doit" button to start an asynchronous call for reading the variable. The read value is displayed in the right entry field.

3. "Write"
   Enter a variable path into the left entry field of the line and a value to be written in the right entry field. Then, press the "Doit" button to start an asynchronous call for writing the variable.

If an error occurs when a call is executed, the hexadecimal error code is displayed next to the two entry fields. The "Read" and "Hotlink" operations return a "#" in this case.

Note

If the cursor is placed over the field of the hexadecimal error display, the equivalent error text appears in the form of a tooltip. If the cursor is placed over the read value, additional information is displayed.

Examples

A few application examples of the "slcaptest.exe" test program are provided below:

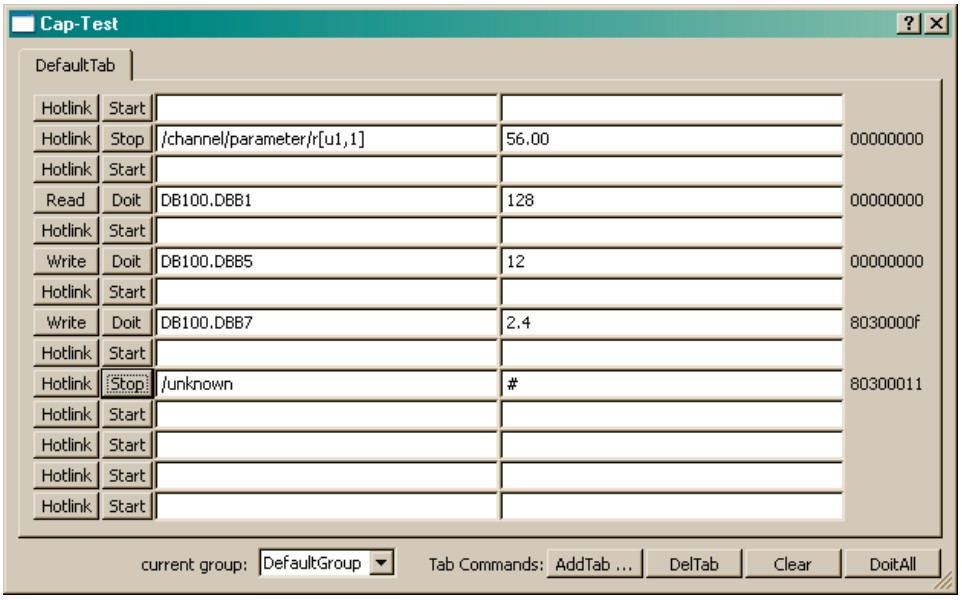


Fig. 3-1: Root screen for "slcaptest.exe"

Unrestricted © Siemens AG 2024 All Rights Reserved

The lines in Fig. 3-1: Root screen for "slcaptest.exe", explain the individual items:

1. A hotlink has been started to monitor the R-parameter R[1] of the first channel. The current value is 56. If the value changes, the display also changes.

2. The second byte of data block 100 has been read out once. The read value was 128.

3. The value 12 has been successfully written to the sixth byte of data block 100.

4. An attempt has been made to write a floating-point number to a byte. This failed, and the return value is 8030000f.

5. An attempt has been made to set up a hotlink to an unknown variable. This failed, and the return value is 803000011.

© Siemens AG 2024 All Rights Reserved

3.3 Step-by-step examples

Overview

The following chapters show the various applications for the DataService step-by-step. An executable example is included in the SINUMERIK Integrate Create MyHMI /3GL for each "step-by-step example".
These and all other examples are shown in the following overview of examples.

Table 3-5: Overview of the examples

Application	Example	Chapter
Synchronous reading/writing of a variable	DataSvcReadWriteChange	3.3.2
Asynchronous reading/writing of a variable	DataSvcReadWriteAsync	3.3.3
Array access	DataSvcArrayReadWriteChange	3.3.5
Reading/writing multiple variables	DataSvcMultiReadWriteChange	3.3.6
Hotlink to a variable	DataSvcReadWriteChange	3.3.4
Hotlink to multiple variables	DataSvcMultiReadWriteChange	-
Hotlink to a variable with several delegates	DataSvcReadWriteChange2Delegate	-

Only the items relevant for the DataService are explained or described with an example. The comments for the sources contain more detailed information (e.g. screen layout, outputs, etc.).

3.3.1 Preparation

Overview

Preparatory measures are described in Chapter 2.1 ➔ "New Project". All of the items described there must be executed first.

3.3.2 Synchronous reading/writing of a variable

Overview

This example shows how data can be read out of and written to a control synchronously using the DataService interface.

In this example, you can enter an item and then read it using the "read data" button or write it using the "write data" button. The status bar indicates whether the calls have been successful.

Unrestricted © Siemens AG 2024 All Rights Reserved

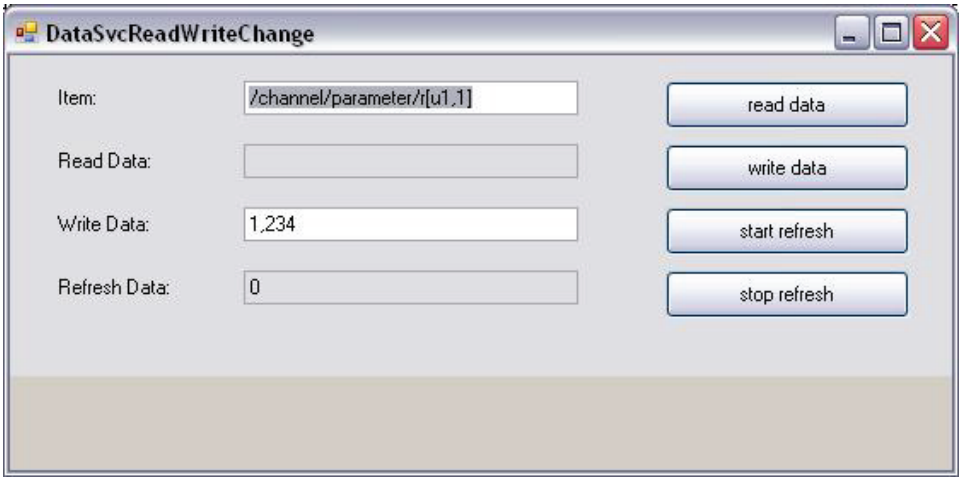


Fig. 3-2: Example, DataSvcReadWriteChange

Step 1

Create a member variable type DataSvc:

```
DataSvc m_DataSvcReadWrite = null;
```

To read and write, it is not absolutely necessary to create the DataSvc object as member variable. It can also be created locally in the specific function.

Step 2

Generate the DataSvc object in the function frmMain().

```
m_DataSvcReadWrite = new DataSvc();
```

Step 3

Read the control value. An item-type object is first created for this purpose. When creating, the variable path to be read is transferred (in the example, the contents of the text field are transferred; here, the path is directly used).
The item is read using the function Read() of the DataSvc object. The read value is in the item object in the Property Value.

```
// create item object and initialize itempath
Item itemRead = new Item("/channel/parameter/r[u1,1]");

// read the value from the control
m_DataSvcReadWrite.Read(itemRead);
```

Step 4

Write the control value. An item-type object is first created for this purpose. When creating, the variable path to be written is transferred (in the example, the contents of the text field are transferred; here, the path is directly used). After this, at the item object, the Property Value is assigned the value that should be written. An additional constructor is optionally available, with which both the item and the value to be written can be set.

The value is written to the control using the Write() function of the DataSvc object.

© Siemens AG 2024 All Rights Reserved

```
// create item object and initialize itempath
Item itemWrite = new Item("/channel/parameter/r[u1,1]");

// value to write
itemWrite.Value = 1.2345;

// write the item to the control
m_DataSvcReadWrite.Write(itemWrite);
```

3.3.3 Asynchronous reading/writing of a variable

Overview

This example shows how data can be read out of and written to a control asynchronously using the DataService interface.

In this example, you can enter an item and then read it using the "read data" button or write it using the "write data" button. The status bar indicates whether the calls have been successful.

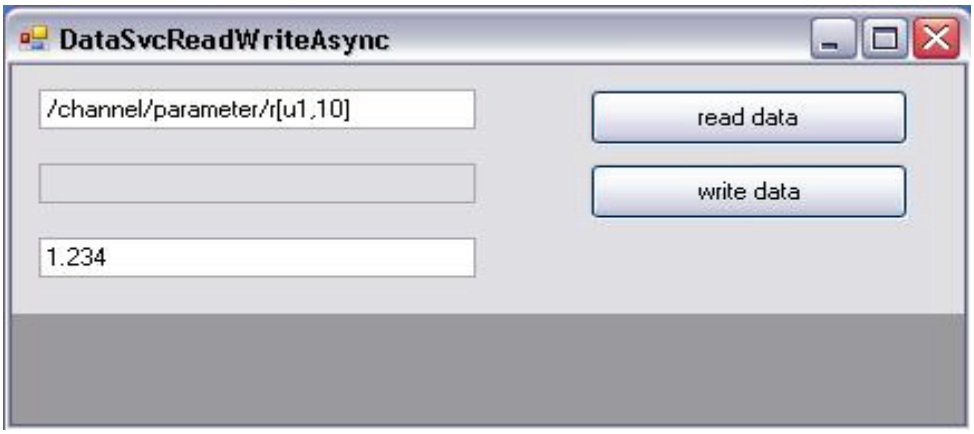


Fig. 3-3: Example: DataSvcReadWriteAsync

Step 1

Create one DataSvc type member variable and two Guid type variables to save the return ID in the case of asynchronous calls.

```
DataSvc m_DataSvcReadWrite = null;

Guid m_GuidWriteAsync;
Guid m_GuidReadAsync;
```

Step 2

Generate the DataSvc object in the function frmMain().

```
m_DataSvcReadWrite = new DataSvc();
```

Unrestricted © Siemens AG 2024 All Rights Reserved

Step 3

Start asynchronous readout of a variable value. An item-type object is first created for this purpose. When this object is created, the variable path to be read is transferred to it (in the example, the content of the text field is transferred; here, the path is used directly).
The read request is set up using the ReadAsync() function of the DataSvc object. The item object and the implemented function of the delegate are transferred as parameters.

```
// create item object and initialize path
Item itemRead = new Item("/channel/parameter/r[u1,10]");

// read the value from the control
m_GuidReadAsync = m_DataSvcReadWrite.ReadAsync(itemRead, OnItemRead);
```

Step 4

Implement the delegate.

! Important

Because the delegate is called from another thread, not all display elements can be accessed. Consequently, it is desirable to call a second delegate with the same signature with which changes can be performed on the user interface.

In the implemented delegate function, a change is sent to the same delegate signature, now in the GUI thread. Here, the changed value is displayed at the interface. The value read is contained in the Item-type object. The status parameter returns additional information about the read operation.

```
public void OnItemRead(Guid guid, Item item, DataSvcStatus status)
{
    // are we in GUI thread (main) ?
    // No we are not in GUI thread => send message to GUI thread
    if (txtReadData.InvokeRequired)
    {
        ItemRead itemDelegate = new ItemRead(OnItemRead);
        BeginInvoke(itemDelegate, new Object[] { guid, item, status });
    }
    else
    {
        if (m_GuidReadAsync == guid)
        {
            // set value to the output textbox
            txtReadData.Text = item.Value.ToString();

            // set status to statusbar
            setStatus("ReadStatus: " + status.Ok.ToString() +
                      " ; ErrorNr: " + status.ErrorNo.ToString());
        }
    }
}
```


© Siemens AG 2024 All Rights Reserved

Step 5

Start asynchronous writing of a variable value. An item-type object is first created for this purpose. This object is transferred when the variable path and variable value to be written are created. (In the example, the values are taken from text boxes; here, the values are entered directly.)
The write request is set up using the WriteAsync() function of the DataSvc object. The item object and the implemented function of the delegate are transferred.

```
// create item object and initialize path and value with text from textbox
Item itemWrite = new Item("/channel/parameter/r[u1,10]", "1.234");
// write the value to the control
m_GuidWriteAsync = m_DataSvcReadWrite.WriteAsync(itemWrite, OnItemWrote);
```

Step 6

Implement the delegate for feedback.

Important

Because the delegate is called from another thread, not all display elements can be accessed. Consequently, it is desirable to call a second delegate with the same signature with which changes can be performed on the user interface.

In the implemented delegate function, a change is sent to the same delegate signature, now in the GUI thread. This is used for feedback for the write request. The item parameter contains the variable that was written. The status parameter contains additional information about the write request.

```
public void OnItemWrote(Guid guid, Item item, DataSvcStatus status)
{
    // are we in GUI thread (main) ?
    // No we are not in GUI thread => send message to GUI thread
    if (txtReadData.InvokeRequired)
    {
        ItemWrite itemDelegate = new ItemWrite(OnItemWrote);
        BeginInvoke(itemDelegate, new Object[] { guid, item, status });
    }
    else
    {
        if (m_GuidWriteAsync == guid)
        {
            // set status to statusbar
            setStatus("WriteStatus: " + status.Ok.ToString() +
                      " ; ErrorNr: "  + status.ErrorNo.ToString());
        }
    }
}
```

Unrestricted © Siemens AG 2024 All Rights Reserved

3.3.4 Hotlink to a variable

Overview

The following step-by-step example shows how to set up a hotlink to a variable. In the example, you can enter an item and then set up a hotlink using the "start refresh" softkey, which can then be exited again using "stop refresh". The status bar indicates whether the calls have been successful.

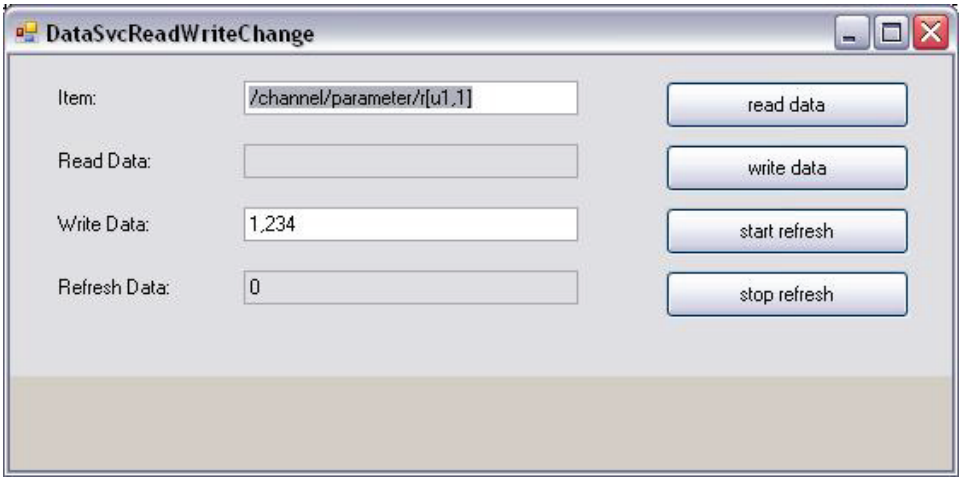


Fig. 3-4: Example, DataSvcReadWriteChange

Step 1

Create the required member variables. For the hotlink, in addition to the DataSvc object, a Guid is required to identify the hotlink:

```
DataSvc m_DataSvcSubscribe = null;
Guid m_GuidSubscribe;
```

Step 2

Generate the DataSvc object in the function frmMain().

```
m_DataSvcSubscribe = new DataSvc();
```

Step 3

Create the hotlink. An item-type object is first created for this purpose. When creating, the variable path to be read is transferred (in the example, the contents of the text field are transferred; here, the path is directly used).
The hotlink is set up using the function Subscribe() of the DataSvc object. Here, the implemented function of the delegate is transferred and also the item object.

```
// create item object and initialize itempath
Item itemSubscribe = new Item("/channel/parameter/r[u1,1]");

// set the callback function for the subscription and get the guid
m_GuidSubscribe = m_DataSvcSubscribe.Subscribe( OnDataChanged,
                                                itemSubscribe);
```

Step 4

Implement the delegate.

© Siemens AG 2024 All Rights Reserved

Important

Because the delegate is called from another thread, not all display elements can be accessed. Consequently, it is desirable to call a second delegate with the same signature with which changes can be performed on the user interface.

In the implemented delegate function, a change is sent to the same delegate signature, now in the GUI thread. Here, the changed value is displayed at the interface.

```
public void OnDataChanged(Guid guid, Item item, DataSvcStatus status)
{
    // are we in GUI thread (main) ?
    // No we are not in GUI thread => send message to GUI thread
    if (txtRefreshData.InvokeRequired)
    {
        ItemChanged itemDelegate = new ItemChanged(OnDataChanged);
        BeginInvoke(itemDelegate, new Object[] { guid, item, status });
    }
    else
    {
        // set value to the output textbox
         txtRefreshData.Text = item.Value.ToString();
    }
}
```

Step 5

Unsubscribe the running hotlink. This is realized using the UnSubscribe function.

```
m_DataSvcSubscribe.UnSubscribe(OnDataChanged);
```

3.3.5 Reading/writing multiple variables – "array access"

Overview

The following step-by-step example shows you how to read and write multiple data items simultaneously. In this case, the data must be located contiguously in the control, for example, a consecutive number of R-parameters or PLC bytes.

In this example, you can enter an item and then read it using the "read data" button or write it using the "write data" button. The status bar indicates whether the calls have been successful.

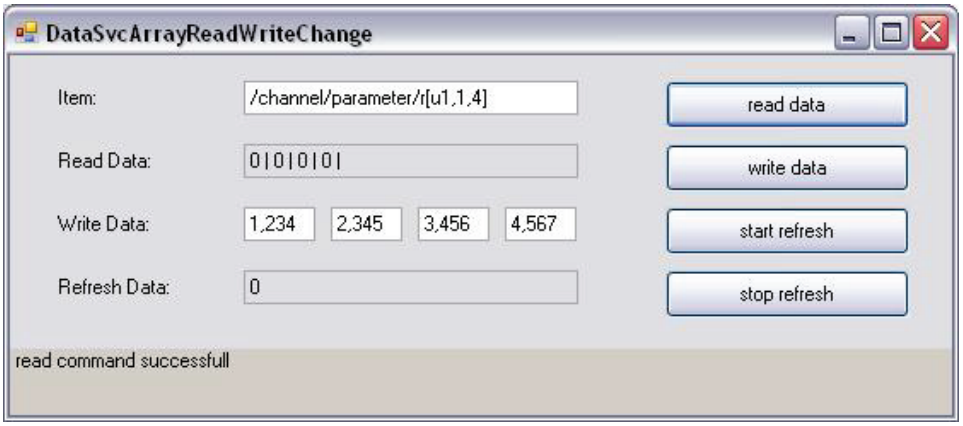


Fig. 3-5: Example, DataSvcArrayReadWriteChange

Unrestricted © Siemens AG 2024 All Rights Reserved

Step 1

Create a member variable type DataSvc:

```
DataSvc m_DataSvcReadWrite = null;
```

To read and write, it is not absolutely necessary to create the DataSvc object as member variable. It can also be created locally in the specific function.

Step 2

Generate the DataSvc object in the function frmMain().

```
m_DataSvcReadWrite = new DataSvc();
```

Step 3

Read the control values. An item-type object is first created for this purpose. When creating, the variable path to be read is transferred (in the example, the contents of the text field are transferred; here, the path is directly used).
The values are read using the Read() function of the DataSvc object. The read value is in the item object in the Property Value.

```
// create item object and initialize itempath
Item itemRead = new Item("/channel/parameter/r[u1,1,4]");

// read the value from the control
m_DataSvcReadWrite.Read(itemRead);
```

Step 4

Process the data that has been read. To start, a check is made as to whether the Property Value of the item object contains an array. If it does, the contents of the property can be converted to an array. After this, using a foreach loop, all read values are combined in a string and separated with a '|'.
This string is displayed at the end.

```
if (itemRead.Value is Array)
{
    System.String sText = "";
    System.Array theValues = (System.Array)itemRead.Value;
    foreach (System.Object o in theValues)
    {
        sText += (Convert.ToDouble(o)).ToString() + " | ";
    }
    txtReadData.Text = sText;
}
```

Step 5

Write multiple control values. An item-type object is first created for this purpose. When creating, the variable path to be written is transferred (in the example, the contents of the text field are transferred; here, the path is directly used).

```
// create item object and initialize itempath
Item itemWrite = new Item(channel/parameter/r[u1,1,4]");
```

The value to be written is then processed. In this case, a type System.Object array is created, which contains the four values to be written.

```
System.Object[] oValueArray = new System.Object[]{
                                                1.111, 2.222, 3.333, 4.444};
```


© Siemens AG 2024 All Rights Reserved

The value property is then assigned the prepared array.

```
itemWrite.Value = oValueArray;
```

The values are written into the control using the Write() function of the DataSvc object.

```
// write the item to the control
m_DataSvcReadWrite.Write(itemWrite);
```

3.3.6 Reading/writing multiple variables – "multi-access"

Overview

The following step-by-step example shows you how to read and write multiple data items simultaneously. In this case, any data can be used, even mixed NC and PLC data.

In this example, you can enter two different items and then read them using the "read data" button or write them using the "write data" button. The status bar indicates whether the calls have been successful.

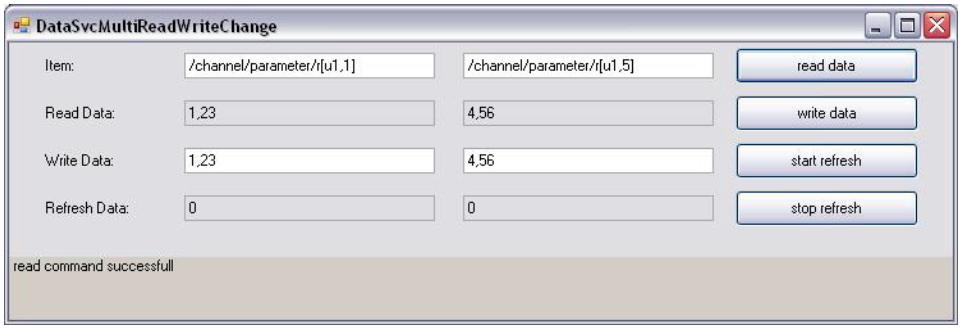


Fig. 3-6: Example, DataSvcMultiReadWriteChange

Step 1

Create a member variable type DataSvc:

```
DataSvc m_DataSvcReadWrite = null;
```

To read and write, it is not absolutely necessary to create the DataSvc object as member variable. It can also be created locally in the specific function.

Step 2

Generate the DataSvc object in the function frmMain().

```
m_DataSvcReadWrite = new DataSvc();
```

Unrestricted © Siemens AG 2024 All Rights Reserved

Step 3

Read the control values. In this case, an item type array is first created.

```
// create item object array
Item[] itemsRead = new Item[2];
```

When creating each array element (item) the variable path is transferred (in the example, the contents of the corresponding text field are transferred; here, the paths are directly used).

```
// fill items
itemsRead[0] = new Item("/channel/parameter/r[u1,1]");
itemsRead[1] = new Item("/channel/parameter/r[u1,5]");
```

The values are read using the Read() function of the DataSvc object.

```
// read the value from the control
m_DataSvcReadWrite.Read(itemsRead);
```

The read values are located as an array of items in the Property Value of the item object.

```
// output to textbox
txtReadData1.Text = itemsRead[0].Value.ToString();
txtReadData2.Text = itemsRead[1].Value.ToString();
```

Step 4

Write multiple control values. For this purpose, an item type array is again created.

```
// create item object array
Item[] itemsWrite = new Item[2];
```

When creating each array element (item) the variable path is transferred (in the example, the contents of the corresponding text field are transferred; here, the paths are directly used).

```
// fill items
itemsRead[0] = new Item("/channel/parameter/r[u1,1]");
itemsRead[1] = new Item("/channel/parameter/r[u1,5]");
```

The values to be written are now entered into the particular value property.

```
// convert value to double for the item
itemsWrite[0].Value = 1.234;
itemsWrite[1].Value = 5.678;
```

The values are written into the control using the Write() function of the DataSvc object.

```
// write the item to the control
m_DataSvcReadWrite.Write(itemsWrite);
```


© Siemens AG 2024 All Rights Reserved

3.4 Reference DataSvc

3.4.1 Definitions

Overview

Data accesses to the NC and PLC are implemented using DataSvc objects. In particular, variables can be read and written, and it is possible to be notified when a value changes.

Synchronous calls block the calling item until the request has been completed. Notification of a value change is realized using the call of the implemented delegates.

The class is defined in the "Siemens.Sinumerik.Operate.Services" namespace.

Mapping of variables

Variables are mapped as item-type objects. This item object includes both the variable path and the value.

Array access

If an access to multiple consecutive variables is specified in the variable path (array access), then DataSvc returns an array in the item object.
In C#, you can query as to whether the item object contains an array:

```
if (theItem.Value is Array)
```

The first element of the array (Index=0) corresponds to the first element of the specified item.

When writing to arrays, the data must also be packed into an item object as array.

3.4.2 Creating a DataSvc object

The DataSvc class has the following constructors.

Table 3-6: Constructors

public DataSvc.DataSvc() **public DataSvc.DataSvc(String server)**	
Parameter	**Meaning**
Server	In 1:N constellations, specifies which NCU is returning or receiving data.
Possible exceptions	DataSvcException

Unrestricted © Siemens AG 2024 All Rights Reserved

3.4.3 Reading variables

Synchronous reading of a variable (read)

Table 3-7: Read – synchronous reading of a variable

public void DataSvc.Read(Item item)	
Parameter	**Meaning**
item	Specifies the variable to be read. The read value is also in this parameter in the value property.
Possible exceptions	ArgumentException, DataSvcException, DataSvcBusyException

Example: See Chapter 3.3.2

Asynchronous reading of a variable (ReadAsync)

Table 3-8: ReadAsync – asynchronous reading of a variable

public Guid DataSvc.ReadAsync(Item item, ItemRead cb)	
Parameter	**Meaning**
item	Specifies the variable to be read.
cb	Delegate implementation for feedback of the read request.
Return value	GUID that can be used to identify the read request.
Possible exceptions	ArgumentException, DataSvcException, DataSvcBusyException

Table 3-9: ItemRead – specification for the delegate implementation

public delegate void DataSvc.ItemRead(Guid guid, Item item, DataSvcStatus status)	
Parameter	**Meaning**
guid	GUID that was returned for the ReadAsync call. It is used to identify the read request.
item	Specifies the variable whose value has been read. The read value is also in this parameter in the value property.
status	Status data for the read request. See Chapter 3.6 Reference DataSvcStatus

Example: See Chapter **3**.3.3

Synchronous reading of multiple variables (read)

Table 3-10: Read – synchronous reading of multiple variables

public void DataSvc.Read(Item[] items)	
Parameter	**Meaning**
items	Array of item objects, which specify the variable to read. The values that are read are also located in these parameters.
Possible exceptions	ArgumentException, DataSvcException, DataSvcBusyException

Example: See Chapter 3.3.6

© Siemens AG 2024 All Rights Reserved

Asynchronous reading of multiple variables (ReadAsync)

Table 3-11: ReadAsync – asynchronous reading of multiple variables

public Guid DataSvc.ReadAsync(Item[] items, ItemsRead cb)	
Parameter	**Meaning**
items	Array of item objects, which specify the variable to read.
cb	Delegate implementation for feedback of the read request.
Return value	GUID that can be used to identify the read request.
Possible exceptions	ArgumentException, DataSvcException, DataSvcBusyException

Table 3-12: ItemsRead – specification for the delegate implementation

public delegate void DataSvc.ItemsRead(Guid guid, Item[] item, DataSvcStatus[] status)	
Parameter	**Meaning**
guid	GUID that was returned for the ReadAsync call. It is used to identify the read request.
item	Specifies an array of variables whose values have been read. The read values are also in this parameter, in the Value property.
status	Array of status data for the items in the read request. See Chapter 3.6 Reference DataSvcStatus

Example: See Chapter 3.3.3

3.4.4 Writing variables

Synchronous writing to a variable (write)

Table 3-13: Write – synchronous writing to a variable

public void DataSvc.Write(Item item)	
Parameter	**Meaning**
item	Specifies the variable to be written. The value is also in this parameter in the Value property.
Possible exceptions	ArgumentException, DataSvcException, DataSvcBusyException

Example: See Chapter 3.3.2

Unrestricted © Siemens AG 2024 All Rights Reserved

Asynchronous writing to a variable (WriteAsync)

Table 3-14: WriteAsync – asynchronous writing to a variable

public Guid DataSvc.WriteAsync(Item item, ItemWrite cb)	
Parameter	**Meaning**
item	Specifies the variable to be written. The value is also in this parameter in the Value property.
cb	Delegate implementation for feedback of the write request.
Return value	GUID that can be used to identify the write request.
Possible exceptions	ArgumentException, DataSvcException, DataSvcBusyException

Table 3-15: ItemWrite – specification for the delegate implementation

public delegate void DataSvc.ItemWrite(Guid guid, Item item, DataSvcStatus status)	
Parameter	**Meaning**
guid	GUID that was returned for the WriteAsync call. It is used to identify the write request.
item	Specifies the variable whose value has been written.
status	Status data for the write request. See Chapter 3.6 Reference DataSvcStatus

Example: See Chapter 3.3.3

Synchronously writing to multiple variables (write)

Table 3-16: Write – synchronous writing to multiple variables

public void DataSvc.Write(Item[] items)	
Parameter	**Meaning**
items	Array of item objects, which specify the variables to be written to. The values are also located in this parameter.
Possible exceptions	ArgumentException, DataSvcException, DataSvcBusyException

Example: See Chapter 3.3.6

Asynchronous writing to multiple variables (WriteAsync)

Table 3-17: WriteAsync – asynchronous writing to multiple variables

public Guid DataSvc.WriteAsync(Item[] items, ItemsWrite cb)	
Parameter	**Meaning**
items	Array of item objects, which specify the variables to be written to. The values are also located in this parameter.
cb	Delegate implementation for feedback of the write request.
Return value	GUID that can be used to identify the write request.
Possible exceptions	ArgumentException, DataSvcException, DataSvcBusyException

© Siemens AG 2024 All Rights Reserved

Table 3-18: ItemsWrite – specification for the delegate implementation

public delegate void DataSvc.ItemsWrite(Guid guid, Item[] item, DataSvcStatus[] status)	
Parameter	**Meaning**
guid	GUID that was returned for the WriteAsync call. It is used to identify the write request.
item	Specifies an array of variables whose values have been written.
status	Array with the status data for the items in the write request. See Chapter 3.6 Reference DataSvcStatus

3.4.5 Notification regarding a variable change (hotlink)

Setting up a hotlink to a variable (Subscribe)

Setting up a notification when a variable changes. The call returns as soon as the setup request has been forwarded to the server. The notification is realized by calling the implementation of the delegate ItemChanged(). The first notification returns the value at the time the hotlink was set up.

Intermediate values can be lost, but not the final value (see also Chapter 3.1.2 "Definitions", completeness of signaled value changes).

Notifications can be canceled using an Unsubscribe call. The request is also canceled as soon as it has been identified that the relevant DataSvc object was deleted.

Table 3-19: Subscribe – hotlink to a variable

public Guid DataSvc.Subscribe(ItemChanged cb, Item item) **public Guid DataSvc.Subscribe(ItemChanged cb)**	
Parameter	**Meaning**
cb	Delegate implementation to notify a value change.
item	Variable, for which a notification is to be set up.
Return value	GUID, which can be used to identify a running hotlink.
Possible exceptions	ArgumentException, DataSvcException

Note

The function with the signature *Subscribe(ItemChanged cb)* can be used to implement additional delegates. This means that if a hotlink is already running, then an additional notification can be activated using this function. In this case, for the value change, two or several functions can be called.

Unrestricted © Siemens AG 2024 All Rights Reserved

Table 3-20:ItemChanged – specification for the delegate implementation

public delegate void DataSvc.ItemChanged(Guid guid, Item item, DataSvcStatus status)	
Parameter	**Meaning**
guid	GUID, which was returned for a Subscribe call. It is used to identify the running hotlink.
item	Specifies the variable, whose value has changed. The new value is also located in this parameter in the Value property.
status	Status data of the running hotlink. See Chapter 3.6 Reference DataSvcStatus

Example: See Chapter 3.3.4

Setting up a hotlink to multiple variables (Subscribe)

Setting up a notification when multiple variables change. The call returns as soon as the setup request has been forwarded to the server. The notification is realized by calling the implementation of the delegate ItemsChanged(). The first notification returns the values of all variables at the time the hotlink was set up.

Intermediate values can be lost, but not the final value. More information can be found in the chapter "Completeness of signaled value changes".

Notifications can be canceled using an Unsubscribe call. The request is also canceled as soon as it has been identified that the relevant DataSvc object was deleted.

Table 3-21: Subscribe – hotlink to a variable

public Guid DataSvc.Subscribe(ItemsChanged cb, Item[] items) **public Guid DataSvc.Subscribe(ItemsChanged cb)**	
Parameter	**Meaning**
cb	Delegate implementation to notify a value change.
items	Variables, for which notifications are to be set up.
Return value	GUID, which can be used to identify a running hotlink.
Possible exceptions	ArgumentException, DataSvcException

Note

The function with the signature *Subscribe(ItemsChanged cb)* can be used to implement additional delegates. This means that if a hotlink is already running, then an additional notification can be activated using this function. In this case, for the value change, two or several functions can be called.

© Siemens AG 2024 All Rights Reserved

Table 3-22:ItemsChanged – specification for the delegate implementation

public delegate void DataSvc.ItemsChanged(Guid guid, Item[] items, DataSvcStatus[] status)	
Parameter	**Meaning**
guid	GUID, which was returned for a Subscribe call. It is used to identify the running hotlink.
items	Specifies the variable, whose value has changed. The new value is also located in this parameter in the Value property.
status	Status data of the running hotlink. See Chapter 3.6 Reference DataSvcStatus
Possible exceptions	ArgumentException, DataSvcException

Unsubscribing a hotlink (UnSubscribe)

A hotlink is ended using the UnSubscribe function.

Table 3-23:Unsubscribe – unsubscribing a hotlink

public void DataSvc.UnSubscribe(ItemChanged cb) **public void DataSvc.UnSubscribe(ItemsChanged cb)**	
Parameter	**Meaning**
cb	Delegate implementation to notify a value change.
Possible exceptions	ArgumentException, DataSvcException

3.4.6 Further properties of the DataSvc class

DeadBand

Suppressed notifications for which the value does not differ sufficiently from the preceding value. This is only active for numerical values and hotlinks.

If a value is greater than 0, no changes are forwarded provided the value does not deviate from the preceding value by the amount of the DeadBand.

A value that is less than 0 produces a grid. Values are forwarded only if the new value is within a different step.

Table 3-24:DeadBand

public double DataSvc.DeadBand { set; get; }

PriorityFlag

Processing priority of DataService accesses.

Table 3-25:PriorityFlag

public SlPriority DataSvc.PriorityFlag { set; get; }

Unrestricted © Siemens AG 2024 All Rights Reserved

DataService uses three processing priorities:

`SlPriority.lowPriority`
Low processing priority. This value should be used for cyclic activities and activities running in the background. By default, the DataService uses this priority for Subscribe requests once the first value has arrived.

`SlPriority.highPriority`
High processing priority. This value is reserved for exclusive use by the machine manufacturer to achieve productivity goals (e.g. tool change).

`SlPriority.defaultPriority`
Default processing priority. This flag is intended for user interaction and screen rebuilding. By default, DataService executes all read and write requests in this priority. This priority is also used when accessing the first value for Subscribe requests.

Please note the following constraints:

1. As requests with high processing priority prevent requests with low processing priority from being processed, only the priority `lowPriority` may be used for cyclic requests (hotlinks).

2. The processing priorities are active only within the DataService. Low-priority requests that have already started will no longer be overtaken by higher-priority requests.

Note

The flag `defaultPriority` (value = 0) does not have to be explicitly set. This value is automatically set after creating a DataSvc object.

Server

Currently not relevant.

SynchronizedMultivarAccess

Occasionally, multiple individual data refer to each other and have to be read or written simultaneously. However, the DataService does not guarantee that the variables of a multi-variable call will be sent in one request. In particular, if there is a high communication load, it attempts to optimize the throughput instead.

This can be influenced using the SynchronizedMultivarAccess property:

Table 3-26:SynchronizedMultivarAccess

public bool DataSvc.SynchronizedMultivarAccess { set; get; }

`SynchronizedMultivarAccess = true;`
For a multiple variable call, by setting this flag in the DataSvc object, it can be forced that the variables are sent in one request and also the result is supplied in one request.

© Siemens AG 2024 All Rights Reserved

Note that there are a number of additional constraints to be complied when setting this flag. Failure to do so will cause the entire request to be rejected. Possible causes could be:

1. The variables must be located on the same unit (PLC, NC, or drive).
2. In the case of NCK variables, channel-specific data may only address one channel and axis-specific data only one axis.
3. Request and result must not exceed the size of one PDU.

If one of these constraints is violated, then a DataSvcException is issued. More detailed information about the cause is contained in the exception text.

Note

The space available in a PDU (protocol data unit) depends on several parameters of the terminal device (NC, PLC, drive) and cannot be specified generally.

The following apply as guide values for the PLC:
Read requests: Space for at least 16 variables
Write requests: Space for at least 8 variables
Net data volume: Approx. 190 bytes

The following apply as guide values for the NC:
Read requests: Space for at least 100 variables
Write requests: Space for at least 50 variables
Net data volume: Approx. 800 bytes

Timeout

Time limit for the call duration, in milliseconds. After this time expires, the call returns with a DataSvcException.

Table 3-27:Timeout

public uint DataSvc.Timeout { set; get; }

Pre-defined constants:

`SlTimeout.standardTimeout`
Default value. This timeout value should be used for fast accesses. It takes into account potential communication delays, connection establishment, etc. (20000 milliseconds).

`SlTimeout.noTimeout`
No time monitoring

UpdateRate

Sampling rate for hotlinks (Subscribe calls).

Table 3-28:UpdateRate

public SlUpdateRate DataSvc.UpdateRate { set; get; }

Unrestricted © Siemens AG 2024 All Rights Reserved

Possible values:

`SlUpdateRate.standardUpdateRate`
Default value. This sampling rate is used for variables whose change affects the display, but that change relatively infrequently. Examples include machine data, operating modes, etc. (200 milliseconds).

`SlUpdateRate.highUpdateRate`
This sampling rate is used for variables whose change is tracked directly by the operator and that change frequently. Examples include axis positions (50 milliseconds).

`SlUpdateRate.lowUpdateRate`
This sampling rate is used for variables whose acquisition is not time-critical (1000 milliseconds).

`SlUpdateRate.veryLowUpdateRate`
This sampling rate is used for variables whose acquisition is not time-critical (10000 milliseconds).

© Siemens AG 2024 All Rights Reserved

3.5 Reference Item

3.5.1 Definitions

Overview

An instance of an item class represents a variable in the control (NC, PLC, machine or axis data. This object includes both the variable path and the value. The class is defined in the "Siemens.Sinumerik.Operate.Services" namespace.

3.5.2 Variable paths

Variable paths for NC data

You obtain the variable path for NC accesses from the OPI help. It is also included in the documentation for SINUMERIK Integrate Create MyHMI /3GL.

Table 3-29: Examples of variable paths (NC accesses)

Variable path	Description
/channel/parameter/r[u1,10]	R-parameter 10 in channel 1.
/channel/parameter/r[u1,4,#5]	R-parameter array from R4 to R8 in channel 1.
/channel/parameter/r[u2,40,42]	R-parameter array from R40 to R42 in channel 2.
/channel/geometricAxis/name[u2,3]	Name of the 3rd axis in channel 2.
/channel/geometricAxis/actToolBasePos[u1,3]	Position of the 3rd axis in channel 1.

Variable paths for GUDs and machine data

GUDs and machine data do not have to be explicitly declared. They are automatically declared when the DataSvc object is instantiated for the first time.

Table 3-30: Examples of GUDs and machine data

Variable path	Description
/NC/_N_CH_GD1_ACX/GUDNAME1[u1]	Access to variable GUDNAME1 in channel 1 (u1) of the 1st GUD area (SGUD)
/NC/_N_NC_GD4_ACX/GUDNAME2	Access to the NC global variable GUDNAME2 of the 4th GUD area
/NC/_N_NC_TEA_ACX/<NAME>	Access to NC global machine data
/NC/_N_CH_TEA_ACX/<NAME>	Access to channel-specific machine data
/NC/_N_AX_TEA_ACX/<NAME>	Access to axis-specific machine data
/NC/_N_NC_SEA_ACX/<NAME>	Access to NC global setting data
/NC/_N_CH_SEA_ACX/<NAME>	Access to channel-specific setting data
/NC/_N_AX_SEA_ACX/<NAME>	Access to axis-specific setting data

Remarks regarding the table:

1. <NAME> must be replaced by the name of the relevant machine or setting data.

GUD arrays are 0-indexed for access, and from the perspective of the DataSvc are always one-dimensional. This means, the index must be calculated for multi-dimensional arrays.

Note

Variable prefix for GUDs in uppercase letters!

Unrestricted © Siemens AG 2024 All Rights Reserved

Example 1 – one-dimensional array:

UGUD.DEF file

```
DEF NCK INT ARRAY[2]
M17
```

Access is realized as follows:

```
ARRAY[0] → /NC/_N_NC_GD3_ACX/ARRAY[0]
ARRAY[1] → /NC/_N_NC_GD3_ACX/ARRAY[1]
```

Example 2 – two-dimensional array:

UGUD.DEF file

```
DEF CHAN INT ABC[3,3]
M17
```

Access is realized as follows:

```
ABC[0,0] → /NC/_N_CH_GD3_ACX/ABC[u1, 0]
ABC[0,1] → /NC/_N_CH_GD3_ACX/ABC[u1, 1]
ABC[0,2] → /NC/_N_CH_GD3_ACX/ABC[u1, 2]
ABC[1,0] → /NC/_N_CH_GD3_ACX/ABC[u1, 3]
ABC[1,1] → /NC/_N_CH_GD3_ACX/ABC[u1, 4]
ABC[1,2] → /NC/_N_CH_GD3_ACX/ABC[u1, 5]
ABC[2,0] → /NC/_N_CH_GD3_ACX/ABC[u1, 6]
ABC[2,1] → /NC/_N_CH_GD3_ACX/ABC[u1, 7]
ABC[2,2] → /NC/_N_CH_GD3_ACX/ABC[u1, 8]
```

Variable paths for PLC accesses (840D sl)

The variable path for PLC axis operations corresponds to the S7 syntax. Both SIMATIC and IEC addressing can be used for this.

Table 3-31: PLC syntax

Area	Address (SIMATIC)	Address (IEC)	Permissible data types
Output image	Ax.y	Qx.y	BOOL
Output image	ABx	QBx	BYTE, CHAR, STRING, DT
Output image	AWx	QWx	WORD, INT, DATE, S5TIME, CHAR
Output image	ADx	QDx	DWORD, DINT, REAL, TIME, TOD
Data block	DBz.DBx.y	DBz.DBx.y	BOOL
Data block	DBz.DBXx.y	DBz.DBXx.y	BOOL
Data block	DBz.DBBx	DBz.DBBx	BYTE, CHAR, STRING, DT
Data block	DBz.DBWx	DBz.DBWx	WORD, INT, DATE, S5TIME, CHAR
Data block	DBz.DBDx	DBz.DBDx	DWORD, DINT, REAL, TIME, TOD
Input image	Ex.y	Ix.y	BOOL
Input image	EBx	IBx	BYTE, CHAR, STRING, DT
Input image	EWx	IWx	WORD, INT, DATE, S5TIME, CHAR
Input image	EDx	IDx	DWORD, DINT, REAL, TIME, TOD
Bit memory	Mx.y	Mx.y	BOOL
Bit memory	MBx	MBx	BYTE, CHAR, STRING, DT
Bit memory	MWx	MWx	WORD, INT, DATE, S5TIME, CHAR
Bit memory	MDx	MDx	DWORD, DINT, REAL, TIME, TOD
Timer	Tx	Tx	S5TIME
Counter	Zx	Cx	WORD

Remarks regarding the table:

1. In the table, "x" stands for the byte offset in the area; "y" for the bit number in the byte; and "z" for the data block.
2. The underscored data type is the default data type in each case and does not have to be specified when addressing. In addition, the specifications DB2.DBB5:BYTE and DB2.DBB5 are equivalent, for example.
3. Square brackets are used to access arrays, e.g. DB5.DBW2:[10] (word array of length 10).

© Siemens AG 2024 All Rights Reserved

Examples

Table 3-32: Examples of variable paths (PLC accesses)

Variable path	Description
M5.0	Memory bit 0 at byte offset 5.
DB5.DBW2	Word (16-bit) at byte offset 2 in data block 5.
DB5.DBW2:S5TIME	Word (16-bit) at byte offset 2 in data block 5 as S5 time.
DB8.DBB2:STRING	UTF8 string starting at byte offset 2 in data block 8.
DB8.DBW2:[10]	Array of 10 words starting at byte offset 2 in data block 8.
DB100.DBB1	Byte at byte offset 1 in data block 100.
DB100.DBW7:[5]	Array of 5 words starting at byte offset 7 in data block 100.
DB2.DBD0:REAL[10]	Array of 10 double words (32-bit) at byte offset 0 in data block 2, which should be formatted as a floating-point number.
Clock	PLC time

The following points should be noted:

- Timers can only be read. A timer is active if it contains a value other than 0.
- If the data type CHAR or STRING is used in conjunction with a byte access, UTF8 characters are read, but if either data type is used in conjunction with a word access, UTF16 characters are read.
- Variables of type STRING contain the maximum length in the first byte/word and the actual length in the second byte/word. When strings are written, the maximum length does not change.
- For data type STRING in conjunction with a byte access (e.g. DB99.DBB0:STRING) the maximum string length is 255 characters. As a result of the UTF8 formatting, for several characters (e.g. for the "µ"), two bytes are required, so that the maximum string length is correspondingly reduced.
- Only one-dimensional arrays are supported. With STRING arrays, all elements have the same maximum length.
- The ASCII value of the character is supplied for field accesses, type CHAR (e.g. DB100.DBB0:CHAR[1]).

  Alternatively, the following switch can be used in systemconfiguration.ini. The ASCII character is then supplied:

```
[miscellaneous]
CharArrayCompatibleToOldHmiPro=false
```

  access operations with 2 or several elements (e.g. DB100.DBB0:CHAR[2]) always deliver
  ASCII characters.

Unrestricted © Siemens AG 2024 All Rights Reserved

Variable paths for PLC accesses (SINUMERIK ONE)

The variable path for PLC axis operations to a SINUMERIK ONE depends on attribute "Optimized block access" in the TIA Portal. This data block can be activated or deactivated using the properties dialog.

If this is deactivated, then the syntax behaves analogously to 840D sl. Symbolic addressing can also be used:

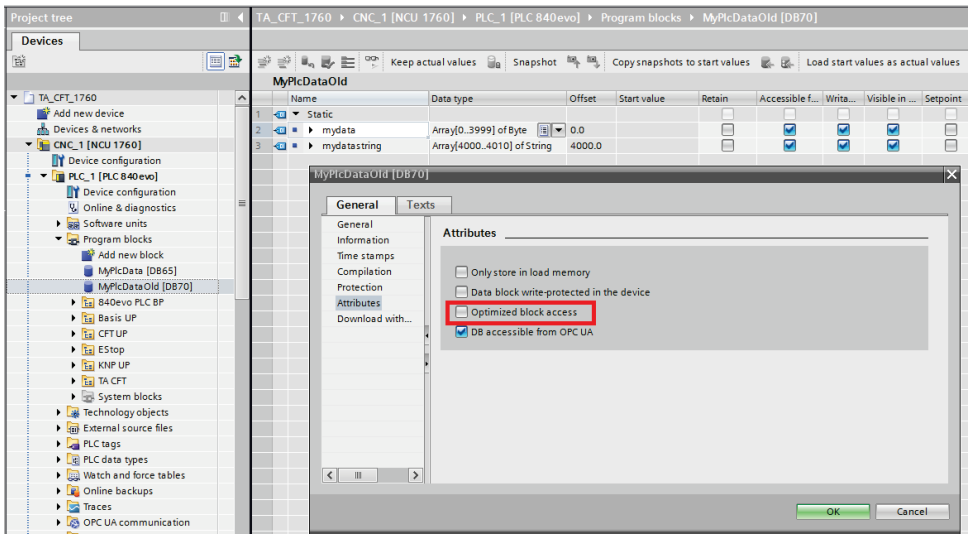


Fig. 3-7: deactivated "Optimized block access"

If this is activated, then only symbolic addressing can be used in DataSvc:

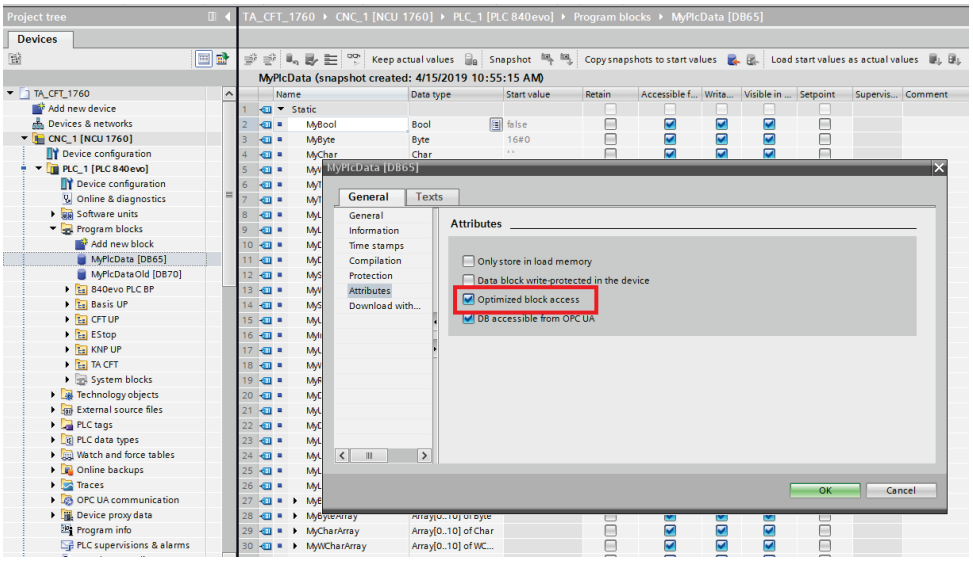


Fig. 3-8: activated "Optimized block access"

For symbolic addressing, the variable path comprises the following:
<NameData block>.<VariableName>

© Siemens AG 2024 All Rights Reserved

Table 3-33: Examples of variable paths (PLC accesses, SINUMERIK ONE)

Variable path	Description
MyPlcData.MyBool	Variable "MyBool" in data block "MyPlcData"
MyPlcData.MyInt	Variable "MyInt" in data block "MyPlcData"
MyPlcData.MyWordArray[4]	5th element of the array variable "MyWordArray" in data block "MyPlcData"

The following data types from the TIA Portal are supported:

- Bool
- Byte
- Char
- WChar
- String
- WString
- Time
- Time_Of_Day
- LTime (read only)
- LTime_Of_Day
- Date
- Date_And_Time
- S5Time
- SInt
- USInt
- Int
- UInt
- DInt
- UDInt
- LInt
- ULInt
- Word
- DWord
- LWord
- Real
- LReal
- LDT

All data types can also be used as array. For arrays, it is only possible to access an individual element.

Variable paths for 1:N constellations

By default, data is accessed on the NCU which is being viewed by Operate. In a 1:N constellation, data can be accessed on a specific NCU (an NCU other than the one displayed in Operate) in two ways:

1. Addition of NCU name and domain (NC/PLC) to the variable path.
   Data can be accessed on a specific NCU by prefixing the actual variable path with the name of the NCU and the domain.
   Examples:
   R 5, from NCU2 and channel 3 - **@NCU2/NC**/Channel/Parameter/R[u3,5]
   MB5 from NCU2 - **@NCU2/PLC**/MB[5]

2. Create DataSvc object with NCU name.
   By specifying the name of an NCU (example: NCU2) when a DataSvc object is created (see DataSvc constructors), access to the specified NCU will always take place via this DataSvc instance. The variable paths do not have to be given prefixes.

Note

NCU names are listed in MMC.INI (see also Fig. 3.7).
Entry:
[GLOBAL]
NcddeMachineNames=*NCU1,NCU2*

Unrestricted © Siemens AG 2024 All Rights Reserved

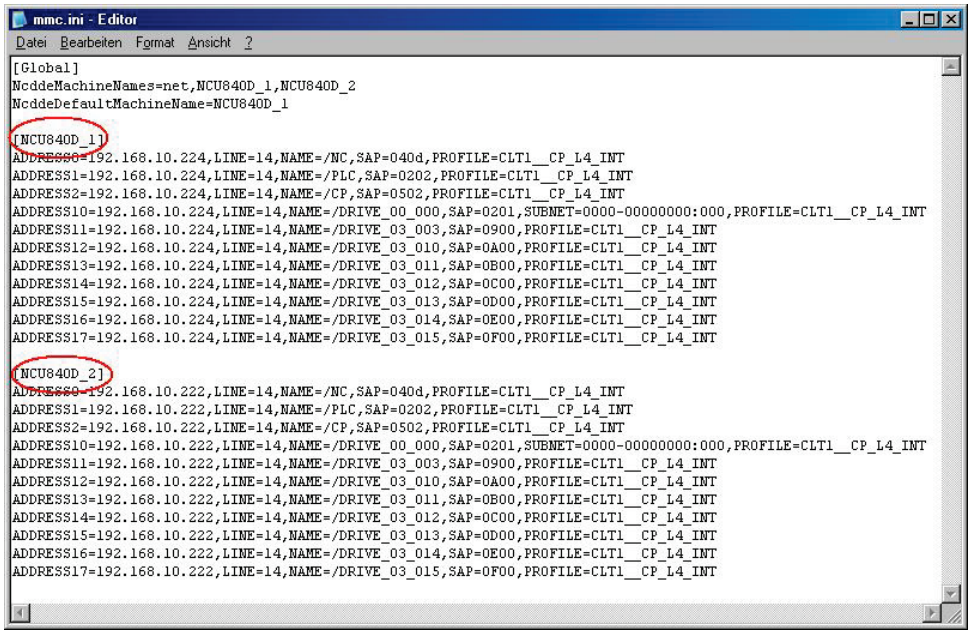


Fig. 3-9: NCU names for 1 : N

Note (GUDs and 1:N)

Only the default machine name is mapped if, for a 1:N constellation, a DataSvc object is created without specifying a server. As a consequence, it is not possible to access the GUDs/machine data of other machine names. This can be resolved by creating a dummy DataSvc object for all of the available machine names in the project:

```
DataSvc m_MyWorkingObject = new DataSvc();
DataSvc m_DummyNCU1 = new DataSvc("NCU840D_1");
DataSvc m_DummyNCU2 = new DataSvc("NCU840D_2");
```

Variable paths CTRL-Energy

The following variable paths are available for accesses to energy management (CTRL-Energy):

Table 3-34: Variable paths CTRL-Energy

Variable	Description	
/Hmi/CtrlEnergy/State	**Format:** **Available:** **Meaning:**	QVariant / UInt Always 1 ➔ Measurement active 0 ➔ Measurement inactive
/Hmi/CtrlEnergy/DeviceNames	**Format:** **Available:** **Meaning:**	QVariant / QVariantList<QString> Always Contains the names (language-independent) of the configured devices as they are listed in the Ctrl-Energy table (total drives, total machine, manual, X1, Y1, …).

© Siemens AG 2024 All Rights Reserved

Variable	Description
/Hmi/CtrlEnergy/Types	**Format:** QVariant / QVariantList<Int> **Available:** Always **Meaning:** Contains, parallel to /Hmi/CtrlEnergy/DeviceNames, an identifier for the type of each configured device: 0..31 ➔ Axis 1000 ➔ Manual 1100 ➔ Total drives 1200 ➔ SENTRON PAC or total machine 1300 ➔ Total machine
/Hmi/CtrlEnergy/Power	**Format:** QVariant / QVariantList<double> **Available:** Always **Meaning:** Parallel to /Hmi/CtrlEnergy/DeviceNames, contains the current power of each configured device.
/Hmi/CtrlEnergy/ActiveEnergy	**Format:** QVariant / QVariantList<double> **Available:** For active measurement **Meaning:** Parallel to /Hmi/CtrlEnergy/DeviceNames, contains the infeed energy of each configured device.
/Hmi/CtrlEnergy/ReactiveEnergy	**Format:** QVariant / QVariantList<double> **Available:** For active measurement **Meaning:** Parallel to /Hmi/CtrlEnergy/DeviceNames, contains the regenerative energy of each configured device.
/Hmi/CtrlEnergy/TotalEnergy	**Format:** QVariant / QVariantList<double> **Available:** For active measurement **Meaning:** Parallel to /Hmi/CtrlEnergy/DeviceNames, contains the total energy (infeed + regenerative energy) of each configured device.

Unrestricted © Siemens AG 2024 All Rights Reserved

The available variable paths in the standard operating area Parameter:

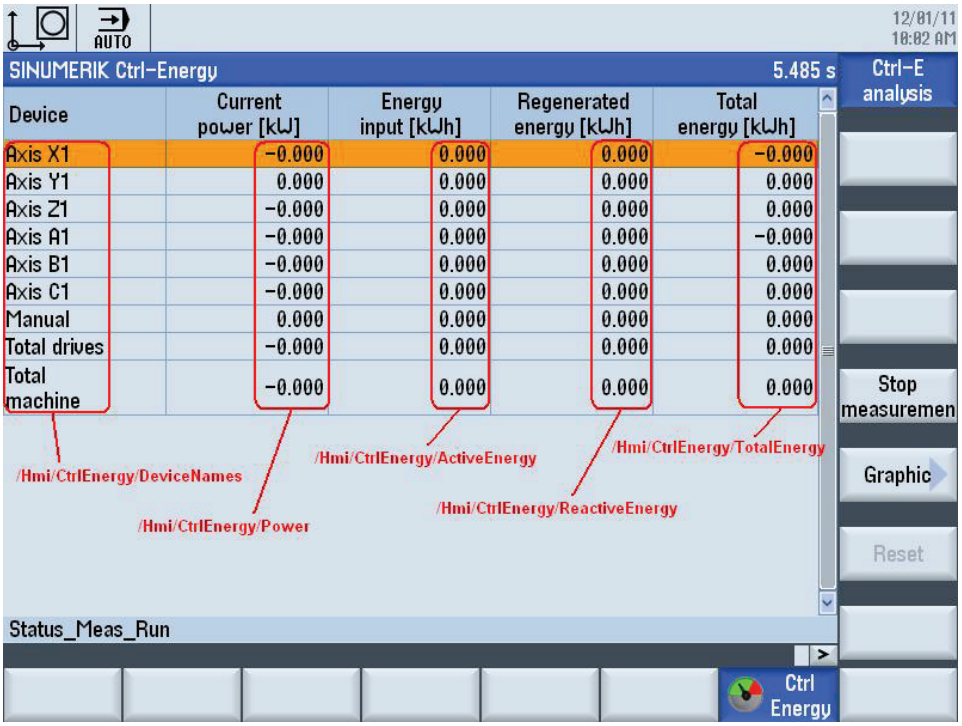


Fig. 3-10: SINUMERIK CTRL-Energy variable paths

In order that the variable paths are active, the configuration file "slctrl_energy.ini" should be created in the directory "<Installation path>/oem/sinumerik/hmi/cfg" with the following content:

```
[Misc]
EnableOAInterface=true
```

Variable path for tool data depending on the cursor position

The data of the tool selected with the cursor in the standard tool management of HMI Operate (orange bar) is available in the following variable:

/Hmi/TMHMICurData

In order that this variable path is active, the configuration file "sltmlistconfig.xml" (milling machine) or "sltmturninglistconfig.xml" (turning machine) should be created in the directory "<Installation path>/oem/sinumerik/hmi/cfg" with the following content:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<CONFIGURATION>
    <SETTINGS>
        <CursorPositionInfoEnabled value="true" type="bool" />
    </SETTINGS>
</CONFIGURATION>
```


© Siemens AG 2024 All Rights Reserved

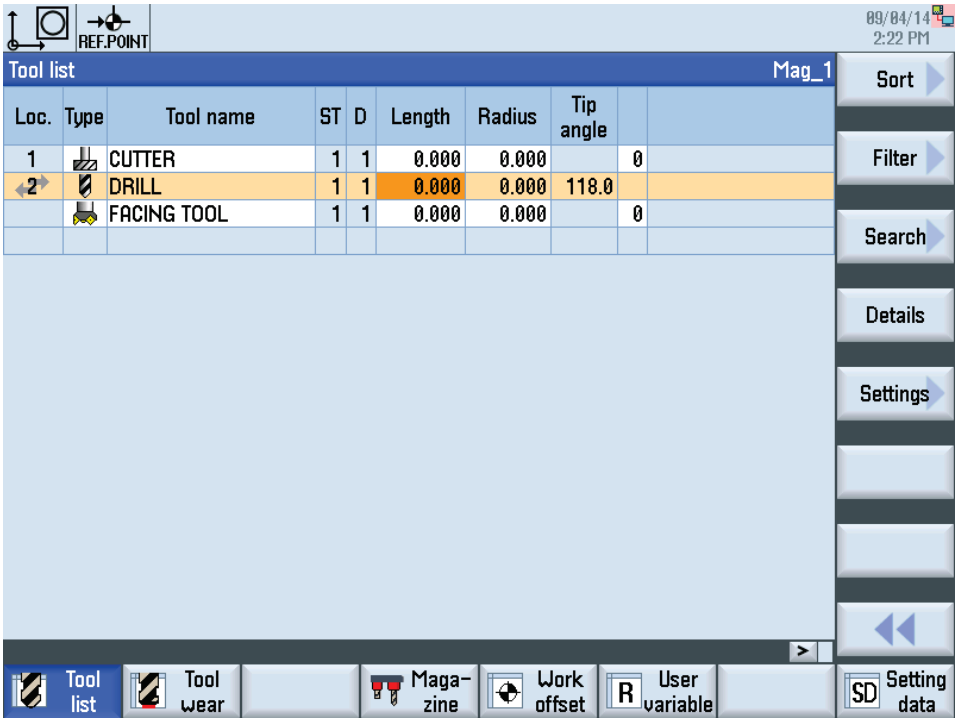


Fig. 3-11: Standard tool management of the "DRILL" tool selected with the cursor

In the variable, the data of the currently selected tool is listed as a string. Access to the variable path can be as read access or as notification (hotlink).

The string has the following format (sample string for above figure):

```
-adaptNo 0 -containerNo 1 -datetime 2014-09-04T14:55:57 -dlNo 0 -edgeNo 1 -
magNo 1 -mtNo 0 -mtPlaceNo 0 -ncuName @ncu1.local -placeNo 2 -toaNo 1 -
toolDuplo 1 -toolIdent DRILL -toolNo 3 -toolType 200
```

Unrestricted © Siemens AG 2024 All Rights Reserved

Table 3-35: Available tool data from "/Hmi/TMHMICurData"

ID	Description
General data	
-datetime	Time stamp
-ncuName	Name of the current NCU
-toaNo	Current TOA number
-containerNo	Current container number
Data to identify a tool	
-toolNo	T no. ("0" if the cursor is at an empty location)
-toolIdent	Tool identifier (empty string if the cursor is at an empty location)
-toolDuplo	Duplo no. of the tool ("0" if the cursor is at an empty location)
-edgeNo	Tool cutting edge number ("0" if the cursor is at an empty location)
-toolType	Tool type ("0" if the cursor is at an empty location)
-dlNo	Tool cutting edge correction location number ("0" if the cursor is at an empty location or no DL corrections have been set up)
Data to identify a location	
-magNo	Magazine number ("0" if the current tool is not on a magazine location)
-placeNo	Magazine location number ("0" if the current tool is not on a magazine location)
-mtNo	Multitool number ("0" if the current tool is not on a multitool location)
-mtPlaceNo	Multitool location number ("0" if the current tool is not on a multitool location)
-adaptNo	Magazine location adapter number ("0" if the current tool is not on a location or an adapter number has not been assigned to the location)

Note

The variable is always updated when one of the list screens of the standard tool management of HMI Operate is exited. This means that the content of the variable is empty after the startup of HMI Operate.

The order in which the tool data is stored in the variables is not specified.

© Siemens AG 2024 All Rights Reserved

3.5.3 Creating an Item object

Several constructors are available to create an item object. If you use the default constructor, the values can be set or overwritten later using the properties.

Table 3-36:Constructors of the item class

public Item.Item() **public Item.Item(string Path)** **public Item.Item(string Path, object Value)**	
Parameter	**Meaning**
Path	Variable path
Value	For a write request, the value to be written can be set in the constructor of the item class.
Possible exceptions	ArgumentException, DataSvcException

Example: See Chapter 3.3.2

3.5.4 Properties

Path

Variable path of the required data item. The path is saved as string. The path formatting can be seen in the "Variable paths" chapter.

Table 3-37:Path

public string Item.Path { set; get; }

SymbolicName

Presently not used.

Value

Value of the variable. The type of these properties is System.Object, and can have different contents depending on the specified variable (number, text, exception). For an array access, the Value property contains an array of item objects.

Table 3-38:Value

public object Item.Value { set; get; }

Unrestricted © Siemens AG 2024 All Rights Reserved

3.6 Reference DataSvcStatus

3.6.1 Definitions

Overview

Information that is evaluated infrequently when reading, writing, or for notifications is combined in the class described below.
The class is defined in the "Siemens.Sinumerik.Operate.Services" namespace.

3.6.2 Properties

Ok

For a running subscription, indicates whether an error has occurred (e.g. connection abort).

Table 3-39:Ok

public bool DataSvcStatus.Ok { get; }

Values:
True: No problems
False: A problem has occurred. A more precise error evaluation can be made using the "ErrorNo" property.

ErrorNo

Error number in the case of a problem with a running subscription.

Table 3-40:ErrorNo

public int DataSvcStatus.ErrorNo { set; get; }

Quality

Presently not used.

SequenceNumber

The sequence number can be used to differentiate whether a request was performed earlier or later. The sequence numbers refer to all activities of a DataService. This means that the sequence numbers of a request are steadily increasing (monotone) but are not without gaps.

Table 3-41:SequenceNumber

public int DataSvcStatus.SequenceNumber { set; get; }

ServerInfo

Additional information attached by DataService. Only relevant for reading out and setting the PLC clock.

Table 3-42:ServerInfo

public object DataSvcStatus.ServerInfo { set; get; }

© Siemens AG 2024 All Rights Reserved

TimeStamp

Time at which the read data was last received in the DataService.

Table 3-43:TimeStamp

public System.DateTime DataSvcStatus.TimeStamp { set; get; }

UpdateRate

Presently not used.

Table 3-44:UpdateRate

public ulong DataSvcStatus.UpdateRate { set; get; }

Unrestricted © Siemens AG 2024 All Rights Reserved

3.7 FAQs

3.7.1 General FAQs

When does an ArgumentException occur for a DataSvc object?

Occurs, for example, when:
- Item.path is empty
- Item.Value is empty for a write call.
-

What value does the ErrorNo property of a DataSvcStatus object have when the Ok property is true?

ErrorNo = 0

How can I check whether there is a connection to the NC?

In the mean time, the PCU50/IPC may ramp up faster than the NCU. A .NET application can check whether communication is possible as follows:

1) By monitoring variables "/nc/connect_state" or "/plc/connect_state". The return value 30 indicates that a connection exists.

2) By setting up a notification for changing the connection state (SubscribeConnectionState) with the infrastructure service.

3.7.2 FAQs regarding hotlink

Do I need a DataSvc object for each hotlink?

Yes. A separate DataSvc object is required for each hotlink because a DataSvc object can only execute precisely one request in parallel.

When should I deactivate or cancel hotlinks?

Whenever a hotlink is not needed, it should be deactivated ("Unsubscribe"). A hotlink on a form that is not visible is an unnecessary load on communication resources.

With an update rate of 200 ms, why are value changes more frequently than every 200 ms?

The update rate specifies a maximum time frame in which potential value changes should be returned. If the value changes within the 200 ms, the client application is informed of this value change more than once within 200 ms. An artificial brake is not built in.
Values are then supplied less frequently (e.g. only every 200 ms) if the communication load in the server component is increased by your own or a further client application.

© Siemens AG 2024 All Rights Reserved

3.7.3 FAQs regarding machine data / GUDs

Do I have to communicate (map) GUDs and machine data?

No. Mapping is performed once automatically during startup of the application. In this case, all data is mapped that can be accessed (channel, axis and general machine data, all GUD areas, channel, axis and general setting data). The access paths of these variables are described in the "Variable paths for GUDs and machine data" chapter.

How do I access two-dimensional GUDs?

GUD variables, which are defined in two dimensions, must first be converted to a one-dimensional array. This means that the variable path must be calculated. To do this, the dimensions of the array must be known.

Example:
Definition of GUD: DEF NCK REAL _MYVAR[99,11]
We intend to read variable _MYVAR[2,3].
The index of the access path is calculated as follows: 2*11 + 3 + 1 => 26
It is assumed that this variable is an SGUD, resulting in the following access path:
/NC/_N_CH_GD1_ACX/_MYVAR [26]

The values in the delegate are not being displayed, although the values have been written to the corresponding control correctly. What is wrong?

Because the delegate is called from another thread, not all display elements can be accessed within the implemented function.
More information is available at Microsoft using the keyword: "How to: Make Thread-Safe Calls to Windows Forms Controls" (http://msdn.microsoft.com/en-us/library/ms171728(v=VS.90).aspx)

Unrestricted © Siemens AG 2024 All Rights Reserved

© Siemens AG 2024 All Rights Reserved

4

4 Program instance services (PI services)

Objective of the chapter

This chapter describes the interface of the PiService. This is used for communication between applications and the NC/PLC, specifically to issue program instance services (PI services).

Note

A PI service call returns immediately. However, it is not guaranteed that the PI service has been completely executed when it returns. Some PI services return their status via OPI variables.

Unrestricted © Siemens AG 2024 All Rights Reserved

4.1 Introduction

4.1.1 Class model

Overview

The class model of the PiService consists primarily of the following class:

- PiSvc

Namespace

All classes are defined in the "Siemens.Sinumerik.Operate.Services" namespace. As a consequence, this namespace can be integrated into the C# project at the start of the particular file:

```
using Siemens.Sinumerik.Operate.Services;
```

Class PiSvc

This class has functions to execute various types of PI services.

© Siemens AG 2024 All Rights Reserved

4.2 Step-by-step examples

Overview

The following chapters show the various applications for the PiService step-by-step. For every "step-by-step example," there is an executable example in SINUMERIK Integrate Create MyHMI /3GL.
These and all other examples are shown in the following overview of examples.

Table 4-1: Overview of the examples

Application	Example	Chapter
Sending a general PI service	PiSvcPiCommand	4.2.2

Only the items relevant for the PiService are explained or described in each example. The comments for the sources contain more detailed information (e.g. screen layout, outputs, etc.).

4.2.1 Preparation

Overview

Preparatory measures are described in Chapter 2.1 ➔ "New Project". All of the items described there must be executed first.

4.2.2 Sending a general PI service

Overview

The following example shows how a general PI service is sent. The required parameters can be entered at the interface. The PI service is then issued using the "run command" button. In this example, part program "MYMPF.MPF" in channel 1 (201) is selected for processing (_N_SELECT). To enable the example to be executed, a part program with the name "MYMPF.MPF" must be created in the NC.

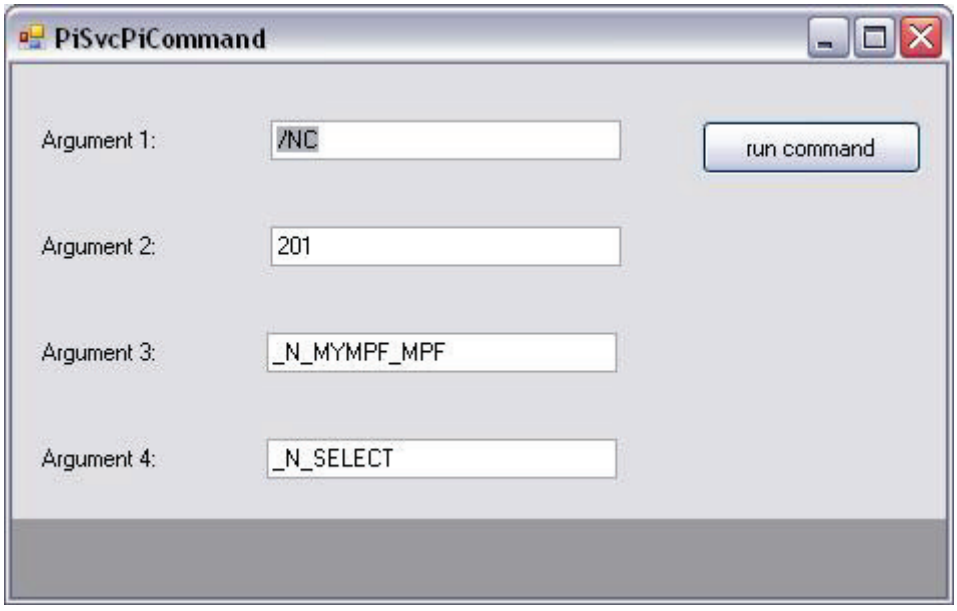


Fig. 4-1: Example, PiSvcPiCommand

Unrestricted © Siemens AG 2024 All Rights Reserved

Step 1

Prepare the arguments. To start, an array of four strings is created and populated. This array includes the parameter assignment for the PI service.

```
// create string array for the pi-arguments
System.String[] piArgs = new System.String[4];

// fill the array with the pi-arguments
piArgs[0] = "/NC";
piArgs[1] = "201";
piArgs[2] = "_N_MYMPF_MPF";
piArgs[3] = "_N_SELECT";
```

Step 2

Create the PiSvc object. When you create the object, the array of the arguments for the PI service is passed in the constructor.

```
// create an instance of the PiSvc
PiSvc PiService = new PiSvc(piArgs);
```

Step 3

Send the PI command. Execution of the PI service is initiated using the Start() function.

```
// start pi service
PiService.Start();
```


© Siemens AG 2024 All Rights Reserved

4.3 Reference PiSvc

4.3.1 Definitions

Overview

The PiSvc object is used to issue program instance services (PI services). General and binary PI services can be issued.

All calls are implemented as synchronous accesses. Synchronous calls block the calling item until the request has been completed.

The class is defined in the "Siemens.Sinumerik.Operate.Services" namespace.

General PI services

General PI services involve requests to the NC. Parameters for the particular service are obtained in the PI help. This is also part of the documentation of the SINUMERIK Operate .NET.

Binary PI services

Binary PI services involve requests to the PLC.

4.3.2 Creating a PiSvc object

Several constructors are available to create a PiSvc object. If you use the default constructor, the values can be set or overwritten later using the properties.

Table 4-2: Constructors of the PiSvc class

public PiSvc.PiSvc() **public PiSvc.PiSvc(String[] PiArgs)** **public PiSvc.PiSvc(String PiStopCommand)** **public PiSvc.PiSvc(String PiBinaryCommand, Byte[] PiBinaryBytes)**	
Parameter	**Meaning**
PiArgs	Array of parameters required to execute a general PI service in the NCK.
PiStopCommand	Command string to execute a general stop PI service in the NCK.
PiBinaryCommand	Command string to execute a binary PI service in the PLC.
PiBynaryBytes	Array, comprising bytes. The bytes contain the parameters to execute a binary PI service in the PLC.
Possible exceptions	ArgumentException, PiSvcException

Example: See Chapter 4.2.2

Unrestricted © Siemens AG 2024 All Rights Reserved

4.3.3 Issuing a PI service

A PI service is issued either using the Start() function or the Stop() function.
There are several types of PI services:

- General PI service
- Binary PI service

The following shows with which function, properties and constructors the corresponding PI service can be initiated.

General PI service

Constructors:

- public PiSvc.PiSvc(string[] PiArgs)
- public PiSvc.PiSvc(string PiStopCommand)

Properties:

- PiArgs
- PiStopCommand

Function to issue:

- Start()
- Stop()

Binary PI service

Constructor:

- public PiSvc.PiSvc(string PiBinaryCommand, byte[] PiBinaryBytes)

Properties:

- PiBinaryCommand
- PiBinaryBytes

Function to issue:

- Start()

Functions (Start/Stop)

Table 4-3: The start function

public void PiSvc.Start()	
Parameter	**Meaning**
Possible exceptions	ArgumentException, PiSvcException

Example: See Chapter 4.2.2

Note

Depending on what property was last set, or what constructor was used to create the PiSvc object, a general or a binary PI service is executed.

Table 4-4: The stop function

public void PiSvc.Start()	
Parameter	**Meaning**
Possible exceptions	ArgumentException, PiSvcException

© Siemens AG 2024 All Rights Reserved

4.3.4 Properties of the PiSvc class

PiArgs

Array of parameters required to execute a general PI service in the NCK.

Table 4-5:PiArgs

public string[] PiSvc.PiArgs { set; get; }

PiStopCommand

Command string to execute a general stop PI service in the NCK.

Table 4-6:PiStopCommand

public string PiSvc.PiStopCommand { set; get; }

PiBinaryBytes

Array, comprising bytes. The bytes contain the parameters to execute a binary PI service in the PLC.

Table 4-7:PiBinaryBytes

public byte[] PiSvc.PiBinaryBytes { set; get; }

PiBinaryCommand

Command string to execute a general stop PI service in the NCK.

Table 4-8:PiBinaryCommand

public string PiSvc.PiBinaryCommand { set; get; }

PriorityFlag

Processing priority of PiService accesses.

Table 4-9:PriorityFlag

public SlPriority PiSvc.PriorityFlag { set; get; }

The PiService uses three processing priorities:

`SlPriority.lowPriority`
Low processing priority. This value should be used for cyclic activities and activities running in the background. By default, the PiService uses this priority for advise requests as soon as the first value arrives.

`SlPriority.highPriority`
High processing priority. This value is reserved for exclusive use by the machine manufacturer to achieve productivity goals (e.g. tool change).

`SlPriority.defaultPriority`
Default processing priority. This flag is intended for user interaction and screen rebuilding. By default, the PiService executes all read and write requests in this priority. This priority is also used when accessing the first value for Subscribe requests.

Unrestricted © Siemens AG 2024 All Rights Reserved

Please note the following constraints:

1. As requests with high processing priority prevent requests with low processing priority from being processed, only the priority `lowPriority` may be used for cyclic requests (hotlinks).
2. The processing priorities are active only within the PiService. Low-priority requests that have already started will no longer be overtaken by higher-priority requests.

Note

The flag `defaultPriority` (value = 0) does not have to be explicitly set. This value is automatically set after creating a PiSvc object.

Timeout

Time limit for the call duration, in milliseconds. After this time expires, the call returns with a PiSvcException.

Table 4-10:Timeout

public uint PiSvc.Timeout { set; get; }

Pre-defined constants:

`SlTimeout.standardTimeout`
Default value. This timeout value should be used for fast accesses. It takes into account potential communication delays, connection establishment, etc. (20000 milliseconds).

`SlTimeout.noTimeout`
No time monitoring.

PI services in 1:N constellations

By default, PI execution takes place on the NCU Operate is viewing. In a 1:N constellation, execution can be performed on a specific NCU (an NCU other than the one displayed in Operate) in two ways:

1. Addition of the first argument ("/NC") to the NCU name.
   A PI service can be triggered in a specific NCU by prefixing the first parameter with the NCU name.
   Examples:
   Trigger PI in NCU2 - **@NCU2/**NC

2. Create PISvc object with NCU name.
   By specifying the name of an NCU (example: NCU2) when a PISvc object is created (see PISvc constructors), access to the specified NCU will always take place via this PISvc instance. The arguments for the PI service do not have to be given prefixes.

Note

NCU names are listed in the MMC.INI file.
Entry:
[GLOBAL]
NcddeMachineNames=*NCU1,NCU2*

© Siemens AG 2024 All Rights Reserved

5 Access to alarms and events

5

Objective of the chapter

This chapter describes the interface of the AlarmService. It provides functions for handling alarms and messages from the PLC, the NCK (including drive alarms), and the HMI. It supplies all alarms and messages of the HMI as well as an NCU connected to the HMI (NCK and PLC). Specific alarms and messages include:

- HMI alarms
- NCK alarms including drive alarms (provided they are forwarded from the NCK)
- Part program messages from the NCK
- Diagnostic buffer alarms and PLC messages (FC10)
- Alarm_S(Q) alarms (SFC17/18, PDiag, HiGraph, S7-Graph) with results of criteria analysis

The AlarmService also provides an interface for setting, acknowledging and deleting alarms.

All applications wishing to acquire alarms and events, for example, for display purposes or for further processing and forwarding to MES, are referred to as clients in the following.

Note

The communication time is not guaranteed. Real-time tasks therefore cannot be handled with SINUMERIK Operate .NET.

Unrestricted © Siemens AG 2024 All Rights Reserved

5.1 Overview

5.1.1 Introduction and explanation of terms

Alarms and events

An alarm designates an **interval** during which an exceptional or critical state exists in a process (NCK/PLC/HMI). In contrast, an event designates a certain **point in time** at which an alarm occurs, disappears, or is acknowledged by the user (coming, going or acknowledgment event). Therefore, an alarm can be in various states depending on the events and an event is a transition from one state to another. The subsequent Fig. 5-1 clearly shows the interrelationship.

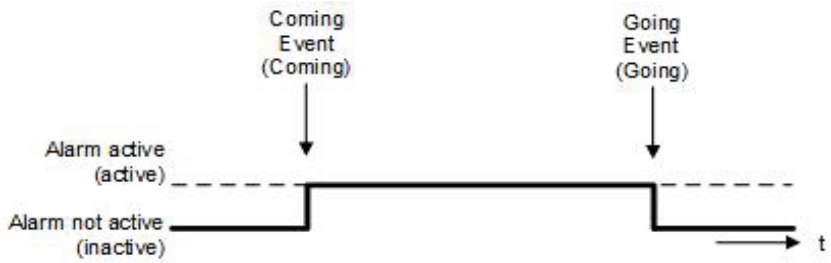


Fig. 5-1: Alarms and events

Client, service and source

The alarm service provides functions for implementing an alarm client.
An alarm client is generally an application that is used for monitoring alarms along with their events, parameters, time stamps, etc. The alarm client always communicates with the alarm service. The AlarmService on its part communicates with the various alarm sources (alarm sources) in the system, i.e. HMI, PLC, and NCK alarms (including drive alarms).
The following subchapters describe the signal flow between client, service, and source as a function of the various alarm types.

Alarm source

The alarm service supplies alarms from various alarm sources of an NC. The alarm service makes a distinction between the following standard sources:

Table 5-1: Specified alarm source identifiers (source URL) of the alarm service

Source	Source URL
HMI	/HMI
NCK	/NCK
NCK channels (only for part program messages)	/NCK/Channel#<Channel No>/Partprogram
PLC diagnostic buffer	/PLC/DiagBuffer
PLC PMC alarms (Alarm_S(Q): PDiag, S7-Graph, and HiGraph)	/PLC/PMC

© Siemens AG 2024 All Rights Reserved

Alarm attributes

With every alarm, the client is provided not only with the alarm number identifying the alarm but also with additional data such as the alarm state, a time stamp, or associated process values. This data is referred to as attributes. There are standard attributes and additional attributes. The standard attributes are identical for all alarm categories (see alarm category), while the additional attributes can vary from category to category.
In addition, the attributes differ according to their source: They can originate directly from the alarm source (e.g. time stamp, associated process values) or, alternately, they can come from a configuration (e.g. priority).
The alarm service provides a fixed number of attributes via the alarm object.

Alarm category

Not all alarms support the same set of attributes: For example, part program messages have no associated process values. To be able to differentiate alarms with regard to their attributes, they are assigned to different categories. All alarms of one category support the same set of attributes.

HMI alarms

HMI alarms are alarms that originate in the HMI itself. In this case, the HMI applications are the alarm source. HMI alarms can also be generated via SINUMERIK Operate .NET.

Synchronous calls

Synchronous calls return only after the request has been completed, i.e. the calling thread is blocked until this is the case. This can interfere with event processing since control inputs and displays are withheld during a synchronous call. For this reason, calls that may take a long time should be performed asynchronously.

Table 6-2: Synchronous calls

AlarmSvc call	Description
Subscribe	Setting up a notification when the alarm list changes.

5.1.2 Class model

Overview

The class model of the alarm service consists primarily of the following classes:

- AlarmSvc
- Alarm
- AlarmSource

Namespace

All classes are defined in the "Siemens.Sinumerik.Operate.Services" namespace. As a consequence, this namespace can be integrated into the C# project at the start of the particular file:

```
using Siemens.Sinumerik.Operate.Services;
```

Unrestricted © Siemens AG 2024 All Rights Reserved

Class AlarmSvc

The AlarmSvc class provides functions to monitor alarms and events. Further, via this class, the existing alarm sources can be read out and HMI alarms can be generated.

Class Alarm

This class represents an individual alarm. The class contains only the properties that reflect the various attributes of an alarm.

Class AlarmSource

The AlarmSource represents an Alarm source. The class comprises only the two properties required to define an alarm source.

© Siemens AG 2024 All Rights Reserved

5.2 Step-by-step examples

Overview

The following chapters show the various applications of the alarm service step-by-step. An executable example is included in the SINUMERIK Integrate Create MyHMI /3GL for each "step-by-step example".
These and all other examples are shown in the following overview of examples.

Table 5-3: Overview of the examples

Application	Example	Chapter
Accessing the alarm list. Displaying the currently pending alarms.	AlarmSvcAlarmList	5.2.2
Access to the alarm events. Display of the event list and all arriving individual events.	AlarmSvcAlarmEvents	5.2.3
Creating and resetting HMI alarms.	AlarmSvcSetResetAlarm	5.2.4

Only the items relevant for the AlarmService are explained or described in each example. The comments for the sources contain more detailed information (e.g. screen layout, outputs, etc.).

5.2.1 Preparation

Overview

Preparatory measures are described in Chapter 2.1 ➔ "New Project". All of the items described there must be executed first.

5.2.2 Accessing the alarm list

Overview

In this example, when loading, the form of a notification is activated via the change of the alarm list. All currently pending alarms are displayed in the list box. If the alarm list changes, then it is output again with current data.

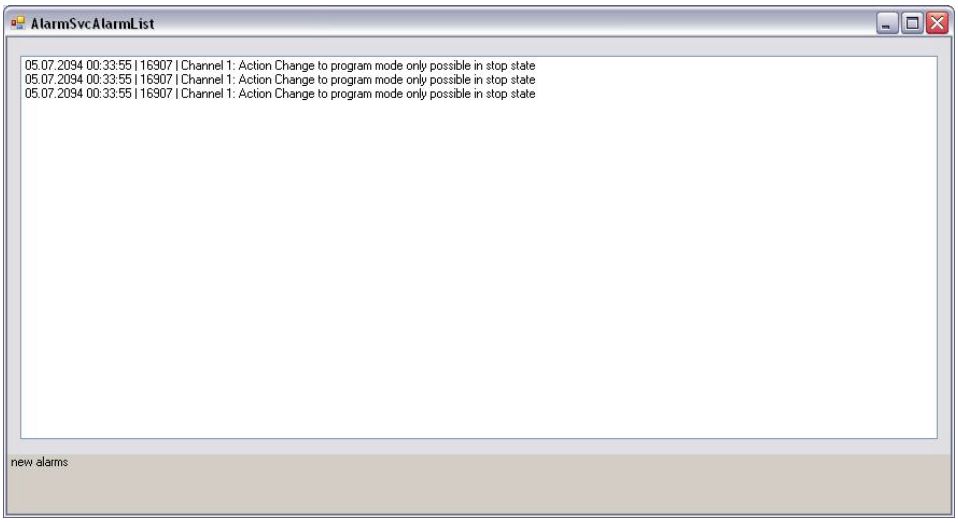


Fig. 5-2: Accessing the alarm list

Unrestricted © Siemens AG 2024 All Rights Reserved

Step 1

Create the required member variables (one object, type AlarmSvc and one variable, type Guid):

```
AlarmSvc m_AlarmService = null;
System.Guid m_AlarmGuid;
```

Step 2

Create the AlarmSvc object in the frmMain() function.
In this case, AlarmSvc is notified that all alarm texts should be in English.

```
m_AlarmService = new AlarmSvc("eng");
```

Step 3

Create a subscription. Here, the implemented function of the delegate is transferred, in which the change of the alarm list should be communicated. A Guid is obtained as return.

```
m_AlarmGuid = m_AlarmService.Subscribe(AlarmList);
```

Step 4

Implement the delegate.

Important

Because the delegate is called from another thread, not all display elements can be accessed. Consequently, it is desirable to call a second delegate with the same signature with which changes can be performed on the user interface.

In the implemented delegate function, a change is sent to the same delegate signature, now in the GUI thread. Here, the alarms that are currently pending are now displayed. All alarms are exported and displayed consecutively using a "foreach loop".

```
public void AlarmList(System.Guid guid, Alarm[] alarms)
{
    // are we in GUI thread (main) ?
    // No we are not in GUI thread => send message to GUI thread
    if (lstAlarms.InvokeRequired)
    {
        AlarmListChanged alarmDelegate = new AlarmListChanged(AlarmList);
        BeginInvoke(alarmDelegate, new Object[] { guid, alarms });
    }
    else
    {
        //check if guid is correct
        if (m_AlarmGuid.Equals(guid))
        {
            // clear the listbox
            lstAlarms.Items.Clear();
            setStatus("new alarms");

            // go through the array for each alarm
            foreach(Alarm element in alarms)
            {
                // write the timeStamp, alarmnumber and the alarmmessage to
                // the listbox
                lstAlarms.Items.Add(element.TimeStamp.ToString() + " | " +
                                    element.Id.ToString() + " | " +
                                    element.Message);
            }
        }
    }
}
```


© Siemens AG 2024 All Rights Reserved

Step 5

Unsubscribe the subscription.

```
m_AlarmService.UnSubscribe(AlarmList);
```

5.2.3 Access to the alarm events

Overview

In this example, the alarm-events list and the individual alarm events are listed in the left-hand and right-hand list box, respectively. Clicking the "subscribe eventlist" button subscribes to the changes in the alarm-events list. Clicking the "unsubscribe eventlist" button ends this subscription. Clicking the "subscribe events" button subscribes to new individual alarm events. Clicking the "unsubscribe events" button ends this subscription.

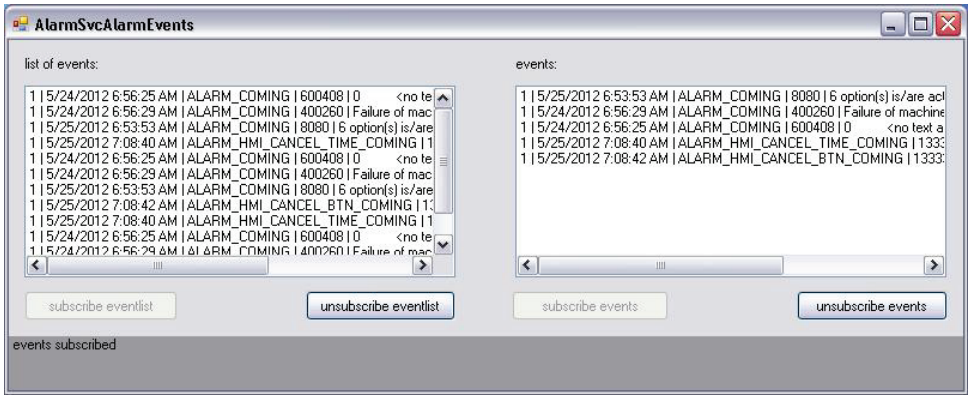


Fig. 5-3: Access to alarm events

Step 1

Create the required member variables of the AlarmSvc and Guid type.

```
AlarmSvc m_AlarmSvc = null;
System.Guid m_EventListGuid;
System.Guid m_EventGuid;
```

Step 2

Create the AlarmSvc object. The "eng" transfer value specifies that all alarm texts should be transferred in English.

```
m_AlarmSvc = new AlarmSvc("eng");
```

Step 3

Create a subscription to the changes of the list with alarm events. In this case, the implemented delegate to be informed when the event list changes is transferred. A Guid is obtained as return.

```
m_EventListGuid = m_AlarmSvc.SubscribeEventList(CbEventListChanged);
```

Unrestricted © Siemens AG 2024 All Rights Reserved

Step 4

Implement the CbEventListChanged() delegate.

Important

Because the delegate is called from another thread, not all display elements can be accessed. Consequently, it is desirable to call a second delegate with the same signature with which changes can be performed on the user interface.

In the implemented delegate function, a change is sent to the same delegate signature, now in the GUI thread. This outputs the events from the current alarm-event list to a list box. All events are exported and displayed consecutively using a "foreach loop".

```
private void CbEventListChanged(System.Guid guid, Alarm[] events)
{
    // are we in GUI thread (main) ?
    // No we are not in GUI thread => send message to GUI thread
    if (lstEventsList.InvokeRequired)
    {
        AlarmEventListChanged eventlistDelegate = new
                                AlarmEventListChanged(CbEventListChanged);
        BeginInvoke(eventlistDelegate, new Object[] { guid, events });
    }
    else
    {
        foreach (Alarm alarm in events)
        {
            lstEventsList.Items.Add(alarm.Instance + " | " +
                      alarm.TimeStamp + " | " + alarm.State + " | " +
                      alarm.Id + " | " + alarm.Message);
        }
    }
}
```

Step 5

Unsubscribe the subscription to the changes of the list with alarm events.

```
m_AlarmSvc.UnSubscribeEventList(CbEventListChanged);
```

Step 6

Create a subscription to newly occurred alarm events. The implemented delegate to be informed of the occurrence of new alarm events is transferred. A Guid is obtained as return.

```
m_EventGuid = m_AlarmSvc.SubscribeEvents(CbNewEvent);
```

Step 7

Implement the CbNewEvent() delegate.

Important

Because the delegate is called from another thread, not all display elements can be accessed. Consequently, it is desirable to call a second delegate with the same signature with which changes can be performed on the user interface.

In the implemented delegate function, a change is sent to the same delegate signature, now in the GUI thread. This outputs the new events to a list box.

© Siemens AG 2024 All Rights Reserved

```
private void CbNewEvent(Guid guid, Alarm[] events)
{
    // are we in GUI thread (main) ?
    // No we are not in GUI thread => send message to GUI thread
    if (lstEvents.InvokeRequired)
    {
        NewAlarmEvents eventDelegate = new NewAlarmEvents(CbNewEvent);
        BeginInvoke(eventDelegate, new Object[] { guid, events });
    }
    else
    {
        foreach (Alarm alarm in events)
        {
            lstEvents.Items.Add(alarm.Instance + " | " +
                                alarm.TimeStamp + " | " +
                                alarm.State + " | " +
                                alarm.Id + " | " + alarm.Message);
        }
    }
}
```

Step 8

Unsubscribe the subscription for notification on the arrival of new alarm events.

```
m_AlarmSvc.UnSubscribeEvents(CbNewEvent);
```

5.2.4 Creating and deleting HMI alarms

Overview

With this example, HMI alarms can be triggered, acknowledged and reset.
The entries in the "alarm ID" selection box and the "alarm instance" input field specify the alarm to be acknowledged or reset using the "acknowledge alarm", "reset alarm", or "reset warning" buttons. The numbers of the alarms that can be created with this example are entered in the selection box.

To execute the example, you must make the following four changes to the configuration of the Alarm server:

1. Define your own alarm source
2. Expand the database of the Alarm & Event server
3. Define your own alarm texts
4. Expand the Alarm & Event server to include your own alarm texts

Defining your own alarm source

In the language-neutral database, a new MSGTEXT attribute must be added for each OEM alarm in order to reference the country-specific size. As no changes can be made in the cfg subdirectory of the siemens directory (slaedatabase.hmi) (this database might be overwritten in the event of an update), a database named after a specific OEM is created containing only new MSGTEXT attributes and saved in the cfg subdirectory in the oem directory. The database is always stored in binary format on the target hardware.
The HMI file (binary format) is generated from the XML file with the slHmiConverterGui tool. Conversion in the other direction is also possible.

Unrestricted © Siemens AG 2024 All Rights Reserved

The configuration of the samplesource_db.hmi file for the example looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<SlAeAlarmAttributes Version="01.00.00.00">
  <Types>
    <Type TypeName="Condition" TypeID="1">
      <Category Version="1.0" CatID="1">
        <CatDescr>Alarms of a DOTNET example .</CatDescr>
      </Category>
    </Type>
  </Types>
  <Sources>
    <Source SourceID="100001" SourceURL="/HMI/SAMPLESOURCE" CatLink="1">
      <Alarms DISPLOC="0">
        <Alarm AlarmID="133333">
          <MSGTEXT>samplesource|133333/HMI/SAMPLESOURCE</MSGTEXT>
        </Alarm>
        <Alarm AlarmID="133334">
          <MSGTEXT>samplesource|133334/HMI/SAMPLESOURCE</MSGTEXT>
        </Alarm>
        <Alarm AlarmID="133335">
          <MSGTEXT>samplesource|133335/HMI/SAMPLESOURCE</MSGTEXT>
        </Alarm>
        <Alarm AlarmID="133336">
          <MSGTEXT>samplesource|133336/HMI/SAMPLESOURCE</MSGTEXT>
        </Alarm>
        <Alarm AlarmID="133337">
          <MSGTEXT>samplesource|133337/HMI/SAMPLESOURCE</MSGTEXT>
        </Alarm>
      </Alarms>
    </Source>
  </Sources>
</SlAeAlarmAttributes>
```

Explanations:

An alarm source is defined with the alarm source identifier ("SourceURL") = "/HMI/SAMPLESOURCE", the source ID ("SourceID") = "100001", and the category ID ("CatLink") = "1". This alarm source contains the alarms with alarm number ("AlarmID") 133333 to 133337, whose language-specific texts are stored in the "samplesource_alarmtext_<Language code>.qm" text files.

Expanding the database of the Alarm & Event server

The OEM-specific database is declared to the Alarm & Event service by an entry in its settings files (slaesvcconf.xml/.cfg). In this case, as well, the entry is not made in the file in the Siemens directory but in a copy created in the corresponding subdirectory (cmg) in the oem directory which only contains the additional entry for the new database. The files that together form the database of the Alarm & Event service are indicated under "CONFIGURATION/DataBases". The database contains the definition of the alarm attributes and the listing of the attribute values independent of the language. The database of the OEM can be expanded by specifying additional files.
The expansion of the database in the slaesvcconf.xml settings of the Alarm & Event service is saved on the target in binary format. The CFG file (binary format) is generated from the XML file with the slHmiConverterGui tool. In this case, as well, the conversion can also be performed in the reverse direction (from binary to XML format).

© Siemens AG 2024 All Rights Reserved

Expansion of the database to include your own database, e.g. "slaesvcconf.cfg":

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- Configuration of the Solutionline Alarm & Event Service -->
<CONFIGURATION>
   <DataBases>
           <DataBase_02 type="QString" value="samplesource_db.hmi"/>
   </DataBases>
</CONFIGURATION>
```

Defining your own alarm texts

The country-specific alarm texts must be made available as SINUMERIK Operate text files in
TS format. The text files converted to binary format QM with slHmiConverterGui are then stored in the lng subdirectory of the oem directory. Alternatively, it is also sufficient if the TS files are saved to the lng directory. These are then automatically converted into the binary QM format the next time SINUMERIK Operate boots. The Alarm & Event service uses the "MSGTEXT" attribute from the database to locate the alarm text associated with an alarm.

The QM file samplesource_alarmtext_eng.qm with language-specific texts:

```
<!DOCTYPE TS>
<TS>
   <context>
     <name>samplesource</name>
     <message>
        <source>133333/HMI/SAMPLESOURCE</source>
        <translation>This is a timeout alarm.</translation>
     </message>
     <message>
        <source>133334/HMI/SAMPLESOURCE</source>
        <translation>This is a cancel alarm.</translation>
     </message>
     <message>
        <source>133335/HMI/SAMPLESOURCE</source>
        <translation>This is an HMI alarm.</translation>
     </message>
     <message>
        <source>133336/HMI/SAMPLESOURCE</source>
        <translation>This is an acknowledge alarm.</translation>
     </message>
     <message>
        <source>133337/HMI/SAMPLESOURCE</source>
        <translation>This is a warning.</translation>
     </message>
   </context>
</TS>
```

Expanding the Alarm & Event server to include your own alarm texts

The country-specific OEM alarm text files are declared to the Alarm & Event service by means of an entry in its settings files (slaesvcadapconf.xml/.cfg). In this case, as well, the entry is not made directly in the file located in the siemens directory, but rather in a copy created in the oem directory. The copied file contains only the additional entry for the new alarm texts. When creating your own alarm texts, the configuration file of the Alarm & Event service slaesvcadapconf.xml must be expanded. A list of base names is specified under "CONFIGURATION/AlarmTexts/BaseNames" (currently just one entry, "BaseName_01"). These entries are used to define which alarm text files are loaded when starting up. The base name corresponds to the name of the alarm text file (e.g. alarm text file for English: "samplesource_alarmtext_eng.qm") without appended language identifier (e.g. "samplesource_alarmtext"). The OEM can add

Unrestricted © Siemens AG 2024 All Rights Reserved

entries to this list. It should be noted that OEM-specific base names are used. The corresponding OEM alarm text files must be saved in the lng directory of the oem or user directory.

Addition of your own texts to alarm texts with the "slaesvcadapconf.cfg" file:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- Configuration of the Solutionline Alarm & Event Service Adapter -->
<CONFIGURATION>
   <AlarmTexts>
          <BaseNames>
                 <BaseName_02 type="QString" value=
                                "samplesource_alarmtext"/>
          </BaseNames>
   </AlarmTexts>
</CONFIGURATION>
```

Note

The storage of the previously described files is explained in the following example in Step 11.

For more information about user alarm texts, refer to the "SINUMERIK Operate Commissioning Manual", starting with Chapter 5.4 "Configuring User Alarm Texts".

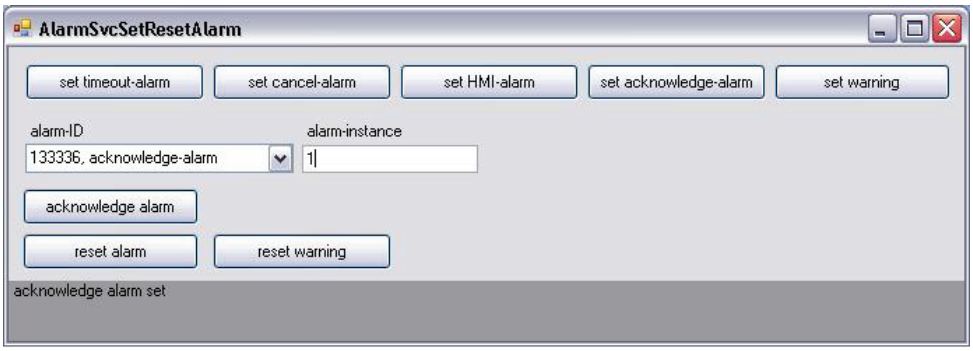


Fig. 5-4: Creating and resetting HMI alarms

Step 1

Create the required member variables of the AlarmSvc and Alarmsource type.

```
AlarmSvc m_AlarmSvc = null;
AlarmSource m_AlarmSource = null;
```

Step 2

Create the AlarmSvc object and an Alarmsource object. The alarm source to be used has the ID 100001.

```
m_AlarmSvc = new AlarmSvc();
m_AlarmSource = new AlarmSource(100001, "SOURCE");
```

Step 3

Create an HMI alarm that is automatically canceled after the expiration of a time defined by SINUMERIK Operate. The alarm has the number 133333.

```
Alarm alarm = new Alarm();
alarm.Id = 133333;
alarm.Source = m_AlarmSource;
m_AlarmSvc.SetTimeCancelAlarm(ref alarm);
```


© Siemens AG 2024 All Rights Reserved

Step 4

Create an HMI alarm that can be canceled with the "BigMac" key. The alarm has the number 133334.

```
Alarm alarm = new Alarm();
alarm.Id = 133334;
alarm.Source = m_AlarmSource;
m_AlarmSvc.SetCancelAlarm(ref alarm);
```

Step 5

Create an HMI alarm that can be reset with the "Reset HMI alarm" softkey in the diagnostics area. The alarm has the number 133335.

```
Alarm alarm = new Alarm();
alarm.Id = 133335;
alarm.Source = m_AlarmSource;
m_AlarmSvc.SetHmiAlarm(ref alarm);
```

Step 6

Create an HMI alarm that must be acknowledged. The alarm has the number 133336.

```
Alarm alarm = new Alarm();
alarm.Id = 133336;
alarm.Source = m_AlarmSource;
m_AlarmSvc.SetAcknowledgeAlarm(ref alarm);
```

Step 7

Create a message with the number 133337.

```
Alarm alarm = new Alarm();
alarm.Id = 133337;
alarm.Source = m_AlarmSource;
m_AlarmSvc.SetWarning(ref alarm);
```

Step 8

Reset a warning that is identified by the alarm number (Id), the source and the instance. In the example, the used alarms are stored in a selection box. The first six characters contain the alarm number.

```
Alarm alarm = new Alarm();
alarm.Id = Convert.ToInt32(cbxAlarmID.Text.Substring(0, 6));
alarm.Source = m_AlarmSource;
alarm.Instance = Convert.ToInt32(txtAlarmInstance.Text);
m_AlarmSvc.ResetWarning(alarm);
```

Step 9

Reset an alarm that is identified by the alarm number (Id), the source and the instance.

```
Alarm alarm = new Alarm();
alarm.Id = Convert.ToInt32(cbxAlarmID.Text.Substring(0, 6));
alarm.Source = m_AlarmSource;
alarm.Instance = Convert.ToInt32(txtAlarmInstance.Text);
m_AlarmSvc.ResetAlarm(alarm);
```

Unrestricted © Siemens AG 2024 All Rights Reserved

Step 10

Acknowledge an alarm that is identified by the alarm number (Id), the source and the instance.

```
Alarm alarm = new Alarm();
alarm.Id = Convert.ToInt32(cbxAlarmID.Text.Substring(0, 6));
alarm.Source = m_AlarmSource;
alarm.Instance = Convert.ToInt32(txtAlarmInstance.Text);
m_AlarmSvc.AcknowledgeAlarm(alarm);
```

Step 11

Copy the files to be created at the beginning. They accompany the "AlarmSvcSetResetAlarm" example:

samplesource_db.xml
slaesvcadapconf.xml
slaesvcconf.xml ➔ copy to "\oem\sinumerik\hmi\cfg"

samplesource_alarmtext_deu.ts
samplesource_alarmtext_eng.ts ➔ copy to "\oem\sinumerik\hmi\lng"

© Siemens AG 2024 All Rights Reserved

5.3 Reference AlarmSvc

5.3.1 Definitions

Overview

The AlarmSvc object can be used to access alarms that are currently present. This is achieved using a subscription. The notification that the current alarm list has changed is realized by calling the implemented delegate.

The class is defined in the "Siemens.Sinumerik.Operate.Services" namespace.

Subscription

The client-side access to alarms (e.g. for an alarm display) is implemented using subscriptions and the alarm service manages a dedicated subscription for each client. If the state of an alarm changes, the client is notified via the subscription assigned to it by calling the implemented delegates.

The following subscriptions are currently available:

- Subscription with alarm list
- Subscription with event list
- Subscription with individual events

Note

No provision is made for sharing of one subscription by multiple clients.

5.3.2 Creating am AlarmSvc object

The following constructors are available to create an AlarmSvc object:

Table 5-4: Creating an AlarmSvc object

public AlarmSvc() **public AlarmSvc(string language)** **public AlarmSvc(string language, string server)** **public AlarmSvc(string language, string server, int eventlistLength)**	
Parameter	**Meaning**
language	Language, in which the alarm texts are to be communicated. ("eng", "deu", etc.) The default language is English.
server	Defines which NC is returning data in 1:N constellations.
int	Defines the size of the event list, default: 20

Unrestricted © Siemens AG 2024 All Rights Reserved

5.3.3 Notification about a change in the current alarm list

Creating a subscription (Subscribe)

Using the function Subscribe(), a notification is created about the change in the current alarm list. The call returns as soon as the setup request has been forwarded to the server. The notification is realized by calling the delegate implementation AlarmListChanged(). The first notification returns the state at the time of the setup.

Notifications can be canceled using an Unsubscribe call. The request is also canceled as soon as the deletion of the relevant AlarmSvc object is detected.

Table 5-5: Subscribe

public Guid AlarmSvc.Subscribe(AlarmListChanged cb)	
Parameter	**Meaning**
cb	Delegate implementation for the state change message.
Possible exceptions	AlarmSvcException

Table 5-6: AlarmListChanged – specification for the delegate implementation

public delegate void AlarmSvc.AlarmListChanged(System.Guid guid Alarm[] alarms)	
Parameter	**Meaning**
guid	GUID, which was returned for a Subscribe call. It is used for identification.
alarms	Array of alarm objects. This array reflects alarms that are currently pending. See Chapter: 5.4 Reference Alarm

Example: See Chapter 5.2.2

Unsubscribing a running subscription (UnSubscribe)

The UnSubscribe function is used to end a running subscription.

Table 5-7: Unsubscribe – unsubscribing a subscription

public void AlarmSvc.UnSubscribe(AlarmListChanged cb)	
Parameter	**Meaning**
cb	Delegate implementation for the state change message.
Possible exceptions	AlarmSvcException

Example: See Chapter 5.2.2

© Siemens AG 2024 All Rights Reserved

5.3.4 Notification about a change in the event list

Subscribe to the changes of the alarm-event list (SubscribeEventList)

The SubscribeEventList() function creates a notification about a change to the content of the alarm-event ring buffer; this notification contains the state changes (coming, going, acknowledgment) of the alarms. The call returns as soon as the setup request has been forwarded to the server. The notification is realized by calling the implementation of the AlarmEventListChanged() delegate. The first notification returns the content of the event ring buffer at the time of creation. Subsequently, the complete content of the ring buffer is returned when new events occur. Because the size of the list cannot be infinite, the oldest elements are deleted when new events are generated (FIFO principle).

Table 5-8: SubscribeEventList

public Guid AlarmSvc.SubscribeEventList(AlarmEventListChanged cb)	
Parameter	**Meaning**
cb	Delegate implementation for reporting the alarm-event list changes.
Possible exceptions	AlarmSvcException

Table 5-9: AlarmEventListChanged

public delegate void AlarmEventListChanged(System.Guid guid, Alarm[] alarms)	
Parameter	**Meaning**
guid	GUID that was returned for the SubscribeEventList call. It is used for identification.
alarms	Array of alarm objects. It contains the alarm events from the ring buffer.

Unsubscribe the subscription to the alarm-event list (UnSubscribeEventList)

The call of the UnSubscribeEventList function ends the subscription to the alarm-event list.

Table 5-10: UnsubscribeEventList

public void AlarmSvc.UnSubscribeEventList(AlarmEventListChanged cb)	
Parameter	**Meaning**
cb	Delegate implementation for reporting the alarm-event list changes.
Possible exceptions	AlarmSvcException

5.3.5 Notification about a change in an individual event

Subscribe to new alarm events (SubscribeEvents)

The SubscribeEvents() function creates a notification for new alarm events, i.e. state changes (coming, going, acknowledgment) of the alarms. The call returns as soon as the setup request has been forwarded to the server. The notification is realized by calling the implementation of the NewAlarmEvents() delegate. The first notification returns the content of the event ring buffer at the time of creation. Subsequently, newly occurred alarm events are returned individually.

Unrestricted © Siemens AG 2024 All Rights Reserved

Table 5-11: SubscribeEvents

public Guid AlarmSvc.SubscribeEvents(NewAlarmEvents cb)	
Parameter	**Meaning**
cb	Delegate implementation for signaling new alarm events.
Possible exceptions	AlarmSvcException

Table 5-12: NewAlarmEvents

public delegate void NewAlarmEvents(System.Guid guid, Alarm[] alarms)	
Parameter	**Meaning**
guid	GUID that was returned for the SubscribeEvents call. It is used for identification.
alarms	Array of alarm objects. It contains the new alarm events.

Unsubscribe the subscription to new alarm events (UnSubscribeEvents)

The call of the UnSubscribeEvents function ends the subscription to individual alarm events.

Table 5-13:UnSubscribeEvents

public void AlarmSvc.UnSubscribeEvents(NewAlarmEvents cb)	
Parameter	**Meaning**
cb	Delegate implementation for signaling new alarm events.
Possible exceptions	AlarmSvcException

5.3.6 Creating, acknowledging and canceling alarms

Triggering an alarm with automatic cancellation by timeout (SetTimeCancelAlarm)

The SetTimeCancelAlarm() function triggers an HMI alarm that is automatically canceled after a time specified by SINUMERIK Operate. The transferred parameter is the reference to an alarm object for which the attributes for the alarm number and the alarm source must be defined. After triggering the alarm, the assigned instance can be obtained from the alarm object.

Table 5-14: SetTimeCancelAlarm - trigger timeout alarm

public void AlarmSvc.SetTimeCancelAlarm(ref Alarm alarm)	
Parameter	**Meaning**
alarm	Reference to a transferred alarm object.
Possible exceptions	AlarmSvcException

Triggering an "HMI Cancel button" alarm (SetCancelAlarm)

The SetCancelAlarm() function triggers an HMI alarm that is canceled by pressing the Alarm Cancel key ("BigMac" key) or by calling the ResetAlarm() function. The transferred parameter is the reference to an alarm object for which the attributes for the alarm number and the alarm source must be defined. After triggering the alarm, the assigned instance can be obtained from the alarm object.

© Siemens AG 2024 All Rights Reserved

Table 5-15: SetCancelAlarm - trigger "HMI Cancel button" alarm

public void AlarmSvc.SetCancelAlarm(ref Alarm alarm)	
Parameter	**Meaning**
alarm	Reference to a transferred alarm object.
Possible exceptions	AlarmSvcException

Triggering an alarm that does not require acknowledgment (SetHMIAlarm)

The SetHMIAlarm() function triggers an HMI alarm not requiring acknowledgment that can be canceled with the "Delete HMI alarm" softkey in the diagnostics area or by calling the ResetAlarm() function. The transferred parameter is the reference to an alarm object for which the attributes for the alarm number and the alarm source must be defined. After triggering the alarm, the assigned instance can be obtained from the alarm object.

Table 5-16: SetHMIAlarm - trigger an alarm not requiring acknowledgment

public void AlarmSvc.SetHMIAlarm(ref Alarm alarm)	
Parameter	**Meaning**
alarm	Reference to a transferred alarm object.
Possible exceptions	AlarmSvcException

Triggering an alarm requiring acknowledgment (SetAcknowledgeAlarm)

The SetAcknowledgeAlarm() function can be used to trigger an alarm that can be canceled with the "Acknowledge alarm" and the "Delete HMI alarm" softkeys in the diagnostic area or by calling the AcknowledgeAlarm() and ResetAlarm() functions. The transferred parameter is the reference to an alarm object for which the attributes for the alarm number and the alarm source must be defined. After triggering the alarm, the assigned instance can be obtained from the alarm object.

Table 5-17: SetAcknowledgeAlarm – trigger an alarm requiring acknowledgment

public void AlarmSvc.SetAcknowledgeAlarm(ref Alarm alarm)	
Parameter	**Meaning**
alarm	Reference to a transferred alarm object.
Possible exceptions	AlarmSvcException

Triggering a warning (SetWarning)

The SetWarning() function triggers a warning. This can by canceled by calling the ResetWarning() function. The transferred parameter is the reference to an alarm object for which the attributes for the alarm number and the alarm source must be defined. After triggering the warning, the assigned instance can be obtained from the alarm object.

Table 5-18: SetWarning - trigger a warning

public void AlarmSvc.SetWarning(ref Alarm alarm)	
Parameter	**Meaning**
alarm	Reference to a transferred alarm object.
Possible exceptions	AlarmSvcException

Unrestricted © Siemens AG 2024 All Rights Reserved

Acknowledging an alarm (AcknowledgeAlarm)

The AcknowledgeAlarm() function allows the client to acknowledge an alarm requiring acknowledgment. To identify the alarm, the transferred alarm object must specify the attributes for the alarm number, the alarm source and the instance.

Table 5-19: AcknowledgeAlarm - acknowledge an alarm

public void AlarmSvc.AcknowledgeAlarm(Alarm alarm)	
Parameter	**Meaning**
alarm	Alarm object with the data of the alarm to be acknowledged.
Possible exceptions	AlarmSvcException

Resetting an alarm (ResetAlarm)

The ResetAlarm() function allows the client to reset an alarm. To identify the alarm, the transferred alarm object must specify the attributes for the alarm number, the alarm source and the instance.

Table 5-20: ResetAlarm - reset an alarm

public void AlarmSvc.ResetAlarm(Alarm alarm)	
Parameter	**Meaning**
alarm	The Alarm object with the data of the alarm to be reset.
Possible exceptions	AlarmSvcException

Resetting a warning (ResetWarning)

The ResetWarning() function allows the client to reset a warning. To identify the warning, the transferred alarm object must specify the attributes for the alarm number, the alarm source and the instance.

Table 5-21: ResetWarning - reset a warning

public void AlarmSvc.ResetWarning(Alarm alarm)	
Parameter	**Meaning**
alarm	The Alarm object with the data of the warning to be reset.
Possible exceptions	AlarmSvcException

Note

If several alarm sources are required to trigger alarms and warnings, an AlarmSvc object should be used for each alarm source.

© Siemens AG 2024 All Rights Reserved

5.3.7 Further properties

Sources

Supplies an array of AlarmSources objects that contains all available alarm sources. See Chapter 5.5 Reference AlarmSource.

Table 5-22:ID

public AlarmSource[] Sources { get; }

Server

Only relevant in 1:N constellations. The server returns the name of the NCU sending the alarms. Default "empty string" means alarms are coming from all NCUs.

Table 5-23:ID

public Server { get; }

AlarmSvc in 1:N constellations

By default, all NCUs are taken into account in subscriptions. To restrict a subscription to a specific NCU in a 1:N constellation, proceed as follows:

1. Create AlarmSvc object with NCU name.
   By specifying the name of an NCU (example: NCU2) when an AlarmSvc object is created (see AlarmSvc constructors), only alarms from the specified NCU will be received.

Note

NCU names are listed in the MMC.INI file.
Entry:
[GLOBAL]
NcddeMachineNames=*NCU1,NCU2*

Unrestricted © Siemens AG 2024 All Rights Reserved

5.4 Reference Alarm

5.4.1 Definitions

Overview

An instance of the alarm class represents a pending alarm. The alarm class only contains properties. These properties ensure access to all available attributes of an alarm.
The class is defined in the "Siemens.Sinumerik.Operate.Services" namespace.

5.4.2 Properties

ID

The alarm ID (alarm number) identifies an alarm in the context of a particular source (alarm source), i.e. the source and alarm ID determine the appropriate alarm text.

Important

It is possible to reuse the same alarm ID within different alarm sources (source IDs) and to provide it with different alarm texts. That is, the alarm IDs no longer have to be unique across all alarm sources. This means that (unique) assignment between an alarm and alarm text is only possible with the tuple alarm ID plus source ID.

Table 5-24:ID

public int Id { get; set; }

A separate number range is reserved for each of the alarm sources. The following table lists the number ranges:

Table 5-25: Alarm number ranges

Number range	Source	Description	
000 000 – 009 999	NCK	General alarms	
010 000 – 019 999		Channel alarms	
020 000 – 029 999		Axis / spindle alarms	
030 000 – 039 999		Functional alarms	General information
040 000 – 059 999			Reserved
060 000 – 064 999			Cycle alarms SIEMENS
065 000 – 069 999			Cycle alarms user
070 000 – 079 999			Compile cycle manufacturer and OEM
080 000 – 081 999			Messages standard cycles
082 000 – 082 999			Messages Shopmill & CMT cycles
083 000 – 084 999			Messages, measurement cycles
085 000 – 089 999			Messages, user cycles
090 000 – 099 999			Reserved
100 000 – 129 000	HMI	System	
130 000 – 139 000		OEM	
140 000 – 199 999		Reserved	

© Siemens AG 2024 All Rights Reserved

Number range	Source	Description	
200 000 – 299 999	NCK	SINAMICS drive	
300 000 – 399 999		611D drive	
400 000 – 499 999	PLC	General alarms	
500 000 – 599 999		Channel alarms	
600 000 – 699 000		Axis / spindle alarms	
700 000 – 799 999		User area	
800 000 – 899 999		Sequencers/graphs	
(810 000 – 810 009)		System error messages	
900 000 – 999 999	NCK	611U drive	

Source

Alarm source of the alarm.

Table 5-26:Source

public AlarmSource Alarm.Source { get; set; }

Message

Alarm text in the specified language.

Table 5-27:Message

public string Alarm.Message { get; }

CancelGroup

Alarms with the same CancelGroup for a collective cancellation group. When one alarm is canceled, all alarms of the same group are automatically also canceled.

Table 5-28:CancelGroup

public AlarmCancelGroup Alarm.CancelGroup { get; }

Possible values:

`AlarmCancelGroup.NONE`
Value not assigned.

`AlarmCancelGroup.HMI`
Cancels all HMI alarms.

`AlarmCancelGroup.HMI_AND_NCU`
Cancels all NCU and HMI alarms.

Category

Category of the alarms

Table 5-29:Category

public AlarmCategory Alarm.Category { get; }

Unrestricted © Siemens AG 2024 All Rights Reserved

Possible values:

`AlarmCategory.NONE`
Value not assigned.

`AlarmCategory.SINUMERIK`
Alarm is a general SINUMERIK alarm.

`AlarmCategory.PARTPROG_MESSAGE`
Alarm is a part program message (MSG() call in the part program).

ClearInfo

The ClearInfo (clear criterion) identifies a group of alarms. The value provides information about the action (e.g. "NC Reset") that can cancel an alarm (and therefore all of the alarms of this group). But in contrast to the CancelGroup, the clear operation for these alarms is not carried out by the AlarmService. Rather, it occurs directly in the alarm source (e.g. NC kernel), i.e. the NC kernel sends a going event as an alarm source for each alarm.

Table 5-30:ClearInfo

public AlarmClearInfo Alarm.ClearInfo { get; }

The following table contains all pre-defined values for ClearInfo. Presently, the AlarmClearInfo structure has still not been assigned - and the numerical values are saved.

Table 5-31: Overview of the cancel criteria

Cancel criterion (ClearInfo)	**Alarm source (source)**	**Description**
0	HMI	Alarms that are canceled by the HMI.
1	NCK	Alarms that are canceled by power-on of the NCU.
2		Conditions are canceled by a hardware reset of the NCU.
3		Conditions are canceled by a "Cancel" command to the NCU.
4		Conditions are canceled by the NCK itself.
5		Conditions are canceled by an "NC Start" command of the NCU.
6		Conditions are canceled by a reset of the mode group.
7		Conditions are canceled by an "NC Reset" command on the NCU.
8	PLC	PLC messages of the FB15 (basic program).
9		PLC alarm of the FB15 (basic program).
10		Dialog alarms of the HMI that are canceled by the "Recall" key [^].
11		Reserved
12		S7-PDiag, S7-Graph, S7-HiGraph or other Alarm_S(Q) alarms of the PLC (SFC17/18) with alarm state "not acknowledged".
13		S7-PDiag, S7-Graph, S7-HiGraph or other Alarm_S(Q) alarms of the PLC (SFC17/18) with alarm state "acknowledged".
14	Drive (NCK)	Drive alarms via NCK.

© Siemens AG 2024 All Rights Reserved

Comment

Comment for an alarm.

Table 5-32:Comment

public string Alarm.Comment { get; }

HelpFileUrl

The HelpFileUrl can be used to locate an HTML file locally or on a server in the network and to display it.

Table 5-33:HelpFileUrl

public string Alarm.HelpFileUrl { get; }

Instance

Instance ID. Several alarms can occur at the same time within one alarm source. The instance ID is used to differentiate these alarm instances.

Table 5-34:Instance

public int Alarm.Instance { get; set;}

Location

The display location specifies at which location an alarm is displayed. This is, for example, the alarm line in the header or a message window at the center of the screen.

Table 5-35:Location

public AlarmDisplayLocation Alarm.Location { get; }

Possible values:

`AlarmDisplayLocation.NONE`
Value not assigned.

`AlarmDisplayLocation.HEADER`
Alarm is displayed in the alarm line.

`AlarmDisplayLocation.SCREEN`
Alarm is displayed as a "message box" in the center of the screen.

Parameters

Contains the dynamic parameters (associated values, process values) of an alarm.

Table 5-36:Parameters

public string[] Alarm.Parameters { get; set; }

Priority

Priority of an alarm. The priority is assigned when configuring the alarm attributes.

Table 5-37:Priority

public int Alarm.Priority { get; }

Unrestricted © Siemens AG 2024 All Rights Reserved

Quality

The quality provides information about the reliability of an alarm.

Table 5-38:Quality

public AlarmQuality Alarm.Quality { get; }

Possible values:

`AlarmQuality.QUALITY_BAD`
The quality of the alarm received is poor, i.e. the data should not be used.

`AlarmQuality.QUALITY_GOOD`
The quality of the alarm received is good, i.e. the data can be used.

`AlarmQuality.QUALITY_UNCERTAIN`
AlarmSvc cannot ascertain whether the data received is good or poor.

More precise information regarding the quality of the data received can be possibly made using the QualityDetails property.

QualityDetails

Provides more detailed information about the quality of the data received.

Table 5-39:QualityDetails

public AlarmQualityDetails Alarm.QualityDetails { get; }

Possible values:

`AlarmQualityDetails.NONE`
Value not set.

QUALITY_GOOD range:

`AlarmQualityDetails.LOCAL_OVERRIDE`
The value was overwritten. This means that typically a connection has been aborted and the value was manually set.

QUALITY_BAD range:

`AlarmQualityDetails.CONFIG_ERROR`
Problems with the configuration, for instance, if the item in the inquiry was already cleared by the configuration of the server.

`AlarmQualityDetails.NOT_CONNECTED`
A connection has been aborted.

`AlarmQualityDetails.DEVICE_FAILURE`
The device has a fault.

`AlarmQualityDetails.SENSOR_FAILURE`
The sensor has a fault.

`AlarmQualityDetails.LAST_KNOWN`
The last known value is supplied as a result of a communication problem. The time stamp (TimeStamp) can be used to see just how old the value is.

© Siemens AG 2024 All Rights Reserved

AlarmQualityDetails.COMM_FAILURE
There is a communication error between server and device, no "last read value" can be transferred.

AlarmQualityDetails.OUT_OF_SERVICE
This quality is also supplied if the value in the server was not updated.

QUALITY_UNCERTAIN range:

AlarmQualityDetails.LAST_USABLE
AlarmQualityDetails.SENSOR_CAL
AlarmQualityDetails.EGU_EXCEEDED
AlarmQualityDetails.SUB_NORMAL
These values are reserved.

State

Alarm state

Table 5-40:State

public AlarmState Alarm.State { get; }

Possible values:

AlarmState.UNKNOWN
Unknown state.

AlarmState.ALARM_STATE_ACTIVE
Alarm is active.

AlarmState.ALARM_STATE_ACKED
Alarm is acknowledged.

AlarmState.ALARM_STATE_ENABLED
Alarm is enabled.

AlarmState.ALARM_CHANGE_ACTIVE
The active flag has changed.

AlarmState.ALARM_CHANGE_ACKED
The acknowledged flag has changed.

AlarmState.ALARM_CHANGE_ENABLED
The enabled flag has changed.

AlarmState.ALARM_ACK_REQUIRED
The alarm requires an acknowledgment.

AlarmState.ALARM_HMI_CANCEL_BTN
HMI alarm with automatic cancellation by the cancel key ("BigMac" key) of the HMI.

AlarmState.ALARM_HMI_CANCEL_TIME
HMI alarm with automatic cancellation by timeout.
Alarm states of alarms not requiring acknowledgment:

AlarmState.ALARM_COMING
Alarm not requiring acknowledgment is coming.
(ALARM_STATE_ENABLED + ALARM_STATE_ACTIVE + ALARM_CHANGE_ACTIVE)

Unrestricted © Siemens AG 2024 All Rights Reserved

AlarmState.ALARM_CAME_GOING
Alarm not requiring acknowledgment came and is now going.
(ALARM_STATE_ENABLED + ALARM_CHANGE_ACTIVE)

Alarm states of HMI alarms not requiring acknowledgment with cancellation using the cancel key ("BigMac" key):

AlarmState.ALARM_HMI_CANCEL_BTN_COMING
An HMI alarm not requiring acknowledgment with cancellation using the cancel key ("BigMac" key) is coming.
(ALARM_HMI_CANCEL_BTN + ALARM_COMING)

AlarmState.ALARM_HMI_CANCEL_BTN_CAME_GOING
An HMI alarm not requiring acknowledgment with cancellation using the cancel key ("BigMac" key) is going.
(ALARM_HMI_CANCEL_BTN + ALARM_CAME_GOING)

Alarm states of HMI alarms not requiring acknowledgment with cancellation by timeout:

AlarmState.ALARM_HMI_CANCEL_TIME_COMING
An HMI alarm not requiring acknowledgment with cancellation by timeout is coming.
(ALARM_HMI_CANCEL_TIME + ALARM_COMING)

AlarmState.ALARM_HMI_CANCEL_TIME_CAME_GOING
An HMI alarm not requiring acknowledgment with cancellation by timeout is going.
(ALARM_HMI_CANCEL_TIME + ALARM_CAME_GOING)

Alarm states of alarms requiring acknowledgment:

AlarmState.ALARM_COMING_TOACK
Alarm requiring acknowledgment is coming and still has to be acknowledged.
(ALARM_ACK_REQUIRED + ALARM_COMING)

AlarmState.SLAE_ALARM_CAME_GOING_TOACK
Alarm requiring acknowledgment came and is going but still has to be acknowledged.
(ALARM_ACK_REQUIRED + ALARM_CAME_GOING)

AlarmState.ALARM_CAME_GONE_ACKING
Alarm requiring acknowledgment came, went, and is being acknowledged.
(ALARM_ACK_REQUIRED + ALARM_STATE_ENABLED + ALARM_STATE_ACKED + ALARM_CHANGE_ACKED)

AlarmState.ALARM_CAME_ACKING
Alarm requiring acknowledgment came and is being acknowledged.
(ALARM_ACK_REQUIRED + ALARM_STATE_ENABLED + ALARM_STATE_ACTIVE + ALARM_STATE_ACKED + ALARM_CHANGE_ACKED)

AlarmState.ALARM_CAME_ACKED_GOING
Alarm requiring acknowledgment came, was acknowledged, and is going.
(ALARM_ACK_REQUIRED + ALARM_STATE_ENABLED + ALARM_STATE_ACKED + ALARM_CHANGE_ACTIVE)

TimeStamp

Alarm time stamp

Table 5-41:TimeStamp

public System.DateTime Alarm.TimeStamp { get; }

© Siemens AG 2024 All Rights Reserved

5.5 Reference AlarmSource

5.5.1 Definitions

Overview

An instance of the AlarmSource class represents an Alarm source. An alarm source is defined using an identification number (ID) and a name. The class only contains properties.
The class is defined in the "Siemens.Sinumerik.Operate.Services" namespace.

5.5.2 Properties

Id

Identification number (ID) of the alarm source.

Table 5-42:Id

public int AlarmSource.Id { set; get; }

Name

Name of the alarm source.

Table 5-43:Name

public string AlarmSource.Name { set; get; }

Unrestricted © Siemens AG 2024 All Rights Reserved

5.6 FAQs

5.6.1 FAQs regarding alarm texts

How can you always obtain the alarm texts in the currently selected language?

Using the infrastructure object (see Chapter 7), it is possible to obtain a notification if the current language has changed. If this notification is received, the current alarm subscription must be exited and re-initiated. In this case, the language code used in the notification can be used.

© Siemens AG 2024 All Rights Reserved

6 Directory and file services

6

Objective of the chapter

This chapter describes the interface of the FileService. This service is used for file and directory operations.

Unrestricted © Siemens AG 2024 All Rights Reserved

6.1 Introduction

6.1.1 Class model

Overview

The class model of the file service consists primarily of the following classes:

- FileSvc
- Node
- FileSvcStatus
- Device

The node class also uses the following classes in addition:

- NodeAttributes
- ControlAccessRights

Namespace

All classes are defined in the "Siemens.Sinumerik.Operate.Services" namespace. As a consequence, this namespace can be integrated into the C# project at the start of the particular file:

```
using Siemens.Sinumerik.Operate.Services;
```

Class FileSvc

File and directory operations can be implemented using FileSvc objects. These are especially:

- Creating files/directories
- Copying files
- Shifting files
- Deleting files/directories
- Renaming files/directories

In addition, functions for SINUMERIK-specific tasks are available. For example, access rights can be modified and part programs can be selected for execution.

Many calls are available in both synchronous and asynchronous forms. Synchronous calls block the thread in which the call takes place, until the request has been completed. For asynchronous calls, after completing the operation, an implementation of the associated delegate is called.

Class Node

The node class represents a file or a directory. The corresponding attributes of the file or the directory can be accessed via the various properties of the class. These include, for example: Access rights, time stamp of the last access or size.

Class NodeAttributes

The NodeAttributes class is used by the node class for the property attributes and contains the data for file size and the time stamp of the last access.

© Siemens AG 2024 All Rights Reserved

Class ControlAccessRights

The ControlAccessRights class is used by the node class for the property AccessRights and contains the data for the access rights in the sense of SINUMERIK (access level for reading, writing, displaying, executing and deleting).

Class FileSvcStatus

The FileSvcStatus object contains information about the status of a request. These are essentially an error number and a flag as to whether the request was successfully completed.

Class Device

The Device class includes methods to access mass memories, for instance, a USB flash drive or CD-ROM. Further, using this class, it is possible to be notified as to whether a new device was connected or removed.

6.1.2 Explanation of terms

Synchronous calls

Synchronous calls return only after the request has been completed, i.e. the calling thread is blocked until this is the case. This can interfere with event processing since control inputs and displays are withheld during a synchronous call. For this reason, calls that may take a long time should be performed asynchronously.

Table 6-1: Synchronous calls

FileSvc call	Description
Copy	Copying files
Create	Creates files or directories
Cancel	Cancels the active asynchronous function
Delete	Deletes files or directories
List	Accesses the contents of a directory
Move	Shifting files
Rename	Renames files or directories
Select	Selects a part program for processing
DeleteMdaBuffer	Deletes the MDA buffer
WriteMdaBuffer	Writes the MDA buffer
ReadMdaBuffer	Reads the MDA buffer

Asynchronous calls

Asynchronous calls return as soon as the request has been sent to the file service. The returned error code cannot therefore indicate whether the request has been completed successfully; it can only indicate whether or not the request has been sent successfully. For example, an error occurs if call parameters are not correctly assigned. The actual request status is supplied in the return calls of the FileService (call of the implemented delegate).

Unrestricted © Siemens AG 2024 All Rights Reserved

Table 6-2: Asynchronous calls

FileSvc call	Description
CancelAsync	Cancels the active asynchronous function
CopyAsync	Copying files
DeleteAsync	Deletes files or directories
ListAsync	Accesses the contents of a directory
MoveAsync	Shifting files
StartBlockTransferReadAsync	Reads a data block
StartBlockTransferWriteAsync	Starts a data block write operation
WriteBlockTransferDataAsync	Provides net data for writing data block

Important

Only one asynchronous service can run on a FileSvc object at the same time. If an attempt is made to start a second service, then the FileService issues a FileSvcException.

Files and directories

The following information (attributes) can be queried for each addressable file and directory.

Table 6-3: Attributes of files and directories

Characteristic	Meaning
Logical path	The logical path of the file or directory. (Example: "//NC/MPF.DIR/TEST.MPF")
Real path	The real path of the file or directory. (Example: "/_N_MPF_DIR/_N_TEST_MPF")
Data format	Defines whether an object is a file or a directory.
Name	Name of the file. (Example: "TEST.MPF")
Access rights	Access rights for the file/directory. (Example: "77777")
Size	File size, in bytes. For a directory, this value is –1.
Time stamp	Time stamp of last change.

Files can be stored in one of the following locations:

1. On the PCU, e.g. "C:\tmp\test.mpf" or "C:/tmp/test.mpf"
2. On the CF card, e.g. "//NC/CF_CARD/oem/sinumerik/hmi/appl/test.mpf"
3. In the file structure of the NC, e.g. "//NC/mpf.dir/test.mpf"
4. On a USB drive, e.g. "//USB:/mpf.dir/test.mpf"
5. On a network drive, e.g. "//NET:/mpf.dir/test.mpf"

Access rights

For files and directories, there are eight access authorization levels.

Table 6-4: The eight access authorization levels

Access level	Can be accessed with	User group
S0	System password	SIEMENS
S1	Machine manufacturer password	Machine manufacturer
S2	Service password	Commissioning engineer, service (machine manufacturer)
S3	End user password	Privileged end user (in-house service)
S4	Keyswitch position 3	Programmers
S5	Keyswitch position 2	Trained operator
S6	Keyswitch position 1	Operator
S7	Keyswitch position 0	Semi-skilled operator (NC Start / NC Stop, MCP)

© Siemens AG 2024 All Rights Reserved

S0 denotes the access level with the highest rights and S7 the access level with the most restrictive rights.

The keyswitch position is enabled by the current keyswitch position or by entry of a password.

Access rights

The following rights can be assigned:

READ → for reading
WRITE → for writing
EXECUTE → for executing
SHOW → for listing
DELETE → for deleting

Unrestricted © Siemens AG 2024 All Rights Reserved

6.2 Step-by-step examples

Overview

The following chapters show the various application areas of the file service. An executable example is included in the SINUMERIK Integrate Create MyHMI /3GL for each "step-by-step example".

These and all other examples are shown in the following overview of examples.

Table 6-5: Overview of the examples

Application	Type	Example	Chapter
Lists the contents of a directory	synchronous	FileSvcFileListSync	6.2.2
	asynchronous	FileSvcFileListAsync	6.2.3
Creates, copies, and deletes a file	synchronous	FileSvcCreateCopyRemoveSync	6.2.4
	asynchronous	FileSvcCreateCopyRemoveAsync	6.2.5
Gets file or directory attributes		FileSvcFileAttributes	6.2.6
Block-by-block reading and writing of data	asynchronous	FileSvcBlocktransfer	6.2.7
Setting up a notification when a USB flash drive was inserted or removed.	-	FileSvcUSBNotification	-

Only the items relevant for the FileService are explained or described in each example. The comments for the sources contain more detailed information (e.g. screen layout, outputs, etc.).

6.2.1 Preparation

Overview

Preparatory measures are described in Chapter 2.1 ➔ "New Project". All of the items described there must be executed first.

6.2.2 Listing the contents of a directory (synchronous)

Overview

The following step-by-step example shows how to read out the content of a directory synchronously.

In the example, a path can be specified whose contents are displayed by pressing the "list folder content" button. The status bar indicates whether the call has been successful.

© Siemens AG 2024 All Rights Reserved

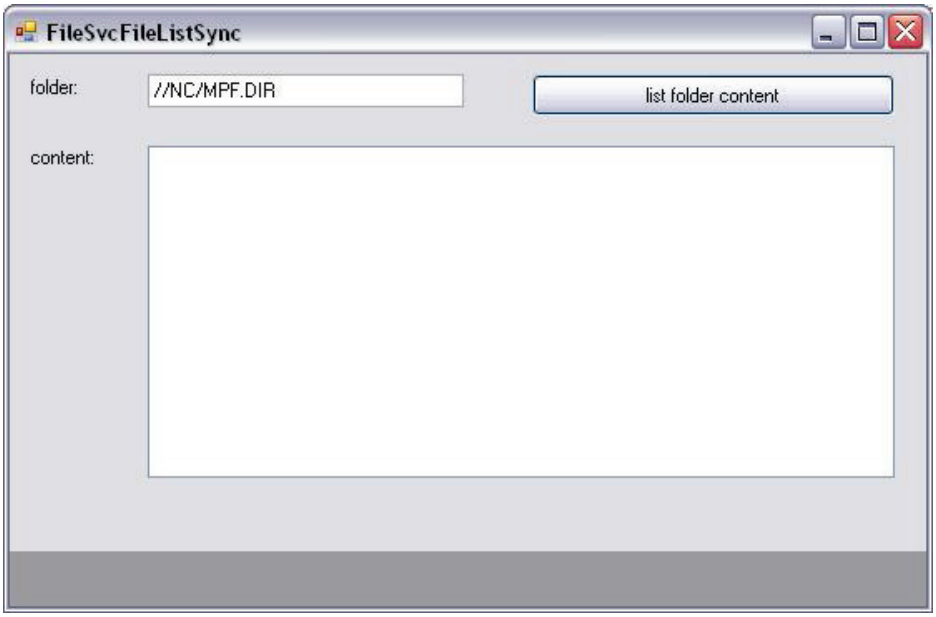


Fig. 6-1: Synchronously listing a directory

Step 1

Create a type FileSvc member variable:

```
FileSvc m_FileService = null;
```

For synchronous calls, it is not absolutely necessary to create the FileSvc object as member variable. It can also be created locally in the specific function.

Step 2

Generate the FileSvc object in the frmMain() function.

```
m_FileService = new FileSvc();
```

Step 3

Create a Node object, which represents the directory to be read-out. The contents of the text box are transferred as a path.

```
Node folderNode = new Node(txtList.Text);
```

Step 4

Read out the content. In so doing, the result is returned as an array of Node objects.

```
Node[] nodearray = m_FileService.List(folderNode);
```

Step 5

Process the data that has been read. In the example, data is written to a ListBox, this is first emptied. After this, using a "foreach loop", data is read out of the array consecutively and written into the List Box.

```
// clear listcontrol
lstFileList.Items.Clear();

// go through the array of file nodes
foreach (Node element in nodearray)
{
    // add logical path of each file node to the listcontrol
    lstFileList.Items.Add(element.LogicalPath);
}
```

Unrestricted © Siemens AG 2024 All Rights Reserved

6.2.3 Listing the contents of a directory (asynchronous)

Overview

The following step-by-step example shows how to read out the content of a directory asynchronously.

In the example, a path can be specified whose contents are displayed by pressing the "list folder content" button. The status bar indicates whether the call has been successful.

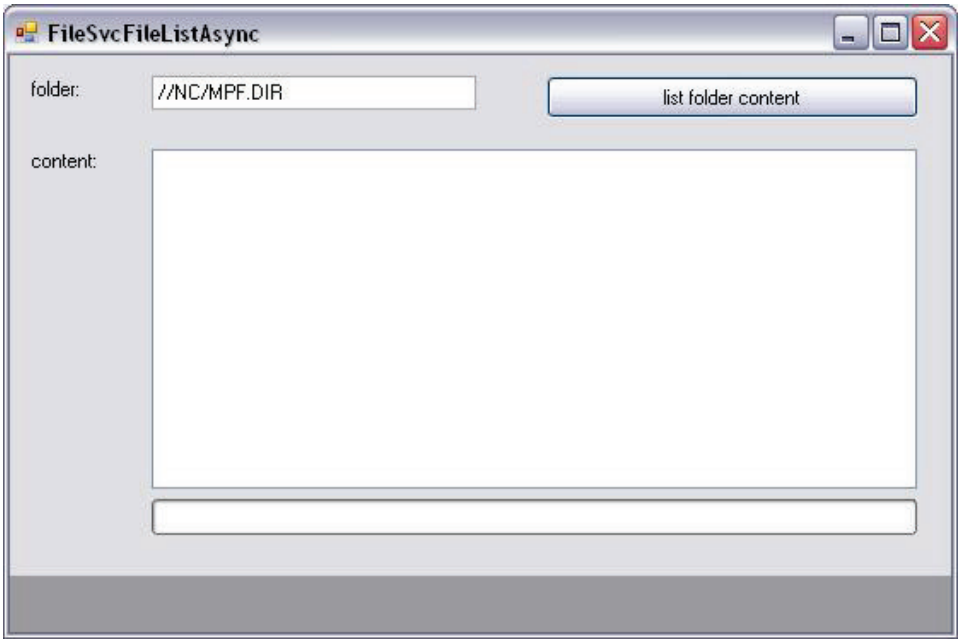


Fig. 6-2: Asynchronously listing a directory

Step 1

Create the required member variables. On the one hand, a type FileSvc variable is required, and on the other hand, a type System.Guid:

```
FileSvc m_FileService = null;
System.Guid m_FileServiceGuid
```

Step 2

Generate the FileSvc object in the frmMain() function.

```
m_FileService = new FileSvc();
```

Step 3

Create a Node object, which represents the directory to be read-out. The contents of the text box are transferred as a path.

```
Node folderNode = new Node(txtList.Text);
```


© Siemens AG 2024 All Rights Reserved

Step 4

Initiate asynchronous readout. Here, a GUID is returned, which is subsequently used to identify the delegates implemented in the calls.
In this case, two delegates are transferred. The function ListCompleted() should be called, if the request is completed and the ListProgress() function should cyclically provide information about the progress. Only the implementation of the ListCompleted() function is described in the following. ListProgress() is identical to this, and if required, can be directly viewed in the example.

```
m_FileServiceGuid = m_FileService.ListAsync(
                                                folderNode,
                                                ListCompleted,
                                                ListProgress);
```

Step 5

Implement the delegate.

! Important

Because the delegate is called from another thread, not all display elements can be accessed. Consequently, it is desirable to call a second delegate with the same signature with which changes can be performed on the user interface.

In the implemented delegate function, a change is sent to the same delegate signature, now in the GUI thread. The file and directory names of the directories to be listed are displayed here. To start, a check is made as to whether the GUID that was transferred into the delegate is the same as the one that was returned when calling ListAsync(). After this, using a foreach loop, the logical paths of all entries of the node array are output.

```
public void ListCompleted(Guid guid, Node[] nodes, FileSvcStatus status)
{
    // are we in GUI thread (main) ?
    // No we are not in GUI thread => send message to GUI thread
    if (lstFileList.InvokeRequired)
    {
        ListAsyncCompleted listDelegate = new
                                          ListAsyncCompleted(ListCompleted);
        BeginInvoke(listDelegate, new Object[] { guid, nodes, status });
    }
    else
    {
        //check if guid is correct
        if (m_FileServiceGuid.Equals(guid))
        {
            // clear listcontrol
            lstFileList.Items.Clear();

            // go through the array of nodes
            foreach (Node element in nodes)
            {
                // add each logical path of each node to listcontrol
                lstFileList.Items.Add(element.LogicalPath);
            }
        }
    }
}
```

Unrestricted © Siemens AG 2024 All Rights Reserved

6.2.4 Generating, copying, and deleting a file (synchronous)

Overview

The following example shows step-by-step how a file is created, synchronously copied, and then synchronously deleted again.

In the example, a part program with contents can be created using the "create source" button. The "copy source" softkey can be used to copy this file to the specified destination path. The two files can then be deleted again using the "remove source" or "remove destination" buttons. The status bar indicates whether the calls have been successful.

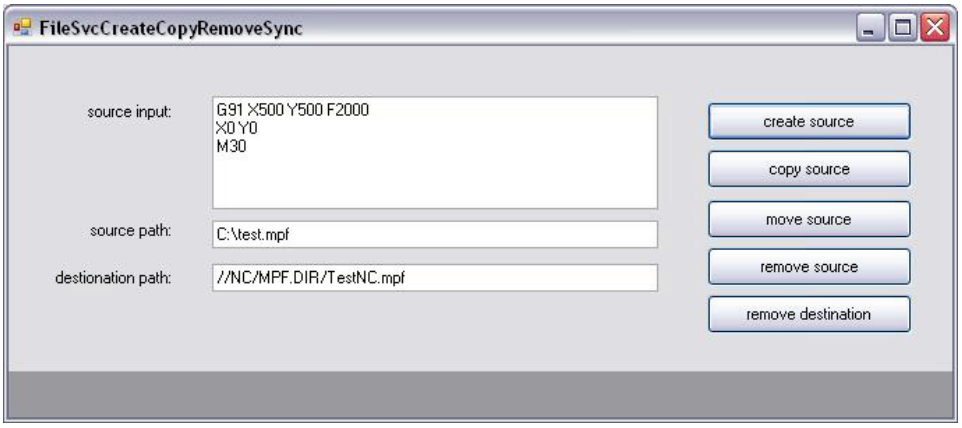


Fig. 6-3: Synchronously creating, copying, and deleting a file

Step 1

Create a type FileSvc member variable:

```
FileSvc m_FileService = null;
```

Step 2

Generate the FileSvc object in the frmMain() function.

```
m_FileService = new FileSvc();
```

Step 3

Create a file. For this purpose, a node object is created, whereby the path (source path) is taken from the text field. The file is created using the Create() function of the FileSvc. In this example, parameter "createNew" is set (true) so that the file is always overwritten if it already exists.

```
// create a node for of the source file
Node sourceNode = new Node(txtSource.Text);

// create the file, if it exists --> overwrite
m_FileService.Create(sourceNode, true);
```


© Siemens AG 2024 All Rights Reserved

Populating a file that has been generated with contents. A StreamWriter object is used for this purpose.

```
// build a streamwriter for the code
StreamWriter sourceStream = new StreamWriter(txtSource.Text, false);

// write the code to the file
sourceStream.Write(txtInput.Text);

// close stream to the file
sourceStream.Close();
```

Step 4

Copy file. Two node objects are required for copying. One represents the file that should be copied; the other, the destination file. The overwrite parameter of the Copy() function is set (true) so that an existing file is overwritten. Before the Copy() function is called, a check is made as to whether the file to be copied exists.

```
// create a node for of the source file
Node sourceNode = new Node(txtSource.Text);

// create a node for of the destination file
Node destNode = new Node(txtDestination.Text);

// check if source file exists, if false then do not copy
if (sourceNode.Exists)
{
    // copy the file, if destination exists --> overwrite
    m_FileService.Copy(sourceNode, destNode, true);
}
```

Step 5

Delete a file. It will only be shown how the destination file is deleted. To start, a node object of the file to be deleted is created. A check is then made as to whether the file exists. If it does, this file is deleted.

```
// create a node for the destination file
Node destNode = new Node(txtDestination.Text);

// check if destination file exists to remove
if (destNode.Exists)
{
    // remove the file
    m_FileService.Delete(destNode);
}
```

Unrestricted © Siemens AG 2024 All Rights Reserved

6.2.5 Generating, copying and deleting a file (asynchronous)

Overview

The following step-by-step example shows how to create, asynchronously copy, and then asynchronously delete a file again.

In the example, a part program with contents can be created using the "create source" button. This file can be asynchronously copied to the specified destination path using "copy source". Both files can then be asynchronously deleted again using the "remove source" or "remove destination" buttons. The status bar indicates whether the calls have been successful.

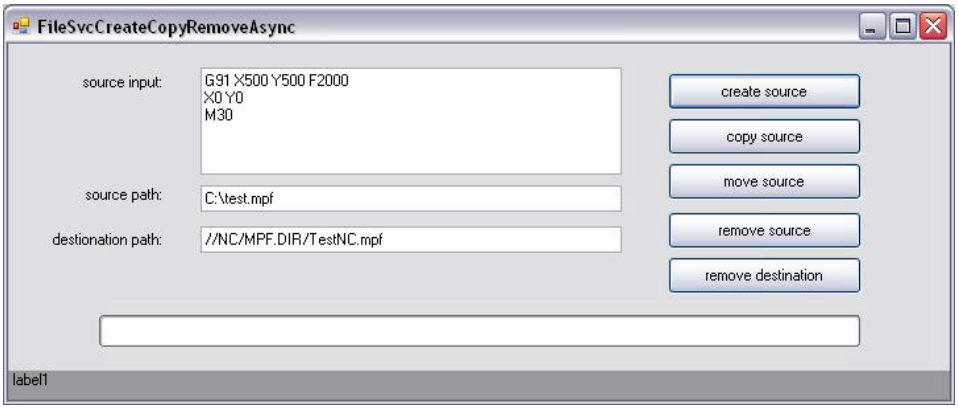


Fig. 6-4: Asynchronously creating, copying, and deleting a file

Step 1

Create the required member variables. On the one hand, a FileSvc type variable is required, and on the other hand, various System.Guid type variables. These Guids are required to identify asynchronous calls:

```
FileSvc m_FileService = null;

System.Guid m_copyGuid;
System.Guid m_deleteDestinationGuid;
```

Step 2

Generate the FileSvc object in the frmMain() function.

```
m_FileService = new FileSvc();
```

Step 3

Create and write a file in the same way as in the synchronous step-by-step example, as there is no asynchronous function for creating a file or a directory.

Step 4

Copy file. Two node objects are required for copying. One represents the file that should be copied; the other, the destination file. The overwrite parameter of the CopyAsync() function is set (true) so that an existing file is overwritten. Before the CopyAsync() function is called, a check is made as to whether the file to be copied exists. In this example, two delegates are transferred. The CopyCompleted() function should be called if the request is completed and the FileProgress()

© Siemens AG 2024 All Rights Reserved

function should cyclically provide information about the progress. In the following, only the implementation of the CopyCompleted() function is described. FileProgress() is identical to this, and if required, can be directly viewed in the example.

```
// create a node for of the source file
Node sourceNode = new Node(txtSource.Text);

// create a node for of the destination file
Node destNode = new Node(txtDestination.Text);

// check if source file exists. If file doesn't exist do not copy.
if (sourceNode.Exists)
{
    // copy the file
    m_copyGuid = m_FileService.CopyAsync(sourceNode, destNode, true,
                                         CopyCompleted, FileProgress);
}
```

Step 5

Implement the delegate.

Important

Because the delegate is called from another thread, not all display elements can be accessed. Consequently, it is desirable to call a second delegate with the same signature with which changes can be performed on the user interface.

In the implemented delegate function, a change is sent to the same delegate signature, now in the GUI thread. The processing is realized here for the case that copying has been completed (e.g. output of a message).
To start, a check is made as to whether the GUID that was transferred into the delegate, is the same as the one that was returned when calling CopyAsync().

```
private void CopyCompleted(Guid guid, FileSvcStatus status)
{
    // are we in GUI thread (main) ?
    // No we are not in GUI thread => send message to GUI thread
    if (prgBar.InvokeRequired)
    {
        AsyncCompleted copyDelegate = new AsyncCompleted(CopyCompleted);
        BeginInvoke(copyDelegate, new Object[] { guid, status });
    }
    else
    {
        //check if guid is correct
        if (m_copyGuid.Equals(guid))
        {
           // set status to statusbar
           setStatus("copy completed");
        }
    }
}
```

Unrestricted © Siemens AG 2024 All Rights Reserved

Step 6

Delete a file. It will only be shown how the destination file is deleted. To start, a node object of the file to be deleted is created. A check is then made as to whether the file exists. If it does, this file is deleted. In this example, only the RemoveCompleted() delegate is transferred.

```
// create a node to the destination file
Node destNode = new Node(txtDestination.Text);

// check if file exists - if true, delete
if (destNode.Exists)
{
    // delete file
    m_deleteDestinationGuid = m_FileService.DeleteAsync(destNode,
                                                        RemoveCompleted);
}
```

Step 7

Implement the delegate.

! Important

Because the delegate is called from another thread, not all display elements can be accessed. Consequently, it is desirable to call a second delegate with the same signature with which changes can be performed on the user interface.

In the implemented delegate function, a change is sent to the same delegate signature, now in the GUI thread. The processing is now realized here for the case that deleting has been completed (e.g. output of a message).
To start, a check is made as to whether the GUID that was transferred into the delegate is the same as the one that was returned when calling RemoveAsync().

```
private void RemoveCompleted(Guid guid, FileSvcStatus status)
{
    // are we in GUI thread (main) ?
    // No we are not in GUI thread => send message to GUI thread
    if (prgBar.InvokeRequired)
    {
        AsyncCompleted removeDelegate = new AsyncCompleted(RemoveCompleted);
        BeginInvoke(removeDelegate, new Object[] { guid, status });
    }
    else
    {
        // check if guid is on of the correct action-guids
        if (m_deleteDestinationGuid.Equals(guid))
        {
            // set status to statusbar
            setStatus("Remove complete");
        }
    }
}
```


© Siemens AG 2024 All Rights Reserved

6.2.6 Determining file or directory attributes

Overview

Attributes of a directory or a file are displayed in this example. The path of the required directory or the file can be entered into a text box. The associated attributes are read out and displayed by pressing the "get attributes" button.

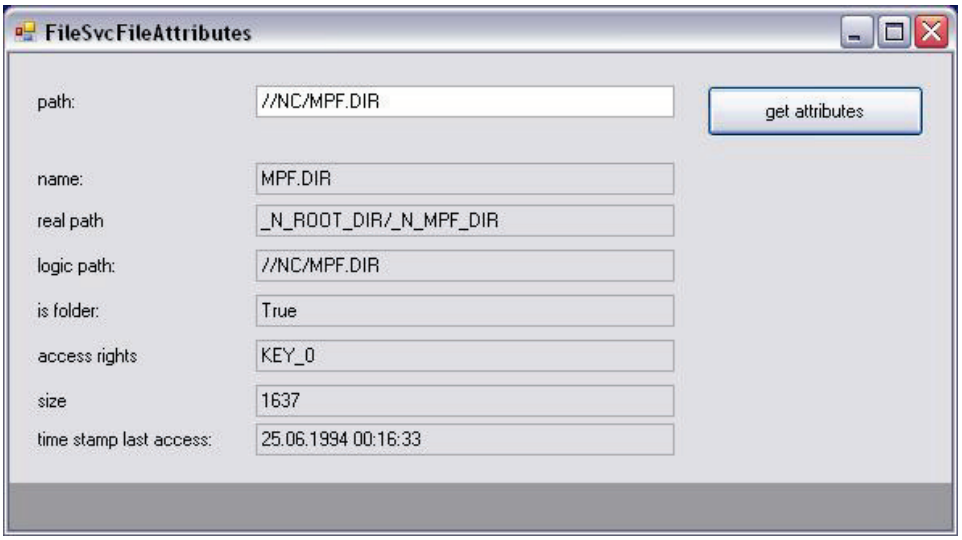


Fig. 6-5: Displaying attributes

Step 1

Create a type FileSvc member variable:

```
FileSvc m_FileService = null;
```

For synchronous calls, it is not absolutely necessary to create the FileSvc object as member variable. It can also be created locally in the specific function.

Step 2

Generate the FileSvc object in the frmMain() function.

```
m_FileService = new FileSvc();
```

Step 3

Create a Node object, which represents the object to be read-out (file or directory). The contents of the text box are transferred as a path.

```
Node fileNode = new Node(txtList.Text);
```

Step 4

"Extract" attributes from the node object. For this purpose, a type NodeAttributes variable is created.

```
// create node attributes
NodeAttributes fileAttributes = new NodeAttributes();

// get the attributes of the file/directory
fileAttributes = fileNode.Attributes;
```

Unrestricted © Siemens AG 2024 All Rights Reserved

Step 5

Output the attributes. All attributes are displayed in text boxes.

```
//show the name
txtName.Text = fileNode.Name;

// show the real path
txtRealPath.Text = fileNode.RealPath;

// show the logical path
txtLogicPath.Text = fileNode.LogicalPath;

// if IsDirNode is true, path is a directory
txtFolder.Text = fileNode.IsDirNode.ToString();

// show the accessrights
txtRights.Text = fileNode.AccessRights.List.ToString();

// show file/directory size
txtSize.Text = fileAttributes.Size.ToString();

// show last access to the file/directory
txtTimeStamp.Text = fileAttributes.LastAccess.ToString();
```

6.2.7 Block-by-block reading and writing of data

Overview

In this example, data is written block-by-block to a part program on the NC. In a further step, the part program is read block-by-block from the NC.
Click the "start blocktransfer write" button to start the block transfer to the destination specified as "filename". Click the "write new datablock" button to make the contents of the "content" text box available for transfer. Click the "end blocktransfer write" button to close the data transfer to the NC. Click the "read datablock async" button to export the data block-by-block from the source defined as "filename" and display it in "content". Click the "cancel running async service" button to cancel an active asynchronous request of the FileService.



Fig. 6-6: Block transfer of data

© Siemens AG 2024 All Rights Reserved

Step 1

Create a member variable of the FileSvc type.

```
FileSvc m_FileService = null;
```

Step 2

Generate the FileSvc object in the frmMain() function.

```
m_FileService = new FileSvc();
```

Step 3

Create a node object that represents the data source or the data destination. The path is taken from the content of the text box.

```
Node destNode = new Node(txtFilename.Text);
```

Step 4

Start block-by-block asynchronous writing. In this example, two delegates are transferred. The asyncWriteProvideNewData() function is called when the service is ready for new data. asyncWriteServiceCompleted() is called when the service completes.

```
m_FileSvc.StartBlockTransferWriteAsync(destNode, asyncWriteProvideNewData,
                                                 asyncWriteServiceCompleted);
```

Step 5

Implement the asyncWriteProvideNewData() delegate.

Important

Because the delegate is called from another thread, not all display elements can be accessed. Consequently, it is desirable to call a second delegate with the same signature with which changes can be performed on the user interface.

In the implemented delegate function, a change is sent to the same delegate signature, now in the GUI thread. This performs the processing when the FileService is ready for new data.

```
private void asyncWriteProvideNewData(Guid guid, FileSvcStatus status)
{
    // are we in GUI thread (main) ?
    // No we are not in GUI thread => send message to GUI thread
    if (btnStartBlocktransfer.InvokeRequired)
    {
        ProvideBlockTransferData provideNewWriteData = new
                        ProvideBlockTransferData(asyncWriteProvideNewData);
        BeginInvoke(provideNewWriteData, new Object[] { guid, status });
    }
    else
    {
        setStatus("OnAsyncWriteProvideNewData");
    }
}
```

Unrestricted © Siemens AG 2024 All Rights Reserved

Step 6

Implement the delegate asyncWriteServiceCompleted(). This is called when the write request completes.

```
private void asyncWriteServiceCompleted(Guid guid, FileSvcStatus status)
{
  // are we in GUI thread (main) ?
  // No we are not in GUI thread => send message to GUI thread
  if (btnStartBlocktransfer.InvokeRequired)
  {
    AsyncCompleted asyncWriteCompletedDelegate = new
                                AsyncCompleted(asyncWriteServiceCompleted);
    BeginInvoke(asyncWriteCompletedDelegate, new Object[] { guid, status });
  }
  else
  {
    setStatus("OnAsyncWriteServiceCompleted");
  }
}
```

Step 7

Write a new data block to the NC.
When the block transfer started with the StartBlockTransferWriteAsync call is ready for data, such data can be made available as a byte array with the WriteBlockTransferDataAsync function. The second parameter specifies whether or not the request with this data block has completed (true) or additional data will be supplied (false).

```
private void btnWriteNewData_Click(object sender, EventArgs e)
{
   // write new data
   Byte[] Data;
   Data = Encoding.UTF8.GetBytes(txtContent.Text);
   m_FileSvc.WriteBlockTransferDataAsync(Data, false);
}
```

Step 8

The StartBlockTransferReadAsync function initiates the block-by-block export of data from the NC. Three delegates are used. The newAsyncReadData delegate is called when the service supplies new data. The asyncReadServiceCompleted delegate is called when the reading completes and the Progress delegate provides information about the read progress.

```
m_FileSvc.StartBlockTransferReadAsync(mNode, newAsyncReadData,
                                      asyncReadServiceCompleted, Progress);
```

Step 9

The CancelAsync function can be used to cancel an active asynchronous service. The reference to the CBCanceled delegate is transferred as parameter.

```
m_FileSvc.CancelAsync(CBCanceled);
```


© Siemens AG 2024 All Rights Reserved

6.3 Reference Node

6.3.1 Definitions

Overview

A node object represents a file or a directory. The node class contains attributes that describe the particular directory or the particular file in more detail. These attributes are provided as properties. This class is defined in the "Siemens.Sinumerik.Operate.Services" namespace.

Path structure

The following examples show the structure of the logical paths, which should be used for the properties of the node class.

Table 6-6: Logical paths

Logical path	Description
//NC/mpf.dir/test.mpf	NC path
//NC/act.dir	NC path (active file system)
G:/test/test.mpf	Windows path
//USB:/tmp/test.mpf //USB:	Local TCU-USB drive with symbolic specification (The symbolic name 'USB' is assigned during the setup of the drive in the ‘commissioning/HMI/Log.drive' operating area)
//NET:/test.mpf	Network drive with symbolic name (The symbolic name 'NET' is assigned during the setup of the drive in the ‘commissioning/HMI/Log.drive' operating area)
//NC/CF_CARD/oem/sinumerik/hmi/appl/test.mpf	Linux or CF card path

Note

When using a network drive or a USB drive, a corresponding drive must first be set up from the commissioning/HMI/Log.drive operating area.

Note

Only the logical paths can be used to call the interface FileSvc. Real paths (e.g. "/_N_MPF_DIR/_N_TEST_MPF") cannot be used.

Unrestricted © Siemens AG 2024 All Rights Reserved

6.3.2 Creating a node object

A number of constructors are available for creating a node object. If you use the default constructor, the values can be set or overwritten later using the properties.

Table 6-7: Constructors of the node class

public Node.Node(String LogicalPath) **public Node.Node(String LogicalPath, String server)**	
Parameter	**Meaning**
LogicalPath	File or directory to be represented
server	In 1:N constellations, specifies from/to which NCU file operations are executed.

6.3.3 Properties

AccessRights

Returns the access rights in the sense of the SINUMERIK protection level concepts to read, write, delete, execute and list.

Table 6-8:AccessRights

public ControlAccessRights Node.AccessRights { get; }

Attributes

Returns additional attributes of the actual file or the actual directory. These include, for example, the time stamp of the last access or file size.

Table 6-9:Attributes

public NodeAttributes Node.Attributes { get; }

Exists

Defines whether the file or the directory exists.

Table 6-10:Exists

public bool Node.Exists { get; }

IsDirNode

Defines whether the object is a directory or a file.

Table 6-11:IsDirNode

public bool Node.IsDirNode { set; get; }

IsNcNode

Defines whether the object is located in the NC.

Table 6-12:IsNcNode

public bool Node.IsNcNode { set; get; }

© Siemens AG 2024 All Rights Reserved

IsOsNode

Marks whether the object is located on the NC card/hard disk.

Table 6-13:IsOsNode

public bool Node.IsOsNode { set; get; }

IsPlcNode

Presently not used.

LogicalPath

Logical path of the object.

Table 6-14:LogicalPath

public string Node.LogicalPath { get; }

Name

Name of the file or the directory (without path).

Table 6-15:Name

public string Node.Name { get; }

RealPath

Real path of the object.

Table 6-16:RealPath

public string Node.RealPath { get; }

Server

Server name for a 1:N constellation.

Table 6-17:RealPath

public string Node.Server { get; }

Unrestricted © Siemens AG 2024 All Rights Reserved

6.4 Reference FileSvc

6.4.1 Definitions

Overview

File and directory operations can be implemented using FileSvc objects. These are especially:

- Creating files/directories
- Copying files
- Shifting files
- Deleting files/directories
- Renaming files/directories

In addition, functions for SINUMERIK-specific tasks are available. For example, access rights can be modified and part programs can be selected for execution.

Many calls are available in both synchronous and asynchronous forms. Synchronous calls block the thread in which the call takes place, until the request has been completed. For asynchronous calls, after completing the operation, an implementation of the associated delegate is called.

This class is defined in the "Siemens.Sinumerik.Operate.Services" namespace.

6.4.2 Creating a FileSvc object

The FileSvc class has the following constructors.

Table 6-18: Constructors, FileSvc class

public FileSvc.FileSvc () **public FileSvc.FileSvc (String server)**	
Parameter	**Meaning**
server	In 1:N constellations, specifies from/to which NCU file operations are executed.

6.4.3 Management functions

Listing the contents of a directory (List / ListAsync)

This function lists the content of a directory. The results are returned as an array of the node class. A node object describes a file or a directory.

With large directories, the call can take a long time. In such cases, the asynchronous version (ListAsync) should be used to avoid blocking the thread in which the call is made.

Table 6-19: List

public Node[] FileSvc.List(Node node)	
Parameter	**Meaning**
Node	Directory to be listed.
Return value	Array of node objects, contents of the directory.
Possible exceptions	FileSvcException, FileSvcBusyException

Example: See Chapter 6.2.2

© Siemens AG 2024 All Rights Reserved

In the asynchronous version, after the call, a GUID is received in order to subsequently identify the call.

The function calls the implementation of the following delegates (see Chapter 6.4.4 Specifications for implementing delegates):

- ListAsyncCompleted()
- AsyncProgress()
- AsyncCanceled()

Table 6-20: ListAsync

public System.Guid FileSvc.ListAsync(Node node, ListAsyncCompleted cb, AsyncProgress prog) public System.Guid FileSvc.ListAsync(Node node, ListAsyncCompleted cb)	
Parameter	**Meaning**
Node	Directory to be listed.
cb	Delegate implementation for the message, if the request has been completed.
prog	Delegate implementation for the progress message.
Return value	GUID to identify the asynchronous operation.
Possible exceptions	FileSvcException, FileSvcBusyException

Example: See Chapter 6.2.3

Creating a directory/file (Create)

Directories or files can be created using the Create function. The directories or files can either be created in the NCU or on the PCU hard disk. A FileSvcException is initiated if the object already exists.

Table 6-21: Create

public Node FileSvc.Create(Node node) public Node FileSvc.Create(Node node, bool createNew)	
Parameter	**Meaning**
Node	Object (file or directory) to be created.
createNew	For files: Marks that the file should be overwritten For directories: Marks that all higher-level directories, which are not available, should be created.
Return value	Node object of the generated object (file or directory)
Possible exceptions	FileSvcException, FileSvcBusyException

Example: See Chapter 6.2.4

Unrestricted © Siemens AG 2024 All Rights Reserved

Renaming a directory / file (Rename)

A file or a directory can be renamed using this function.

Table 6-22: Rename

public Node FileSvc.Rename(Node node, string name)	
Parameter	**Meaning**
Node	File or directory to be renamed
name	New file name
Return value	Node object of the renamed object (file or directory)
Possible exceptions	FileSvcException, FileSvcBusyException

Example: See Chapter **6**.2.4

Copying a file (Copy / CopyAsync)

This function copies a file. If a file with the same name already exists in the destination directory, then a FileSvcException is initiated, unless the override parameter is set to true.

Table 6-23: Copy

public void FileSvc.Copy(Node srcNode, Node destNode) **public void FileSvc.Copy(Node srcNode, Node destNode, bool overwrite)**	
Parameter	**Meaning**
srcNode	Source file; file to be copied
destNode	Destination file.
override	Overwrite file if it already exists
Possible exceptions	FileSvcException, FileSvcBusyException

Example: See Chapter 6.2.4

In the asynchronous version, after the call, a GUID is received in order to subsequently identify the call.

The function calls the implementation of the following delegates (see Chapter **6**.4.4 Specifications for implementing delegates):

- AsyncCompleted()
- AsyncProgress()
- AsyncCanceled()

© Siemens AG 2024 All Rights Reserved

Table 6-24: CopyAsync

```
public System.Guid FileSvc.CopyAsync(
                                    Node srcNode,
                                    Node destNode,
                                    AsyncCompleted cb,
                                    AsyncProgress prog)

public System.Guid FileSvc.CopyAsync(
                                    Node srcNode,
                                    Node destNode,
                                    bool overwrite,
                                    AsyncCompleted cb,
                                    AsyncProgress prog)

public System.Guid FileSvc.CopyAsync(
                                    Node srcNode,
                                    Node destNode,
                                    AsyncCompleted cb)

public System.Guid FileSvc.CopyAsync(
                                    Node srcNode,
                                    Node destNode,
                                    bool overwrite,
                                    AsyncCompleted cb)
```

Parameter	Meaning
srcNode	Source file; file to be copied
destNode	Destination file.
override	Overwrite file if it already exists
cb	Delegate implementation for the message, if the request has been completed.
prog	Delegate implementation for the progress message.
Return value	GUID to identify the asynchronous operation.
Possible exceptions	FileSvcException, FileSvcBusyException

Example: See Chapter 6.2.5

Copying several files (Copy / CopyAsync)

This function copies several files. The success status of the individual copying actions is contained in the returned status.

This copy operation is carried out without taking into account Siemens access levels.

Table 6-25: Copy

```
public void Copy(Node[] srcNodes, Node destNode, out FileSvcStatus[] status);

public void Copy(Node[] srcNodes, Node destNode, out FileSvcStatus[] status,
                                                                bool overwrite);
```

Parameter	Meaning
srcNodes	Source files; the files to be copied
destNode	Destination path
status	Success status of the individual copying actions
override	Overwrite files if they already exist.
Possible exceptions	FileSvcException, FileSvcBusyException

In the asynchronous version, after the call, a GUID is received in order to subsequently identify the call.

Unrestricted © Siemens AG 2024 All Rights Reserved

The function calls the implementation of the following delegates (see Chapter 6.4.4 Specifications for implementing delegates):

- AsyncMultiFilesCompleted()
- AsyncProgress()
- AsyncCanceled()

Table 6-26: CopyAsync

public System.Guid CopyAsync(Node[] srcNodes, Node destNode, AsyncMultiFilesCompleted cb, AsyncProgress prog); **public System.Guid CopyAsync(Node[] srcNodes, Node destNode, bool overwrite, AsyncMultiFilesCompleted cb, AsyncProgress prog);** **public System.Guid CopyAsync(Node[] srcNodes, Node destNode, AsyncMultiFilesCompleted cb);** **public System.Guid CopyAsync(Node[] srcNodes, Node destNode, bool overwrite, AsyncMultiFilesCompleted cb);**

Parameter	Meaning
srcNodes	Source files; the files to be copied
destNode	Destination path
override	Overwrite file if it already exists
cb	Delegate implementation for the message, if the request has been completed.
prog	Delegate implementation for the progress message.
Return value	GUID to identify the asynchronous operation.
Possible exceptions	FileSvcException, FileSvcBusyException

Moving a file (Move / MoveAsync)

This function moves a file. If a file with the same name already exists in the destination directory, then a FileSvcException is initiated, unless the override parameter is set to true.

Table 6-27: Move

public void FileSvc.Move(Node srcNode, Node destNode) **public void FileSvc.Move(Node srcNode, Node destNode, bool overwrite)**

Parameter	Meaning
srcNode	Source file; file that should be moved
destNode	Destination file.
override	Overwrite file if it already exists
Possible exceptions	FileSvcException, FileSvcBusyException

Example: See Chapter 6.2.4
In the asynchronous version, after the call, a GUID is received in order to subsequently identify the call.

© Siemens AG 2024 All Rights Reserved

The function calls the implementation of the following delegates (see Chapter 6.4.4 Specifications for implementing delegates):

- AsyncCompleted()
- AsyncProgress()
- AsyncCanceled()

Table 6-28: MoveAsync

public System.Guid FileSvc.MoveAsync(	**Node srcNode, Node destNode, AsyncCompleted cb, AsyncProgress prog)**
public System.Guid FileSvc.MoveAsync(	**Node srcNode, Node destNode, bool overwrite, AsyncCompleted cb, AsyncProgress prog)**
public System.Guid FileSvc.MoveAsync(	**Node srcNode, Node destNode, AsyncCompleted cb)**
public System.Guid FileSvc.MoveAsync(	**Node srcNode, Node destNode, bool overwrite, AsyncCompleted cb)**

Parameter	Meaning
srcNode	Source file; file that should be moved
destNode	Destination file.
override	Overwrite file if it already exists
cb	Delegate implementation for the message, if the request has been completed.
prog	Delegate implementation for the progress message.
Return value	GUID to identify the asynchronous operation.
Possible exceptions	FileSvcException, FileSvcBusyException

Example: See Chapter 6.2.5

Deleting a directory/file (Delete / DeleteAsync)

This function deletes a directory or a file.

Table 6-29: Delete

public void FileSvc.Delete(Node node)	
Parameter	**Meaning**
Node	File that should be deleted
Possible exceptions	FileSvcException, FileSvcBusyException

Example: See Chapter 6.2.4

In the asynchronous version, after the call, a GUID is received in order to subsequently identify the call.

The function calls the implementation of the following delegates (see Chapter 6.4.4 Specifications for implementing delegates):

- AsyncCompleted()
- AsyncProgress()
- AsyncCanceled()

Unrestricted © Siemens AG 2024 All Rights Reserved

Table 6-30: DeleteAsync

public System.Guid FileSvc.DeleteAsync(Node node, AsyncCompleted cb) **public System.Guid FileSvc.DeleteAsync(** **Node node,** **AsyncCompleted cb,** **AsyncProgress prog)**	
Parameter	**Meaning**
Node	File to be deleted
cb	Delegate implementation for the message, if the request has been completed.
prog	Delegate implementation for the progress message.
Return value	GUID to identify the asynchronous operation.
Possible exceptions	FileSvcException, FileSvcBusyException

Example: See Chapter 6.2.5

Deleting several directories/files (Delete / DeleteAsync)

This function deletes several directories or files.

Table 6-31: Delete

public void Delete(Node[] nodes, out FileSvcStatus[] status);	
Parameter	**Meaning**
nodes	Files/directories that should be deleted
status	Success status of the individual deletion actions
Possible exceptions	FileSvcException, FileSvcBusyException

In the asynchronous version, after the call, a GUID is received in order to subsequently identify the call.

The function calls the implementation of the following delegates (see Chapter 6.4.4 Specifications for implementing delegates):

- AsyncMultiFilesCompleted()
- AsyncProgress()
- AsyncCanceled()

Table 6-32: DeleteAsync

public System.Guid DeleteAsync(Node[] nodes, **AsyncMultiFilesCompleted cb);** **public System.Guid DeleteAsync(Node[] nodes,** **AsyncMultiFilesCompleted cb,** **AsyncProgress prog);**	
Parameter	**Meaning**
nodes	Files/directories that should be deleted
cb	Delegate implementation for the message, if the request has been completed.
prog	Delegate implementation for the progress message.
Return value	GUID to identify the asynchronous operation.
Possible exceptions	FileSvcException, FileSvcBusyException

© Siemens AG 2024 All Rights Reserved

Selecting a part program (Select)

The function selects a program saved on the NC for the addressed channel for processing. This is only possible if the file is an executable file (e.g. a part program).

Table 6-33: Select

public void FileSvc.Select(Node node, int channel)	
Parameter	**Meaning**
Node	Part program to be selected for processing
channel	Channel in which the part program is selected
Possible exceptions	FileSvcException, FileSvcBusyException

Selecting an external part program (SelectExtern)

The function selects an externally saved program (e.g. hard disk) for the addressed channel for execution. This is only possible if the file is an executable file (e.g. a part program).

Table 6-34: SelectExtern

public void FileSvc.SelectExtern (Node node, int channel)	
Parameter	**Meaning**
Node	Part program to be selected for processing
channel	Channel in which the part program is selected
Possible exceptions	FileSvcException, FileSvcBusyException

Canceling an asynchronous action (Cancel/CancelAsync)

This function cancels an active asynchronous operation.

Table 6-35: Cancel

public System.Guid FileSvc.Cancel()	
Parameter	**Meaning**
Return value	GUID to identify the asynchronous operation.
Possible exceptions	FileSvcException, FileSvcBusyException

The asynchronous variant of this function calls the implementation of the following delegates (see Chapter 6.4.4 Specifications for implementing delegates):

- AsyncCanceled()

Table 6-36: CancelAsync

public System.Guid FileSvc.CancelAsync(AsyncCanceled cb)	
Parameter	**Meaning**
cb	Delegate implementation for the message if the asynchronous operation was canceled.
Return value	GUID to identify the asynchronous operation.
Possible exceptions	FileSvcException, FileSvcBusyException

Unrestricted © Siemens AG 2024 All Rights Reserved

Starting a block-by-block data transfer from the NC or PLC

This function starts the block-by-block data transfer from a specified NC or PLC source.

A GUID for the subsequent identification of the call is returned.

The function calls the implementation of the following delegates (see Chapter 6.4.4 Specifications for implementing delegates):

- AsyncCompleted()
- AsyncProgress()
- NewBlockTransferData()

Table 6-37: StartBlockTransferReadAsync

public System.Guid FileSvc.StartBlockTransferReadAsync(Node source, NewBlockTransferData onNewData, AsyncCompleted onCompleted) public System.Guid FileSvc.StartBlockTransferReadAsync(Node source, NewBlockTransferData onNewData, AsyncCompleted onCompleted, AsyncProgress onProgress)	
Parameter	**Meaning**
source	Data source from which the data is to be read. For example, //NC/MPF.DIR/TEST.MPF or /PLC/0A00301A for the DB301
onNewData	Delegate implementation for the provided data.
onCompleted	Delegate implementation for the message when the request completes.
onProgress	Delegate implementation for the progress message.
Return value	GUID to identify the asynchronous operation.
Possible exceptions	FileSvcException, FileSvcBusyException

Starting a block-by-block data transfer to the NC or PLC

This function starts the block-by-block data transfer to a specified NC or PLC destination.

A GUID for the subsequent identification of the call is returned.

The function calls the implementation of the following delegates (see Chapter 6.4.4 Specifications for implementing delegates):

- AsyncCompleted()
- ProvideBlockTransferData()

Table 6-38: StartBlockTransferWriteAsync

public System.Guid FileSvc.StartBlockTransferWriteAsync(Node destination, ProvideBlockTransferData onNewData, AsyncCompleted onCompleted)	
Parameter	**Meaning**
destination	Data destination in which data should be written. For example, //NC/MPF.DIR/TEST.MPF or /PLC/0A00301A for the DB301
onNewData	Delegate implementation for the data to be provided
onCompleted	Delegate implementation for the message when the request completes.
Return value	GUID to identify the asynchronous operation.
Possible exceptions	FileSvcException, FileSvcBusyException

© Siemens AG 2024 All Rights Reserved

Providing a data block for transfer

This function provides a data block for the transfer to a specified NC or PLC destination.

Table 6-39: WriteBlockTransferDataAsync

public void FileSvc.WriteBlockTransferDataAsync(Byte[ ] data, bool eof)	
Parameter	**Meaning**
data	Byte array with the data to be transferred to the data destination.
eof	Specifies whether the data transfer has completed (true) or whether additional data blocks will be transferred (false).
Possible exceptions	FileSvcException, FileSvcBusyException

Deleting an MDA buffer

This function deletes the MDA buffer.

Table 6-40: DeleteMdaBuffer

public void DeleteMdaBuffer(int ChannelNumber);	
Parameter	**Meaning**
ChannelNumber	Channel number
Possible exceptions	FileSvcException, FileSvcBusyException

Reading an MDA buffer

This function reads the MDA buffer.

Table 6-41: ReadMdaBuffer

public string ReadMdaBuffer(int ChannelNumber);	
Parameter	**Meaning**
ChannelNumber	Channel number
Return value	Content of the MDA buffer
Possible exceptions	FileSvcException, FileSvcBusyException

Writing an MDA buffer

This function writes the MDA buffer.

Table 6-42: WriteMdaBuffer

public void WriteMdaBuffer(int ChannelNumber, string Text);	
Parameter	**Meaning**
ChannelNumber	Channel number
Text	Content of the MDA buffer
Possible exceptions	FileSvcException, FileSvcBusyException

Unrestricted © Siemens AG 2024 All Rights Reserved

6.4.4 Specifications for implementing delegates

ListAsyncCompleted

This function is called when the ListAsync() function exits.

Table 6-43: ListAsyncCompleted – specification for the delegate implementation

public delegate void FileSvc.ListAsyncCompleted(System.Guid guid, Node[] nodes, FileSvcStatus status)	
Parameter	**Meaning**
guid	GUID, which was returned when calling ListAsync(). It is used to identify the call.
nodes	Array of node objects, content of the directory.
status	Status of the request, see Section: 6.8 Reference FileSvcStatus

AsyncProgress

This function is cyclically called for an asynchronously running request in order to communicate the progress of this request.

Table 6-44: AsyncProgress – specification for the delegate implementation

public delegate void FileSvc.AsyncProgress(System.Guid guid, int percentage)	
Parameter	**Meaning**
guid	GUID that was returned after calling the asynchronous function. It is used to identify the call.
percentage	Percentage value of how far the request has progressed.

AsyncCompleted

This function is called if the running asynchronous request has been completed.

Table 6-45: AsyncCompleted – specification for the delegate implementation

public delegate void FileSvc.AsyncCompleted(System.Guid guid, FileSvcStatus status)	
Parameter	**Meaning**
guid	GUID that was returned after calling the asynchronous function. It is used to identify the call.
status	Status of the request, see Section: 6.8 Reference FileSvcStatus

AsyncMultiFilesCompleted

This function is called when the running asynchronous request for copying or deleting several files has completed.

Table 6-46: AsyncMultiFilesCompleted – specification for the delegate implementation

public delegate void AsyncMultiFilesCompleted(Guid guid, FileSvcStatus[] status);	
Parameter	**Meaning**
guid	GUID that was returned after calling the asynchronous function. It is used to identify the call.
status	Status of the request, see Section: 6.8 Reference FileSvcStatus

© Siemens AG 2024 All Rights Reserved

AsyncCanceled

This function is called if a running request has been completely canceled.

Table 6-47: AsyncProgress – specification for the delegate implementation

public delegate void FileSvc.AsyncCanceled(System.Guid guid)	
Parameter	**Meaning**
guid	GUID, which was returned when calling ListAsync(). It is used to identify the call.

NewBlockTransferData

This function is called when the data that was requested with the StartBlockTransferReadAsync function has been made available.

Table 6-48: NewBlockTransferData – provision of the requested data

public delegate void FileSvc.NewBlockTransferData(System.Guid guid, Byte[ ] data)	
Parameter	**Meaning**
guid	GUID that was returned after calling the asynchronous function. It is used to identify the call.
data	Byte array with the data that has been read from the NC or PLC.

ProvideBlockTransferData

This function is called as request for the provision of data after a data transfer has been initiated with the StartBlockTransferWriteAsync function.

Table 6-49: ProvideBlockTransferData – request to provide data

public delegate void FileSvc.ProvideBlockTransferData(System.Guid guid, FileSvcStatus status)	
Parameter	**Meaning**
guid	GUID that was returned after calling the asynchronous function. It is used to identify the call.
status	Status of the last write request.

FileSvc in 1:N constellations

By default, file operations are performed on the NCU which Operate is viewing. In a 1:N constellation, a file operation can be performed on a specific NCU (an NCU other than the one displayed in Operate) in two ways:

1. Addition of keyword and NCU name to the file path.
   A file operation can be performed on a specific NCU by prefixing the file path with the keyword **NCU** and the name of the NCU.
   Examples:
   Part program path in NCU2 - **//NCU/NCU2/**NC/MPF.DIR

2. Create FileSvc object with NCU name.
   By specifying the name of an NCU (example: NCU2) when a FileSvc object is created (see FileSvc constructors), access to the specified NCU will always take place via this FileSvc instance. The file paths for the file operations do not have to be given prefixes.

Unrestricted © Siemens AG 2024 All Rights Reserved

Note

NCU names are listed in MMC.INI (see also Fig. 6.7).
Entry:
[GLOBAL]
NcddeMachineNames=*NCU1,NCU2*

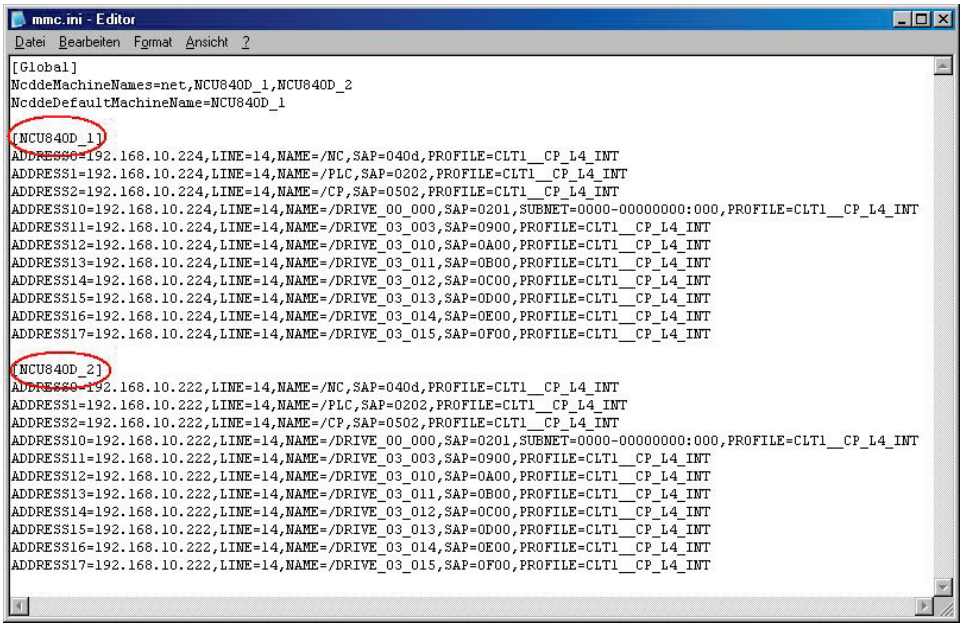


Fig. 6-7: NCU names for 1 : N

© Siemens AG 2024 All Rights Reserved

6.5 Reference ControlAccessRights

6.5.1 Definitions

Overview

A ControlAccessRights object contains the access authorizations in the sense of the SINUMERIK protection level concept of a file or a directory.
These authorizations are available for:

- Reading
- Writing
- Deleting
- Listing
- Executing

The class only contains properties for these 5 access authorizations and is defined in the "Siemens.Sinumerik.Operate.Services" namespace.

6.5.2 Properties

Enumerator AccessLevel

All of the properties described in the following supply an enumerator value, type AccessLevel. The possible values are described in the following table:

Table 6-50: Values of the AccessLevel enumerator

Value	Access level	Can be accessed with
AccessLevel.SIEMENS	S0	System password
AccessLevel.MANUFACT	S1	Machine manufacturer password
AccessLevel.SERVICE	S2	Service password
AccessLevel.USER	S3	End user password
AccessLevel.KEY_3	S4	Keyswitch position 3
AccessLevel.KEY_2	S5	Keyswitch position 2
AccessLevel.KEY_1	S6	Keyswitch position 1
AccessLevel.KEY_0	S7	Keyswitch position 0

Read

Access level to read a file or a directory.

Table 6-51:Read

public ControlAccesRights.AccessLevel ControlAccesRights.Read { set; get; }

Write

Access level for writing to a file or a directory.

Table 6-52:Write

public ControlAccesRights.AccessLevel ControlAccesRights.Write { set; get; }

Delete

Access level to delete a file or a directory.

Table 6-53:Delete

public ControlAccesRights.AccessLevel ControlAccesRights.Delete { set; get; }

Unrestricted © Siemens AG 2024 All Rights Reserved

List

Access level to list (display in the file browser) a file or a directory.

Table 6-54:List

public ControlAccesRights.AccessLevel ControlAccesRights.List { set; get; }

Execute

Access level to execute a file. To execute means selecting a part program for processing.

Table 6-55:Execute

public ControlAccesRights.AccessLevel ControlAccesRights.Execute { set; get; }

© Siemens AG 2024 All Rights Reserved

6.6 Reference NodeAttributes

6.6.1 Definitions

Overview

NodeAttributes objects are used to access additional information of a file or a directory. They are part of node objects (Property: Attributes). The class only contains properties and is defined in the "Siemens.Sinumerik.Operate.Services" namespace.

6.6.2 Properties

LastAccess

Time stamp of the last access to the file or the directory.

Table 6-56:LastAccess

public System.DateTime NodeAttributes.LastAccess { set; get; }

Size

File size. For a directory, the value is -1.

Table 6-57:Size

public int NodeAttributes.Size { set; get; }

Unrestricted © Siemens AG 2024 All Rights Reserved

6.7 Reference device

6.7.1 Definitions

Overview

A device object represents a connected device. For instance, this can be a USB mass memory or a connected network drive.
Further, the class offers functions so that a notification is issued if a new device is connected or a device is removed.
The class is defined in the "Siemens.Sinumerik.Operate.Services" namespace.

6.7.2 Creating a device object

The following constructor is available to create a device object.

Table 6-58: Constructors of the Device class

public Device(string deviceName)	
Parameter	**Meaning**
deviceName	Name of the drive, e.g. "//USB:/" or "//NET:" (The symbolic name 'USB' or 'NET' is assigned during the setup of the drive in the ‘commissioning/HMI/Log.drive' operating area

6.7.3 Notification for drives

Setting up a notification for drives (Subscribe)

Setting up a notification if drives are inserted/connected or withdrawn. The call returns as soon as the setup request has been forwarded to the server. The notification is issued by calling the implementations of the DeviceMounted() and DeviceUnmounted() delegates.

Table 6-59: Subscribe

public void Device.Subscribe(DeviceMounted mt, DeviceUnmounted un)	
Parameter	**Meaning**
mt	Delegate implementation for the message - drive inserted
un	Delegate implementation for the message - drive withdrawn

Table 6-60: DeviceMounted – specification for the delegate implementation

public delegate void Device.DeviceMounted(string devicePath)	
Parameter	**Meaning**
devicePath	Path of the new drive

Table 6-61: DeviceUnmounted – specification for the delegate implementation

public delegate void Device.DeviceUnmounted(string devicePath)	
Parameter	**Meaning**
devicePath	Path of the withdrawn drive

Unsubscribing a notification for drives (UnSubscribe)

Unsubscribing a notification, if drives are inserted/connected or withdrawn, is realized using the UnSubscribe function.

Table 6-62: UnSubscribe

public void Device.UnSubscribe()	
Parameter	**Meaning**
-	-

© Siemens AG 2024 All Rights Reserved

6.7.4 Properties

DeviceName

Name of the drive.

Table 6-63:DeviceName

public string Device.DeviceName { get; }

DeviceType

Drive type. The type can assume the following values:

Table 6-64: Values of the WinDeviceType enumerator

Value	Description
ALL	Any type
CDROM	CD drive
FIXED	Hard disk drive
RAMDISK	Drive, that was set up as ram disk
REMOTE	Network drive
REMOVABLE	USB drive

Table 6-65:DeviceType

public WinDeviceType Device.DeviceType { get; }

FreeSpace

Available drive memory.

Table 6-66:FreeSpace

public long Device.FreeSpace { get; }

UsedSpace

Used drive memory space.

Table 6-67:UsedSpace

public long Device.UsedSpace { get; }

Mounted

Drive is available.

Table 6-68:Mounted

public bool Device.Mounted { get; }

Unrestricted © Siemens AG 2024 All Rights Reserved

6.8 Reference FileSvcStatus

6.8.1 Definitions

Overview

A FileSvcStatus object contains the result of a completed asynchronous request.
The object only contains properties.

6.8.2 Properties

Ok

For a completed asynchronous request, indicates whether an error has occurred.

Values:
True: No problems
False: A problem has occurred. A more precise error evaluation can be made using the "ErrorNo" property.

Table 6-69:Ok

public bool Ok { set; get; }

ErrorNo

Error number in the case of a problem for completed asynchronous requests.

Table 6-70:ErrorNo

public int ErrorNo { set; get; }

© Siemens AG 2024 All Rights Reserved

6.9 FAQs

6.9.1 FAQs regarding "FileSvc"

How do I create a directory?

To start, a node object is created, which points to the directory to be generated. Further, the IsDirNode property is set to true. If the Create() function of FileSvc is now called, and this Node object is transferred, a directory is generated.

```
FileSvc slFile = new FileSvc();
Node node = new Node("C:/tmp/Test");
node.IsDirNode = true ;
slFile.Create(node, true);
```

How many asynchronous requests can I "simultaneously" initiate on a FileSvc object?

One. If an attempt is made to issue a second asynchronous call, then although a request is already running, a FileSvcException is initiated.

Unrestricted © Siemens AG 2024 All Rights Reserved

© Siemens AG 2024 All Rights Reserved

7

7 Infrastructure services

Objective of the chapter

This chapter describes the interface of the infrastructure service. This can be used in order to access global data such as the current resolution, current language, current channel, current operating area, current NCU, or current access level. The connection status can also be determined. It is also possible to obtain a notification if one of these values has changed.

Unrestricted © Siemens AG 2024 All Rights Reserved

7.1 Introduction

7.1.1 Class model

Overview

The class model of the InfrastructureService primarily comprises one class:

- Infrastructure

Namespace

The Infrastructure object is defined in the "Siemens.Sinumerik.Operate.Services" namespace. As a consequence, this namespace can be integrated into the C# project at the start of the particular file:

```
using Siemens.Sinumerik.Operate.Services;
```

Class Infrastructure

The infrastructure class is implemented as static class, which means that an object for this class does not have to be created. Functions and properties can be directly called or queried.

7.1.2 Explanation of terms

Completeness of signaled value changes

Setting up a hotlink (subscribe calls) does not guarantee that all changes are signaled. The reasons for this are as follows:

1. The values are read cyclically by the NCK. Value changes continue to be signaled if the read value differs from the previously signaled value. Value changes are not detected by this mechanism if a value changes and then changes back to the starting value between two consecutive sampling points.
2. The sampling times do not have to be isochronous with respect to time. NCK/PLC may suppress the continued signaling of value changes if the real-time area would otherwise be disturbed or if the communication connection is overloaded.
3. An attempt is made to combine the largest possible number of values in one communication protocol unit (PDU) in order to reduce the data volume for the applications.
4. Changes can be suppressed if a processing backup is detected on the client side.

In summary, the following is applicable:
The InfrastructureService is not in a position to detect all value changes of a variable. In addition, the required sampling rates are not guaranteed; rather, they should be viewed as a target specification. However, the InfrastructureService will attempt to continue sending the client all value changes it has detected provided the response capability of the client is sufficient. In this case, the client has no real-time requirements, but it must prevent an ever-increasing backup from occurring. The InfrastructureService guarantees that the final value of a series of value changes is always signaled after a certain time delay.

© Siemens AG 2024 All Rights Reserved

7.2 Step-by-step examples

Overview

The following chapters show the various applications for the InfrastructureService step-by-step. An executable example is included in the SINUMERIK Integrate Create MyHMI /3GL for each "step-by-step example".
These and all other examples are shown in the following overview of examples.

Table 7-1: Overview of the examples

Application	Example	Chapter
Notification of a change to the current language. Read out and switch the current language.	Infrastructure	**7**.2.2
Notification about the change to the current resolution. Read out and switch the current resolution.	Infrastructure	-
Notification about a change to the current access level. Read out the current access level.	Infrastructure	-
Notification about the connection status to the NC/PLC and the status of SINUMERIK Operate.	Infrastructure	-
Read and set the current operating area, the current channel and currently selected NCU, and shut down the HMI Operate software.	InfraHMI	7.2.3

Only the items relevant for the InfrastructureService are explained or described in each example. The comments for the sources contain more detailed information (e.g. screen layout, outputs, etc.).

7.2.1 Preparation

Overview

Preparatory measures are described in Chapter 2.1 ➔ "New Project". All of the items described there must be executed first.

7.2.2 Current language - read, changeover, notify for change

Overview

The "infrastructure" example for the infrastructure interface includes the monitoring and the setting of the selected language, the screen resolution and the access level. The connection status to the NC/PLC and status of SINUMERIK Operate can also be monitored. However, only the part of the current language is shown step-by-step.

Using the "start subscription" button, an automatic notification for changes to the current language is set up. Click "stop subscription" to end this mechanism. The current language is explicitly read-out using the "get actual values" button. "set language deu" and "set language eng" toggle the current language to German or to English.

Unrestricted © Siemens AG 2024 All Rights Reserved

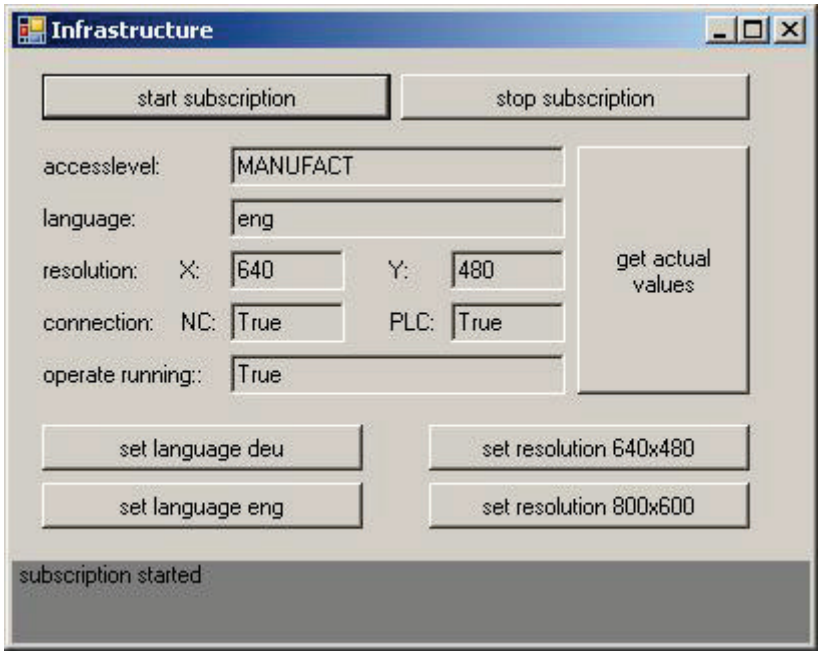


Fig. 7-1: Example of the infrastructure class

Note

As all functions and properties of the Infrastructure object have been statically implemented, it is not necessary to create an object of this class.

Step 1

Read out the current language.

```
// get language
txtLanguage.Text = Infrastructure.CurrentLanguage;
```

Step 2

Set the current language. In this case, the display language changes to German.

```
// set German language
Infrastructure.CurrentLanguage = "deu";
```

Step 3

Setting up the notification regarding the change to the current language. If the language is changed over, then the CBLanguageChanged() function should be called.

```
// subscribe callback to LanguageChanged
Infrastructure.SubscribeLanguageChanged(CBLanguageChanged);
```

Step 4

Implement the delegate.

Important

Because the delegate is called from another thread, not all display elements can be accessed. Consequently, it is desirable to call a second delegate with the same signature with which changes can be performed on the user interface.

© Siemens AG 2024 All Rights Reserved

In the implemented delegate function, a change is sent to the same delegate signature, now in the GUI thread. The currently selected language is now displayed here. The current language is transferred in the "language" call parameter.

```
private void CBLanguageChanged(System.String language)
{
    // are we in GUI thread (main) ?
    // No we are not in GUI thread => send message to GUI thread
    if (txtLanguage.InvokeRequired)
    {
        LanguageChanged myDelegate = new LanguageChanged(CBLanguageChanged);
        BeginInvoke(myDelegate, new Object[] { language });
    }
    else
    {
        // set the language to the textbox
        txtLanguage.Text = language;
    }
}
```

Step 5

Unsubscribe the notification. In this case, the implemented function of the delegate must be transferred again.

```
// unsubscribe callback LanguageChanged
Infrastructure.UnSubscribeLanguageChanged(CBLanguageChanged);
```

7.2.3 Operating area selection, NCU and channel change

Overview

The "InfraHMI" example for the infrastructure interface includes the monitoring of the current channel, the exporting and setting of the selected operating area, the exporting and setting of the current NCU, and the shutdown of the HMI Operate software.

Click the "get active area" button to export the name of the currently selected operating area. Click the "subscribe channel" button to setup the automatic notification for a channel change. Click the "unsubscribe channel" button to end this mechanism. Click the "get active NCU" button to export the active NCU, and "set active NCU" to set the active NCU. Click the "switch to area" button to change the operating area, and the "shutdown HMI" button to exit the HMI Operate.

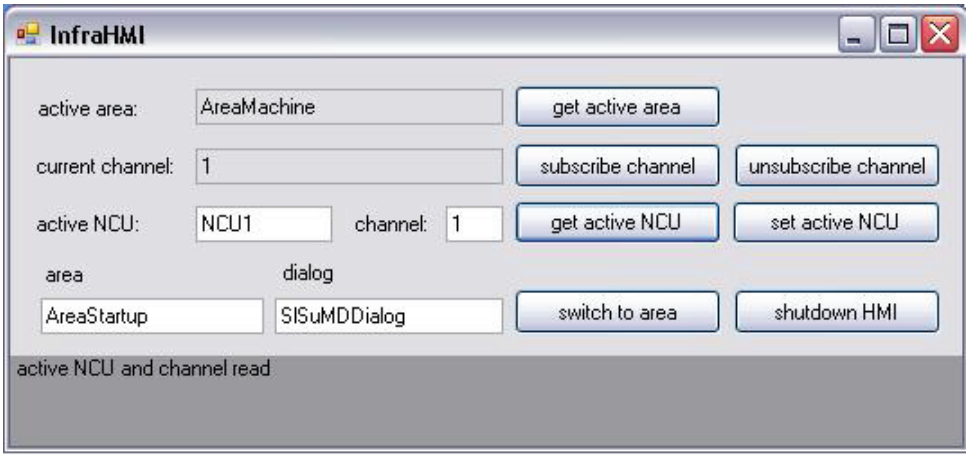


Fig. 7-2: Second example of the Infrastructure class

Unrestricted © Siemens AG 2024 All Rights Reserved

Note

Because all functions and properties of the Infrastructure object have been implemented statically, it is not necessary to create an object of this class.

Step 1

Export the name of the currently selected operating area.

```
// get active area
txtActiveArea.Text = Infrastructure.ActiveArea;
```

Step 2

Setup the notification regarding the change of the current channel.
If the channel is changed, the OnChannelChanged() function should be called.

```
// subscribe callback to ChannelNoChanged
Infrastructure.SubscribeChannelNoChanged(CBChannelChanged);
```

Step 3

Implement the delegate.

Important

Because the delegate is called from another thread, not all display elements can be accessed. Consequently, it is desirable to call a second delegate with the same signature with which changes can be performed on the user interface.

In the implemented delegate function, a change is sent to the same delegate signature, now in the GUI thread. The currently selected channel is now displayed. The current channel is transferred in the "channelno" call parameter.

```
private void CBChannelChanged(int channelno)
{
    // are we in GUI thread (main) ?
    // No we are not in GUI thread => send message to GUI thread
    if (txtCurrentChannel.InvokeRequired)
    {
        ChannelNumberChanged channelChanged = new
                                    ChannelNumberChanged(CBChannelChanged);
        BeginInvoke(channelChanged, new Object[] { channelno });
    }
    else
    {
        // set the channel number to the textbox
        txtCurrentChannel.Text = channelno.ToString();
    }
}
```

Step 4

Unsubscribe the notification. In this case, the implemented function of the delegate must be transferred again.

```
// unsubscribe callback ChannelChanged
Infrastructure.UnSubscribeChannelNoChanged(CBChannelChanged);
```


© Siemens AG 2024 All Rights Reserved

Step 5

Determine the name of the currently selected NCU and the channel set there.

```
// get active NCU and channel
txtNCU.Text = Infrastructure.ActiveNCU;
txtChannel.Text = Infrastructure.ChannelNumber.ToString();
```

Step 6

Select an NCU and a channel.

```
// set NCU and channel
Infrastructure.SetActiveNcu(txtNCU.Text, Convert.ToInt32(txtChannel.Text));
```

Step 7

Perform an area change with a specified area name and dialog name.

```
// switch to area
Infrastructure.SwitchToArea(txtAreaName.Text, txtDialogName.Text);
```

Step 8

Exit HMI operate.

```
// shutdown HMI
Infrastructure.ShutdownHmi(false);
```

Unrestricted © Siemens AG 2024 All Rights Reserved

7.3 Reference infrastructure

7.3.1 Definitions

Overview

The infrastructure class can be used to access global data such as the current resolution, current language, current channel, or current access level. The connection status can also be determined. Further, a message is received if one of these values has changed. This is realized by calling the implementation of the particular delegates.

Access levels

There are eight different access levels in the control.

Table 7-2: The eight access authorization levels

Access level	Can be accessed with	User group
S0	System password	SIEMENS
S1	Machine manufacturer password	Machine manufacturer
S2	Service password	Commissioning engineer, service (machine manufacturer)
S3	End user password	Privileged end user (in-house service)
S4	Keyswitch position 3	Programmers
S5	Keyswitch position 2	Trained operator
S6	Keyswitch position 1	Operator
S7	Keyswitch position 0	Semi-skilled operator (NC Start / NC Stop, MCP)

S0 denotes the access level with the highest rights and S7 the access level with the most restrictive rights. The keyswitch position is enabled by the current keyswitch position or by entry of a password.

7.3.2 Notifications

Overview

Various notifications can be activated using the subsequently described Subscribe functions. To call the particular function, an instance of the infrastructure class does not have to be created, as the class was statically implemented.

Example:

```
Infrastructure.SubscribeAccessLevelChanged(OnAccessLevelChanged);
```

Language selection (SubscribeLanguageChanged)

Activate a notification for the language selection. If the language at the HMI is changed, the implementation of the "LanguageChanged" delegate is called.

© Siemens AG 2024 All Rights Reserved

Table 7-3: SubscribeLanguageChanged

public static void Infrastucture.SubscribeLanguageChanged(LanguageChanged cb)	
Parameter	**Meaning**
cb	Delegate implementation for the message of the language selection change
Possible exceptions	InfrastructureException

Table 7-4: LanguageChanged – specification for the delegate implementation

public delegate void Infrastucture.LanguageChanged(string language)	
Parameter	**Meaning**
language	Currently set language

Example: See Chapter 7.2.2

Changing the access level (SubscribeAccessLevelChanged)

Activate a notification for a change to the access level. If the access level of the control is changed, the implementation of the "AccessLevelChanged" delegate is called.

Table 7-5: SubscribeAccessLevelChanged

public static void Infrastucture.SubscribeAccessLevelChanged(AccessLevelChanged cb)	
Parameter	**Meaning**
cb	Delegate implementation for the message of the access level change
Possible exceptions	InfrastructureException

Table 7-6: AccessLevelChanged – specification for the delegate implementation

public delegate void Infrastucture.AccessLevelChanged(AccessLevel level)	
Parameter	**Meaning**
level	Currently set access level

Resolution changeover (SubscribeResolutionChanged)

Activate a notification for a change to the resolution. If the resolution at the HMI is changed, the implementation of the "ResolutionChanged" delegate is called.

Table 7-7: SubscribeResolutionChanged

public static void Infrastucture.SubscribeResolutionChanged(ResolutionChanged cb)	
Parameter	**Meaning**
cb	Delegate implementation for the message of the resolution change
Possible exceptions	InfrastructureException

Table 7-8: ResolutionChanged – specification for the delegate implementation

public delegate void Infrastucture.ResolutionChanged(ScreenResolution resolution)	
Parameter	**Meaning**
resolution	Current resolution

Unrestricted © Siemens AG 2024 All Rights Reserved

Changing the current channel number (SubscribeChannelNoChanged)

Activate a notification to change the channel number. If another channel is selected on the HMI, the implementation of the "ChannelNumberChanged" delegate is called.

Table 7-9: SubscribeChannelNoChanged

public static void Infrastructure.SubscribeChannelNoChanged(ChannelNumberChanged cb)	
Parameter	**Meaning**
cb	Delegate implementation for reporting the change of the set channel.
Possible exceptions	InfrastructureException

Table 7-10: ChannelNumberChanged – specification for the delegate implementation

public delegate void Infrastucture.ChannelNumberChanged(int channelNo)	
Parameter	**Meaning**
channelNo	New channel number.

Changing the connection status (SubscribeConnectionState)

Activate a notification for a change of the connection status to the NC/PLC. If this changes, the implementation of the delegate "ControlConnectionStateChanged" is called.

Table 7-11: SubscribeConnectionState

public static void Infrastructure.SubscribeConnectionState(ControlConnectionStateChanged cb);	
Parameter	**Meaning**
cb	Delegate implementation for reporting the change of the connection status to the NC/PLC.
Possible exceptions	InfrastructureException

Table 7-12: ControlConnectionStateChanged – specification for the delegate implementation

public delegate void Infrastucture. ControlConnectionStateChanged(bool ncuConnected, bool plcConnected)	
Parameter	**Meaning**
ncuConnected	True ➔ Connected to the NC
plcConnected	True ➔ Connected to the PLC

Status change of SINUMERIK Operate (SubscribeOperateStateChanged)

Activate a notification for a status change of SINUMERIK Operate. If this changes, the implementation of the delegate "OperateStateChanged" is called.

Table 7-13: SubscribeOperateStateChanged

public static void Infrastructure.SubscribeOperateStateChanged(OperateStateChanged cb);	
Parameter	**Meaning**
cb	Delegate implementation for reporting the change of the SINUMERIK Operate status
Possible exceptions	InfrastructureException

Table 7-14: OperateStateChanged – specification for the delegate implementation

public delegate void Infrastucture.OperateStateChanged(bool active)	
Parameter	**Meaning**
active	True ➔ SINUMERIK Operate is running

© Siemens AG 2024 All Rights Reserved

7.3.3 Unsubscribing notifications

Language changeover (UnSubscribeLanguageChanged)

Unsubscribing the notification for the language changeover.

Table 7-15: UnSubscribeLanguageChanged

public static void Infrastucture.UnSubscribeLanguageChanged(LanguageChanged cb)	
Parameter	**Meaning**
cb	Delegate implementation for the message of the language selection change
Possible exceptions	InfrastructureException

Changing the access level (UnSubscribeAccessLevelChanged)

Unsubscribing the notification for the access level change.

Table 7-16: UnSubscribeAccessLevelChanged

public static void Infrastucture.UnSubscribeAccessLevelChanged(AccessLevelChanged cb)	
Parameter	**Meaning**
cb	Delegate implementation for the message of the access level change
Possible exceptions	InfrastructureException

Resolution changeover (UnSubscribeResolutionChanged)

Unsubscribing the notification for the resolution change.

Table 7-17: UnSubscribeResolutionChanged

public static void Infrastucture.UnSubscribeResolutionChanged(ResolutionChanged cb)	
Parameter	**Meaning**
cb	Delegate implementation for the message of the resolution change
Possible exceptions	InfrastructureException

Changing the current channel number (UnsubscribeChannelNoChanged)

Unsubscribe the notification for the change of the channel number.

Table 7-18: UnSubscribeChannelNoChanged

public static void Infrastucture.UnSubscribeChannelNoChanged(ChannelNumberChanged cb)	
Parameter	**Meaning**
cb	Delegate implementation for reporting the change of the set channel.

Changing the connection status (UnSubscribeConnectionState)

Unsubscribe the notification of the connection status to the NC/PLC.

Table 7-19: UnSubscribeConnectionState

public static void Infrastucture.UnSubscribeConnectionState(ControlConnectionStateChanged cb)	
Parameter	**Meaning**
cb	Delegate implementation for reporting the change of the connection status to the NC/PLC.

Unrestricted © Siemens AG 2024 All Rights Reserved

Status change of SINUMERIK Operate (UnSubscribeOperateStateChanged)

Unsubscribe the notification for a status change of SINUMERIK Operate.

Table 7-20: UnSubscribeOperateStateChanged

public static void Infrastucture.UnSubscribeOperateStateChanged(OperateStateChanged cb)	
Parameter	**Meaning**
cb	Delegate implementation for reporting the change of the SINUMERIK Operate status

7.3.4 Properties of the class infrastructure

Overview

With the properties subsequently described, at any instant in time, the currently set language, access level, resolution, NCU, operating area, or channel can be accessed. The connection status to the NC/PLC and status of SINUMERIK Operate can also be queried. As the infrastructure class is static, an object does not have to be created, it is possible to directly access the properties:

Example:

```
string lng = Infrastructure.CurrentLanguage;
```

Current language

Read out / set the current language.

Table 7-21:CurrentLanguage

public static string Infrastucture.CurrentLanguage { set; get; }

Current access level

Read out the current access level.

Table 7-22:AccessLevel

public static AccessLevel Infrastucture.AccessLevel { get; }

Current resolution

Read out / set the current resolution.

When setting the resolution, the following parameter must be set in systemconfiguration.ini:

```
[miscellaneous]
adaptHmiResolutionToActiveUnit=false
```

Table 7-23:CurrentResolutionX

public static ScreenResolution Infrastucture.CurrentResolution { set; get; }

Current channel

Read out / set the current channel.

Table 7-24:ChannelNumber

public static int Infrastucture.ChannelNumber { set; get; }

© Siemens AG 2024 All Rights Reserved

Current NCU

Read out / set the currently selected NCU. Writing the properties triggers a switch to the associated NCU in channel 1.

Table 7-25:ActiveNcu

public static string ActiveNcu{ set; get; }

Current operating area

Read out the currently selected operating area of the HMI.

Table 7-26:ActiveArea

public static string Infrastucture.ActiveArea { get; }

NC connection status

Read out the connection status to the NC.

Table 7-27: ConnectionStateNcu

public static bool ConnectionStateNcu { get; }

PLC connection status

Read out the connection status to the PLC.

Table 7-28: ConnectionStatePlc

public static bool ConnectionStatePlc { get; }

SINUMERIK Operate status

Read out the status of SINUMERIK Operate.

Table 7-29: OperateRunning

public static bool OperateRunning { get; }

7.3.5 Setting the active NCU

Set the active NCU for an m:n configuration.

Table 7-30: SetActiveNcu

public static void SetActiveNcu(string ncu, int channel)	
Parameter	**Meaning**
ncu	Name of the NCU from the MMC.INI file.
channel	Channel number that should be selected.
Possible exceptions	InfrastructureException

Unrestricted © Siemens AG 2024 All Rights Reserved

7.3.6 Selecting the operating area

The SwitchToArea function permits the selection of a specific operating area and, optionally, a dialog.

Table 7-31: SwitchToArea

public static void SwitchToArea(string areaName) **public static void SwitchToArea(string areaName, string dialogName)** **public static void SwitchToArea(string areaName, string dialogName, string args)**	
Parameter	**Meaning**
areaName	Name of the operating area.
dialogName	Name of the dialog.
args	Additional parameters for the dialog.
Possible exceptions	InfrastructureException

7.3.7 Setting the active channel

Set the active channel.

Table 7-32: SelectChannel

public static void SelectChannel(int channel)	
Parameter	**Meaning**
channel	Channel number
Possible exceptions	InfrastructureException

7.3.8 Shutting down the HMI

The ShutdownHmi function can be used to shut down the HMI Operate software.

Table 7-33: ShutdownHmi

public static void ShutdownHmi(bool force)	
Parameter	**Meaning**
force	The **true** value forces the shutdown.
Possible exceptions	InfrastructureException

7.3.9 Is HMI Operate active?

The IsHmiRunning function returns a Boolean value that indicates whether or not the HMI Operate software is active.

Table 7-34: IsHMIRunning

public static bool IsHmiRunning()	
Parameter	**Meaning**
Possible exceptions	InfrastructureException

© Siemens AG 2024 All Rights Reserved

8

8 Archive service

Objective of the chapter

This chapter describes the interface of the archive service. The archive service allows different archive types to be generated and read-in. The following types are supported:

SINUMERIK 840D sl:

- Series commissioning archive
- User archive
- PLC upgrade archive
- Paper-tape archive

SINUMERIK ONE / MCU1720:

- Series commissioning archive
- Backup archive

Furthermore, it is possible to monitor the archive service functions. For example, you can specifically respond to a series commissioning (standard operating area "Commissioning").

Unrestricted © Siemens AG 2024 All Rights Reserved

8.1 Introduction

8.1.1 Class model

Overview

The class model of the archive service primarily consists of the following classes:

- ArchiveSvc
- ArchiveSvcNotifier

Namespace

All classes are defined in the "Siemens.Sinumerik.Operate.Services" namespace. As a consequence, this namespace can be integrated into the C# project at the start of the particular file:

```
using Siemens.Sinumerik.Operate.Services;
```

Class ArchiveSvc

You can generate and read-in various archive types using ArchiveSvc objects. When reading in, the contents of the archive are unzipped in the appropriate paths.

It is only possible to asynchronously generate and read-in the archive as a result of the long execution time required. In this case, the ArchiveSvc object calls the implementation of a delegate when a request has been completed.

Class ArchiveSvcNotifier

Using ArchiveSvcNotifier objects, archive service functions that are being executed can be monitored so that you can specifically respond to a series commissioning in the standard operating area "Commissioning".

8.1.2 Explanation of terms

Synchronous calls

Synchronous calls return only after the request is performed, i.e. the calling thread is blocked in the meantime. This can interfere with event processing since control inputs and displays are withheld during a synchronous call. For this reason, calls that potentially may take a long time should be performed asynchronously.

Asynchronous calls

Asynchronous calls return as soon as the request has been transferred to the archive service. The returned error code cannot therefore indicate whether the request has been completed successfully; it can only indicate whether or not the request has been sent successfully. For example, an error occurs if call parameters are not correctly assigned. The actual request status is supplied in the callbacks of the archive service (call of the implemented delegate).

© Siemens AG 2024 All Rights Reserved

Archive types (SINUMERIK 840D sl)

a) **Series commissioning archive** (setup)
A series commissioning archive should facilitate:
- After a first commissioning, bringing a further control on the same type of machine into the same state as after first commissioning.
- Putting a new control into the original state as easily as possible in case of service (hardware replacement).

b) **User archive** (user)
A user archive can comprise any files (e.g. part programs, workpieces, etc.). It is used to archive individual files from the NC memory and the local drive.

c) **PLC upgrade archive** (PlcUpgrade)
A PLC upgrade archive contains only the SDBs of the PLC and is used to upgrade the PLC.

d) **Paper-tape archive** (paper tape)
Files in the ASCII format can be archived using a paper-tape archive. It should be noted that:
- Only files with characters that can be displayed, i.e. files created in the text editor, can be saved. No binary data, however, can be saved.
- Paper-tape archives can be edited or created with a text editor.

Note

A description of the paper-tape format can be found on DOConCD in the "Operating Manual HMI-Embedded" in the "Operator area Services" chapter (keyword: paper-tape format).

Archive types (SINUMERIK ONE)

a) **Series commissioning archive** (DSF_Setup)
- After first commissioning, this archive type allows an additional control on the same type of machine to be brought into the same state as after first commissioning.
- No individual data are archived, and when reading in, the target data are first cleared.

b) **Backup archive** (DSF_Backup)
- This archive type is intended to bring a new control into the original state with the lowest possible associated time and costs in the case of service (hardware replacement).
- This involves data backup for a specific target system.

Access rights

Access level S1 (machine manufacturer) must be set in order to use the interface functions of the archive service. For example, this can be directly realized in the code by calling the PI services "_N_LOGIN_" or "_N_LOGOUT".

Unrestricted © Siemens AG 2024 All Rights Reserved

8.2 Step-by-step examples

Overview

The following chapters show the various applications of the archive service step-by-step. An executable example is included in the SINUMERIK Integrate Create MyHMI /3GL for each "step-by-step example".

Table 8-1: Overview of the examples

Application	Example	Chapter
Creating a series commissioning archive	ArchiveSvcCreateRead	8.2.2
Creating a user archive	ArchiveSvcCreateRead	8.2.3
Read-in an archive	ArchiveSvcCreateRead	8.2.4
Reading out additional data of an archive	ArchiveSvcCreateRead	-
Monitoring the functions of the archive service monitor	ArchiveSvcNotify	8.2.5

Only the items relevant for the ArchiveService are explained or described in each example. The comments for the sources contain more detailed information (e.g. screen layout, outputs, etc.).

8.2.1 Preparation

Overview

Preparatory measures are described in Chapter 2.1 ➔ "New Project". All of the items described there must be executed first.

8.2.2 Creating a series commissioning archive

The following step-by-step example shows which actions are necessary in order to create a series commissioning archive. The matching executable example in the SINUMERIK Integrate Create MyHMI /3GL is the example "ArchiveSvcCreateRead". The functionality under the "Create Setup Archive" tab under the "create setup archive" button is described in this chapter.

In this example, under "archive type", you can select what should be archived. The storage location of the archive can be obtained from the "archive path" and the "archive name". The progress of the asynchronous call can be tracked under "archive protocol". The status bar indicates whether the call has been successful.

The example makes a distinction as to whether a SINUMERIK 840D sl or a SINUMERIK ONE is connected.

© Siemens AG 2024 All Rights Reserved

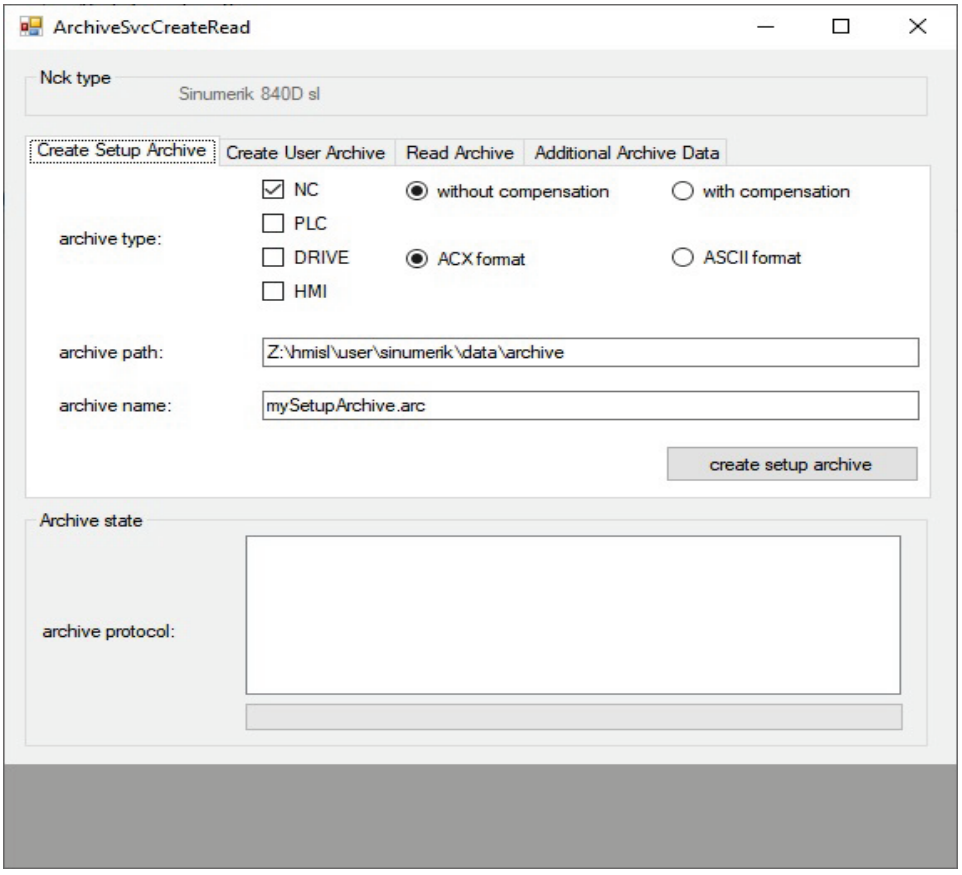


Fig. 8-1: Example, ArchiveSvcCreateRead - creating a series commissioning archive

Step 1

Create the required member variables.

```
ArchiveSvc m_Archive = null;
Guid m_gdActivGuid = Guid.Empty;
string m_strLastListEntry = string.Empty;
```

Step 2

Create the ArchiveSvc object. This is realized in the ArchiveSvcCreateRead function:

```
m_Archive = new ArchiveSvc();
```

Step 3

To create a series commissioning archive, a "SetupArchiveComponent" array type is required. This array contains the components to be backed up. In the example, a list of such components is the first created, depending on which selection boxes have been clicked. A selection box is shown here as example:

```
List<SetupArchiveComponent> components = new List<SetupArchiveComponent>();

//PLC
if (chkSetupPLC.Checked)
{
    components.Add(SetupArchiveComponent.Plc);
}
```

Unrestricted © Siemens AG 2024 All Rights Reserved

Step 4 - SINUMERIK 840D sl

Archiving is started using the function call CreateSetupArchiveAsync(). The following data is transferred as transfer parameter:

- Path and name of the archive to be created.
- Components that should be backed up.
- Additional files (here "zero" is transferred, because no additional files are to be saved).
- Name of the creator
- Comment
- Delegate for completion
- Delegate for progress

```
m_gdActivGuid = m_Archive.CreateSetupArchiveAsync(
                      txtSetupArchivePath.Text + txtSetupArchiveName.Text,
                      components.ToArray(),
                      null,
                      "CreatorName",
                      "This is a comment",
                      onArchiveSvcActionCompleted,
                      onArchiveSvcActionProgress);
```

Step 4 - SINUMERIK ONE

Archiving is started with function call CreateArchiveAsync(). The following data is transferred as transfer parameter:

- Path and name of the archive to be created.
- Archive type (DSF_Setup)
- Components that should be backed up.
- Name of the creator
- Comment
- Delegate for completion
- Delegate for progress

```
m_gdActivGuid = m_Archive.CreateArchiveAsync(
                      txtSetupArchivePath.Text + txtSetupArchiveName.Text,
                      ArchiveType.DSF_Setup,
                      components.ToArray(),
                      "CreatorName",
                      "This is a comment",
                      onArchiveSvcActionCompleted,
                      onArchiveSvcActionProgress);
```

Step 5

Implement the delegates.

! Important

Because the delegate is called from another thread, not all display elements can be accessed. Consequently, it is desirable to call a second delegate with the same signature with which changes can be performed on the user interface.

In the implemented delegate function, a change is sent to the same delegate signature, now in the GUI thread. The actual progress and the end of archiving is now displayed here. As far as the progress is concerned, it is possible that a file name is frequently transferred each time with a different percentage. This is the reason why the last file name is saved in a member variable in the example. Before the file name is output, a check is made as to whether it differs from the last one saved.

© Siemens AG 2024 All Rights Reserved

```
private void onArchiveSvcActionProgress(Guid guid,
                                        int percentage,
                                        string archiveItem)
{
    // are we in GUI thread (main) ?
    // No we are not in GUI thread => send message to GUI thread
    if (prgProgress.InvokeRequired)
    {
        ArchiveSvcActionProgress internalDelegate = new
                    ArchiveSvcActionProgress(onArchiveSvcActionProgress);
        BeginInvoke(internalDelegate, new Object[] { guid, percentage,
                                                     archiveItem });
    }
    else
    {
        if (guid == m_gdActivGuid)
        {
            //Avoid empty and duplicate entries
            if (("" != archiveItem) && (archiveItem != m_strLastListEntry))
            {
                lstArchiveProtocol.Items.Add(archiveItem);
                m_strLastListEntry = archiveItem;
            }

            //actualize progress bar
            prgProgress.Value = percentage;
        }
    }
}
```

```
private void onArchiveSvcActionCompleted(System.Guid guid,
                                         ArchiveSvcStatus status)
{
    // are we in GUI thread (main) ?
    // No we are not in GUI thread => send message to GUI thread
    if (prgProgress.InvokeRequired)
    {
        ArchiveSvcActionCompleted internalDelegate = new
                    ArchiveSvcActionCompleted(onArchiveSvcActionCompleted);
        BeginInvoke(internalDelegate, new Object[] { guid, status });
    }
    else
    {
        if (guid == m_gdActivGuid)
        {
            //check error
            if (false == status.Ok)
            {
                //and display message
                setStatus("onArchiveSvcActionCompleted(): Error in Completed
                                                           Delegate !!!");
            }
            else
            {
                //Write status text
                setStatus("Archive done!");
            }
        }
    }
}
```

Unrestricted © Siemens AG 2024 All Rights Reserved

8.2.3 Creating a user archive

The following step-by-step example shows which actions are necessary in order to create a user archive. The matching executable example in the SINUMERIK Integrate Create MyHMI /3GL is the example "ArchiveSvcCreateRead". The functionality under the "Create User Archive" tab under the "create user archive" button is described in this chapter.

In this example, under "data list" you can select which part programs are to be archived. The storage location of the archive can be obtained from the "archive path" and the "archive name". The progress of the asynchronous call can be tracked under "archive protocol". The status bar indicates whether the call has been successful.

When activating the "Create User Archive" tab via the FileSvc interface, all part programs in the MPF.DIR are read-out and listed. This step is no longer explained here. More detailed information about the FileSvc interface can be found in Chapter 6: "Folder and file services".

This functionality is only available on a SINUMERIK 840D sl.

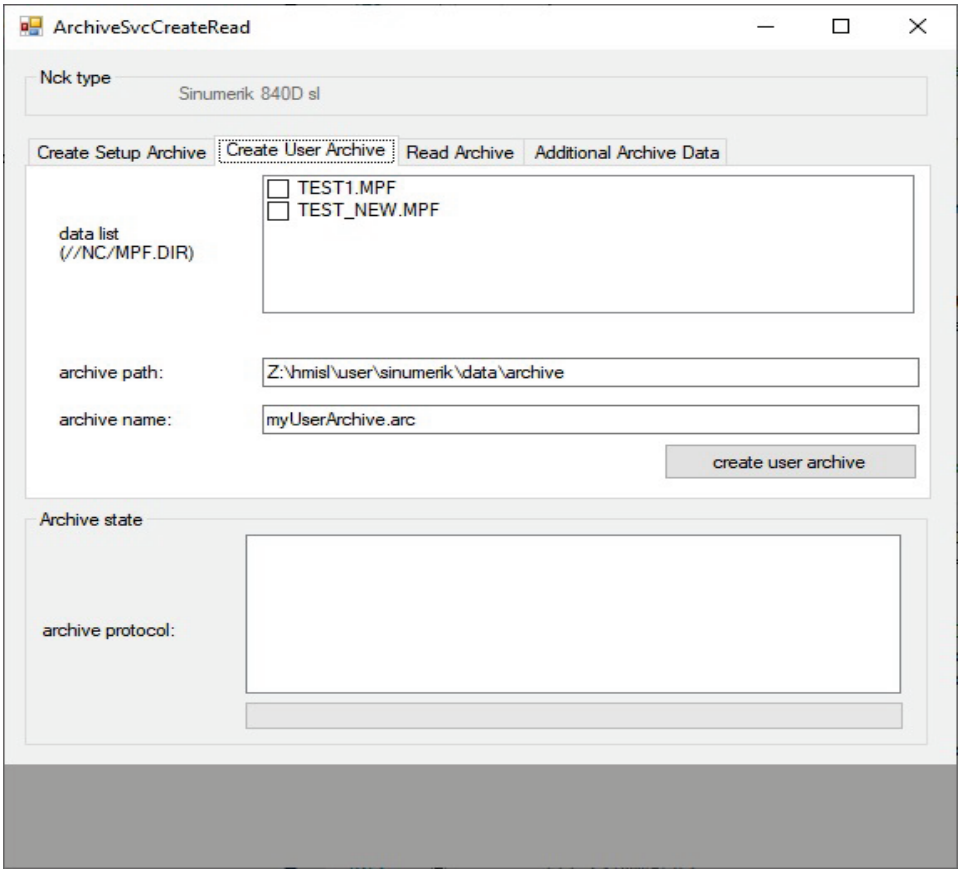


Fig. 8-2: Example, ArchiveSvcCreateRead – creating a user archive

© Siemens AG 2024 All Rights Reserved

Step 1

Create the required member variables.

```
ArchiveSvc m_Archive = null;
Guid m_gdActivGuid = Guid.Empty;
string m_strLastListEntry = string.Empty;
```

Step 2

Create the ArchiveSvc object. This is realized in the ArchiveSvcCreateRead function:

```
m_Archive = new ArchiveSvc();
```

Step 3

Create the list of files to be archived. This list is created based on the selected part programs.

```
List<string> lstFiles = new List<string>();

//get selected Files for archive
foreach (object item in lstUserFiles.CheckedItems)
{
    lstFiles.Add(USER_ARCHIVE_FILE_PATH + "\\" + item.ToString());
}
```

Step 4

Archiving is started using the function call CreateUserArchiveAsync(). The following data is transferred as transfer parameter:

- Path and name of the archive to be created.
- List of files that must be backed up.
- Include HMI files (here: false → no HMI files).
- Include NC compile cycles (here: false → no compile cycles).
- Name of the creator
- Comment
- Delegate for completion
- Delegate for progress

```
m_gdActivGuid = m_Archive.CreateUserArchiveAsync(
                      txtUserArchivePath.Text + txtUserArchiveName.Text,
                      lstFiles.ToArray(),
                      false,
                      false,
                      "CreatorName",
                      "This is a comment",
                      onArchiveSvcActionCompleted,
                      onArchiveSvcActionProgress);
```

Step 5

Implement the delegates.

Important

Because the delegate is called from another thread, not all display elements can be accessed. Consequently, it is desirable to call a second delegate with the same signature with which changes can be performed on the user interface.

Unrestricted © Siemens AG 2024 All Rights Reserved

In the implemented delegate function, a change is sent to the same delegate signature, now in the GUI thread. The actual progress and the end of archiving is now displayed here. As far as the progress is concerned, it is possible that a file name is frequently transferred each time with a different percentage. This is the reason why the last file name is saved in a member variable in the example. Before the file name is output, a check is made as to whether it differs from the last one saved.

```
private void onArchiveSvcActionProgress(Guid guid,
                                        int percentage,
                                        string archiveItem)
{
    // are we in GUI thread (main) ?
    // No we are not in GUI thread => send message to GUI thread
    if (prgProgress.InvokeRequired)
    {
        ArchiveSvcActionProgress internalDelegate = new
                    ArchiveSvcActionProgress(onArchiveSvcActionProgress);
        BeginInvoke(internalDelegate, new Object[] { guid, percentage,
                                                     archiveItem });
    }
    else
    {
        if (guid == m_gdActivGuid)
        {
            //Avoid empty and duplicate entries
            if (("" != archiveItem) && (archiveItem != m_strLastListEntry))
            {
                lstArchiveProtocol.Items.Add(archiveItem);
                m_strLastListEntry = archiveItem;
            }

            //actualize progress bar
            prgProgress.Value = percentage;
        }
    }
}
```

```
private void onArchiveSvcActionCompleted(System.Guid guid,
                                         ArchiveSvcStatus status)
{
    // are we in GUI thread (main) ?
    // No we are not in GUI thread => send message to GUI thread
    if (prgProgress.InvokeRequired)
    {
        ArchiveSvcActionCompleted internalDelegate = new
                    ArchiveSvcActionCompleted(onArchiveSvcActionCompleted);
        BeginInvoke(internalDelegate, new Object[] { guid, status });
    }
    else
    {
        if (guid == m_gdActivGuid)
        {
            //check error
            if (false == status.Ok)
            {
                //and display message
                setStatus("onArchiveSvcActionCompleted(): Error in Completed
                                                          Delegate !!!");
            }
            else
            {
                //Write status text
                setStatus("Archive done!");
            }
        }
    }
}
```


© Siemens AG 2024 All Rights Reserved

8.2.4 Read-in an archive

The following example shows step-by-step which actions are necessary in order to read-in an archive. The matching executable example in the SINUMERIK Integrate Create MyHMI /3GL is the example "ArchiveSvcCreateRead". This chapter describes the functionality under the "Read Archive" tab under the "read in archive" button.

In this example, under "data list" you can select which archives are to be read-in. The progress of the asynchronous call can be tracked under "archive protocol". The status bar indicates whether the call has been successful.

When activating the "Read Archive" tab via the FileSvc interface, all the archive files from the directory "oem\sinumerik\data\archive" are read-out and listed. This step is no longer explained here. More detailed information about the FileSvc interface can be found in Chapter 6: "Directory and File Services"

The example makes a distinction as to whether a SINUMERIK 840D sl or a SINUMERIK ONE is connected. Only the appropriate archives are displayed. The function call does not make differentiation.

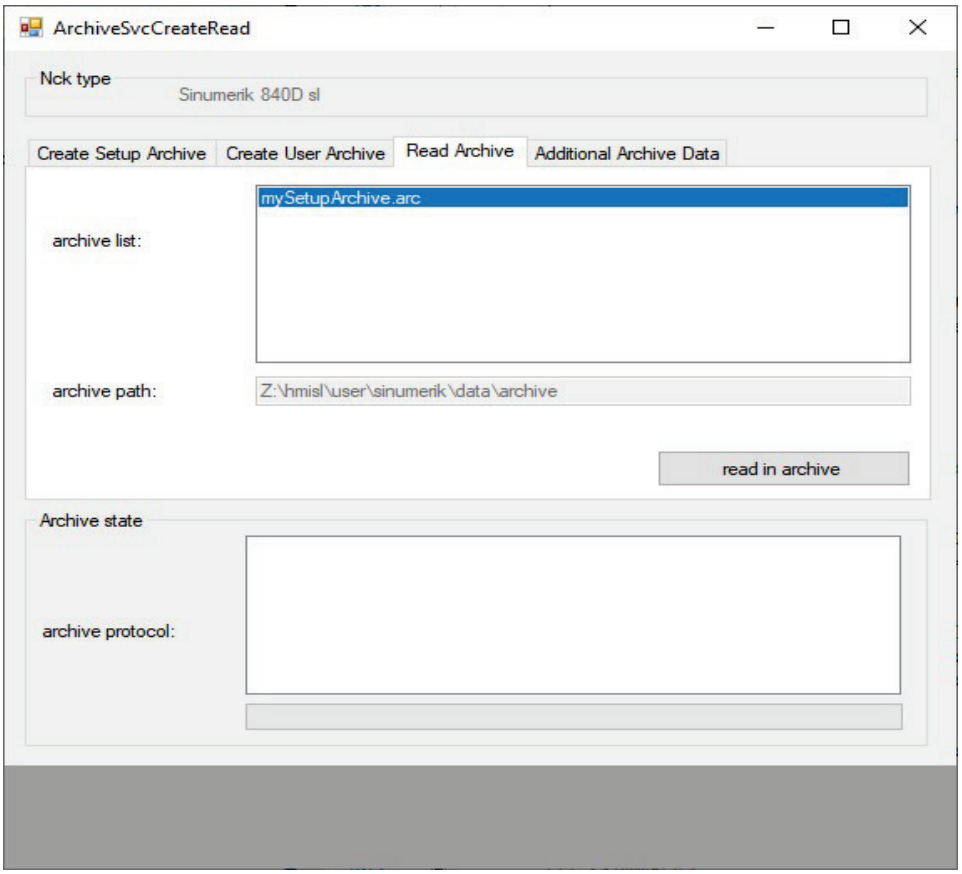


Fig. 8-3: Example, ArchiveSvcCreateRead - reading in an archive

Unrestricted © Siemens AG 2024 All Rights Reserved

Step 1

Create the required member variables.

```
ArchiveSvc m Archive = null;
Guid m_gdActivGuid = Guid.Empty;
string m_strLastListEntry = string.Empty;
```

Step 2

Create the ArchiveSvc object. This is realized in the ArchiveSvcCreateRead function.

```
m_Archive = new ArchiveSvc();
```

Step 3

Start reading-in. In this case, the complete file path is compiled from the selected archived file. The ReadArchiveAsync function then starts the actual reading in (import).
The following data is transferred as transfer parameter:

- Path and name of the archive to be created.
- Delegate for completion
- Delegate for progress

```
//get selected Files for archive
strArchiveFile = READ_ARCHIVE_FILE_PATH + "\\" +
                                    lstReadArchives.SelectedItem.ToString();

//start archiving
m_gdActivGuid = m_Archive.ReadArchiveAsync(strArchiveFile,
                                           onArchiveSvcActionCompleted,
                                           onArchiveSvcActionProgress);
```

Step 4

Implement the delegates.

! Important

Because the delegate is called from another thread, not all display elements can be accessed. Consequently, it is desirable to call a second delegate with the same signature with which changes can be performed on the user interface.

In the implemented delegate function, a change is sent to the same delegate signature, now in the GUI thread. The actual progress and the end of archiving is now displayed here. As far as the progress is concerned, it is possible that a file name is frequently transferred each time with a different percentage. This is the reason why the last file name is saved in a member variable in the example. Before the file name is output, a check is made as to whether it differs from the last one saved.

© Siemens AG 2024 All Rights Reserved

```
private void onArchiveSvcActionProgress(Guid guid,
                                        int percentage,
                                        string archiveItem)
{
    // are we in GUI thread (main) ?
    // No we are not in GUI thread => send message to GUI thread
    if (prgProgress.InvokeRequired)
    {
        ArchiveSvcActionProgress internalDelegate = new
                    ArchiveSvcActionProgress(onArchiveSvcActionProgress);
        BeginInvoke(internalDelegate, new Object[] { guid, percentage,
                                                     archiveItem });
    }
    else
    {
        if (guid == m_gdActivGuid)
        {
            //Avoid empty and duplicate entries
            if (("" != archiveItem) && (archiveItem != m_strLastListEntry))
            {
                lstArchiveProtocol.Items.Add(archiveItem);
                m_strLastListEntry = archiveItem;
            }

            //actualize progress bar
            prgProgress.Value = percentage;
        }
    }
}
```

```
private void onArchiveSvcActionCompleted(System.Guid guid,
                                         ArchiveSvcStatus status)
{
    // are we in GUI thread (main) ?
    // No we are not in GUI thread => send message to GUI thread
    if (prgProgress.InvokeRequired)
    {
        ArchiveSvcActionCompleted internalDelegate = new
                    ArchiveSvcActionCompleted(onArchiveSvcActionCompleted);
        BeginInvoke(internalDelegate, new Object[] { guid, status });
    }
    else
    {
        if (guid == m_gdActivGuid)
        {
            //check error
            if (false == status.Ok)
            {
                //and display message
                setStatus("onArchiveSvcActionCompleted(): Error in Completed
                                                           Delegate !!!");
            }
            else
            {
                //Write status text
                setStatus("Archive done!");
            }
        }
    }
}
```

Unrestricted © Siemens AG 2024 All Rights Reserved

8.2.5 Monitoring the functions of the archive service

The following example shows the actions step-by-step that are necessary in order to monitor asynchronous functions of the archive service that are being executed – and therefore, for example, to be able to respond to a series commissioning in the standard "Commissioning" operating area. The matching executable example in the SINUMERIK Integrate Create MyHMI /3GL is the example "ArchiveSvcNotify".

The example shows a list in which all of the asynchronous function calls (in each case beginning and end) of the archive service are entered.

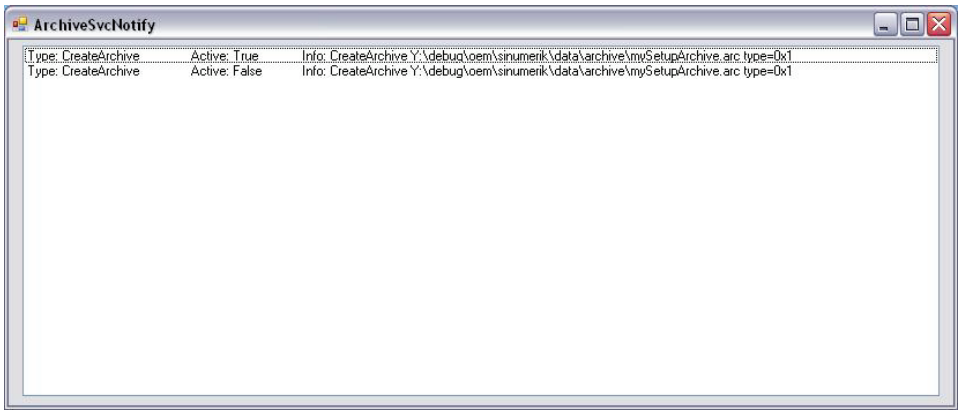


Fig. 8-4: Example, ArchiveSvcNotify – monitoring the archive service

Step 1

Create the required member variables

```
ArchiveSvcNotifier m_asNotifier = null;
Guid m_gdSubscriptionGuid = Guid.Empty;
```

Step 2

Create the ArchiveSvcNotifier object.

```
m_asNotifier = new ArchiveSvcNotifier();
```

Step 3

Install the notification (subscription). In this case, only the delegate is required as the transfer parameter, in which the status changes of the archive service should be received (onArchiveSvcNotificationCb).

```
m_gdSubscriptionGuid = m_asNotifier.Subscribe(onArchiveSvcNotificationCb);
```

Step 4

Implement the delegates.

Important

Because the delegate is called from another thread, not all display elements can be accessed. Consequently, it is desirable to call a second delegate with the same signature with which changes can be performed on the user interface.

© Siemens AG 2024 All Rights Reserved

In the implemented delegate function, a change is sent to the same delegate signature, now in the GUI thread. The state change is now output here. All parameters are written to a list box.

```
private void onArchiveSvcNotificationCb(
                                    Guid guid,
                                    ArchiveSvcNotificationType notification,
                                    bool active,
                                    string info)
{
    // are we in GUI thread (main) ?
    // No we are not in GUI thread => send message to GUI thread
    if (lstData.InvokeRequired)
    {
        ArchiveSvcNotification internalDelegate = new
                        ArchiveSvcNotification(onArchiveSvcNotificationCb);
        BeginInvoke(internalDelegate, new Object[] { guid, notification,
                                                              active, info });
    }
    else
    {
        //check if guid is correct
        if (guid == m_gdSubscriptionGuid)
        {
            //put data into the list box
            lstData.Items.Add("Type: " + notification.ToString() + " \t" +
                                "Active: " + active.ToString() + " \t" +
                                "Info: " + info);
        }
    }
}
```

Unrestricted © Siemens AG 2024 All Rights Reserved

8.3 Reference ArchiveSvc

8.3.1 Definitions

Overview

You can generate and read-in various archive types using ArchiveSvc objects.

When reading in, the contents of the archive are unzipped in the appropriate paths.

It is only possible to asynchronously generate and read-in the archive as a result of the long execution time required.

The notification of the progress or the end of the respective call is realized using the call of the respective implemented delegates.

The class is defined in the "Siemens.Sinumerik.Operate.Services" namespace.

8.3.2 Creating an ArchiveSvc object

Several constructors are available to create an ArchiveSvc object.

Table 8-2: Creating an ArchiveSvc object

public ArchiveSvc.ArchiveSvc() **public ArchiveSvc.ArchiveSvc(string ncu)**	
Parameter	**Meaning**
server	In 1:N constellations, specifies from/to which NCU archive operations are executed.
Return value	-
Possible exceptions	ArgumentException, ArchiveSvcException, ArchiveSvcBusyException

8.3.3 Creating an archive (SINUMERIK 840D sl)

The following archives can be generated:

- Series commissioning archive
- PLC update archive
- User archive
- Paper-tape archive

For each archive type there is a dedicated function with two versions. Variant one only contains the notification about the end of the archiving, variant two also contains a notification about the progress.

© Siemens AG 2024 All Rights Reserved

Creating a series commissioning archive

Table 8-3: Creating a series commissioning archive (SINUMERIK 840D sl)

public Guid ArchiveSvc.CreateSetupArchiveAsync(**string archive,** **SetupArchiveComponent[] components,** **string[] additionalFiles,** **string creator,** **string comment,** **ArchiveSvcActionCompleted onArchiveCreateCompleted)** **public Guid ArchiveSvc.CreateSetupArchiveAsync(** **string archive,** **SetupArchiveComponent[] components,** **string[] additionalFiles,** **string creator,** **string comment,** **ArchiveSvcActionCompleted onArchiveCreateCompleted)** **ArchiveSvcActionProgress onArchiveCreateProgress)**	
Parameter	**Meaning**
Archives	Name and path of the archive to be created.
components	Components that should be backed up.
additionalFiles	Additional files that must be backed up.
creator	Name of the creator
comment	Comment
onArchiveCreateCompleted	Delegate implementation for the message, if the request has been completed.
onArchiveCreateProgress	Delegate implementation for the progress message.
Return value	GUID, which can be used to identify a running request.
Possible exceptions	ArgumentException, ArchiveSvcException, ArchiveSvcBusyException

When parameterizing which components should be archived, the Enumerator **SetupArchiveComponent** is required. The following components are available on a SINUMERIK 840D sl.

Table 8-4: Enumerator SetupArchiveComponent

public enum SetupArchiveComponent	
Parameter	**Meaning**
Hmi	HMI (Operate) data
Ncu	NCU data without compensation data
NcuAndCompensationData	NCU data, including compensation data
CompileCycles	Compile cycles
Plc	PLC data
DrivesAscii	Drive data in ASCII Form
DrivesBinary	Drive data in binary form

Example: See Chapter: 8.2.2

Unrestricted © Siemens AG 2024 All Rights Reserved

Creating a PLC upgrade archive

Table 8-5: Creating a PLC upgrade archive

public Guid ArchiveSvc.CreatePlcUpgradeArchiveAsync(**string archive,** **string creator,** **string comment,** **ArchiveSvcActionCompleted onArchiveCreateCompleted)** **public Guid ArchiveSvc.CreatePlcUpgradeArchiveAsync(** **string archive,** **string creator,** **string comment,** **ArchiveSvcActionCompleted onArchiveCreateCompleted,** **ArchiveSvcActionProgress onArchiveCreateProgress)**

Parameter	Meaning
Archives	Name and path of the archive to be created.
creator	Name of the creator
comment	Comment
onArchiveCreateCompleted	Delegate implementation for the message, if the request has been completed.
onArchiveCreateProgress	Delegate implementation for the progress message.
Return value	GUID, which can be used to identify a running request.
Possible exceptions	ArgumentException, ArchiveSvcException, ArchiveSvcBusyException

Creating a user archive

Table 8-6: Creating a user archive

public Guid ArchiveSvc.CreateUserArchiveAsync(**string archive,** **string[] components,** **bool includeHmi,** **bool includeCompileCycles,** **string creator,** **string comment,** **ArchiveSvcActionCompleted onArchiveCreateCompleted)** **public Guid ArchiveSvc.CreateUserArchiveAsync(** **string archive,** **string[] components,** **bool includeHmi,** **bool includeCompileCycles,** **string creator,** **string comment,** **ArchiveSvcActionCompleted onArchiveCreateCompleted,** **ArchiveSvcActionProgress onArchiveCreateProgress)**

Parameter	Meaning
Archives	Name and path of the archive to be created.
components	List of files to be backed up.
includeHmi	Archive HMI archive files.
includeCompileCycles	Archive compile cycles.
creator	Name of the creator
comment	Comment
onArchiveCreateCompleted	Delegate implementation for the message, if the request has been completed.

© Siemens AG 2024 All Rights Reserved

```
public Guid ArchiveSvc.CreateUserArchiveAsync(
                        string archive,
                        string[] components,
                        bool includeHmi,
                        bool includeCompileCycles,
                        string creator,
                        string comment,
                        ArchiveSvcActionCompleted onArchiveCreateCompleted)

public Guid ArchiveSvc.CreateUserArchiveAsync(
                        string archive,
                        string[] components,
                        bool includeHmi,
                        bool includeCompileCycles,
                        string creator,
                        string comment,
                        ArchiveSvcActionCompleted onArchiveCreateCompleted,
                        ArchiveSvcActionProgress onArchiveCreateProgress)
```

Parameter	Meaning
onArchiveCreateProgress	Delegate implementation for the progress message.
Return value	GUID, which can be used to identify a running request.
Possible exceptions	ArgumentException, ArchiveSvcException, ArchiveSvcBusyException

Example: See Chapter: **8**.2.3

Creating a paper-tape archive

Table 8-7: Creating a paper-tape archive

```
public Guid ArchiveSvc.CreatePaperTapeArchiveAsync(
                        string archive,
                        string[] components,
                        bool iso,
                        bool crlf,
                        ArchiveSvcActionCompleted onArchiveCreateCompleted)

public Guid ArchiveSvc.CreatePaperTapeArchiveAsync(
                        string archive,
                        string[] components,
                        bool iso,
                        bool crlf,
                        ArchiveSvcActionCompleted onArchiveCreateCompleted,
                        ArchiveSvcActionProgress onArchiveCreateProgress)
```

Parameter	Meaning
Archives	Name and path of the archive to be created.
components	List of files to be backed up.
iso	ISO (true) or DIN file (false).
crlf	With/without carriage linefeed.
onArchiveCreateComplete	Delegate implementation for the message, if the request has been completed.
onArchiveCreateProgress	Delegate implementation for the progress message.
Return value	GUID, which can be used to identify a running request.
Possible exceptions	ArgumentException, ArchiveSvcException, ArchiveSvcBusyException

Unrestricted © Siemens AG 2024 All Rights Reserved

8.3.4 Creating an archive (SINUMERIK ONE)

The following archives can be generated:

- Series commissioning archive
- Backup archive

A function with 2 variants is available. Variant one only contains the notification about the end of the archiving, variant two also contains a notification about the progress.

Creating a series commissioning or backup archive

Table 8-8: Creating a series commissioning or backup archive (SINUMERIK ONE)

```
public Guid CreateArchiveAsync(
                        string archive,
                        ArchiveType archiveType,
                        SetupArchiveComponent[] components,
                        string creator,
                        string comment,
                        ArchiveSvcActionCompleted onArchiveCreateCompleted)

public Guid CreateArchiveAsync(
                        string archive,
                        ArchiveType archiveType,
                        SetupArchiveComponent[] components,
                        string creator,
                        string comment,
                        ArchiveSvcActionCompleted onArchiveCreateCompleted,
                        ArchiveSvcActionProgress onArchiveCreateProgress)
```

Parameter	Meaning
Archives	Name and path of the archive to be created.
archiveType	Valid archive types for SINUMERIK ONE are: DSF_Setup ➔ Series commissioning archive DSF_Backup ➔ Backup archive
components	Components that should be backed up.
creator	Name of the creator
comment	Comment
onArchiveCreateCompleted	Delegate implementation for the message, if the request has been completed.
onArchiveCreateProgress	Delegate implementation for the progress message.
Return value	GUID, which can be used to identify a running request.
Possible exceptions	ArgumentException, ArchiveSvcException, ArchiveSvcBusyException

© Siemens AG 2024 All Rights Reserved

When parameterizing which components should be archived, the Enumerator **SetupArchiveComponent** is required. The following components are available on SINUMERIK ONE:

Table 8-9: Enumerator SetupArchiveComponent

public enum SetupArchiveComponent	
Parameter	**Meaning**
DsfHmi	HMI (Operate) data
DsfNck	NCK data
DsfNcExtend	NCExtend data
DsfPlc	PLC data
DsfDrive	Drive data
DsfSys	System settings

Example: See Chapter: 8.2.2

8.3.5 Reading-in an archive

The function for reading-in an archive is available in three versions. Variant one only contains the notification about the end of the archiving, variant two also contains a notification about the progress. The third version has – as a supplement – also a notification for status changes or queries.
A prompt could be whether a file should be overwritten, for example.
In the case of versions one and two, all possible queries are automatically and positively acknowledged.
Details on the queries / status changes are provided in Chapter: 8.4

Table 8-10: Reading-in an archive

public Guid ArchiveSvc.ReadArchiveAsync(
string archive,
ArchiveSvcActionCompleted onArchiveReadCompleted)

public Guid ArchiveSvc.ReadArchiveAsync(
string archive,
ArchiveSvcActionCompleted onArchiveReadCompleted,
ArchiveSvcActionProgress onArchiveReadProgress)

public Guid ArchiveSvc.ReadArchiveAsync(
string archive,
ArchiveSvcActionCompleted onArchiveReadCompleted,
ArchiveSvcActionProgress onArchiveReadProgress,
ArchiveSvcStatusChanged onArchiveSvcStatusChanged))

Parameter	**Meaning**
Archives	Name and path of the archive that is to be read in.
onArchiveCreateComplete	Delegate implementation for the message, if the request has been completed.
onArchiveCreateProgress	Delegate implementation for the progress message.
onArchiveSvcStatusChanged	Delegate implementation for the notification of a status change. This can also be a prompt of the interface, e.g. whether or not a file should be overwritten.
Return value	GUID, which can be used to identify a running request.
Possible exceptions	ArgumentException, ArchiveSvcException, ArchiveSvcBusyException

Unrestricted © Siemens AG 2024 All Rights Reserved

8.3.6 Further functions

Canceling a request that is being executed

Table 8-11: Canceling a request that is being executed

public Guid ArchiveSvc.CancelAsync(ArchiveSvcActionCanceled cb)	
Parameter	**Meaning**
cb	Delegate implementation for the message, if the request has been interrupted.
Return value	GUID, which can be used to identify a running request.
Possible exceptions	ArgumentException, ArchiveSvcException, ArchiveSvcBusyException

Reading-out the type of an archive

Table 8-12: Reading-out the type of an archive

public ArchiveType ArchiveSvc.GetType(string archive)	
Parameter	**Meaning**
Archives	Name and path of the archive
Return value	Archive type
Possible exceptions	ArgumentException, ArchiveSvcException, ArchiveSvcBusyException

Reading-out entries of an archive

Table 8-13: Reading-out entries of an archive

public string[]ArchiveSvc.GetEntries(string archive)	
Parameter	**Meaning**
Archives	Name and path of the archive
Return value	Array of texts that contain individual archive entries
Possible exceptions	ArgumentException, ArchiveSvcException, ArchiveSvcBusyException

Reading-out comments of an archive

Table 8-14: Reading-out comments of an archive

public string ArchiveSvc.GetComment(string archive)	
Parameter	**Meaning**
Archives	Name and path of the archive
Return value	Comment of the archive
Possible exceptions	ArgumentException, ArchiveSvcException, ArchiveSvcBusyException

Reading-out the creator of an archive

Table 8-15: Reading-out the creator of an archive

public string ArchiveSvc.GetCreator(string archive)	
Parameter	**Meaning**
Archives	Name and path of the archive
Return value	Creator of the archive
Possible exceptions	ArgumentException, ArchiveSvcException, ArchiveSvcBusyException

© Siemens AG 2024 All Rights Reserved

Reading-out the creation date of an archive

Table 8-16: Reading-out the creation date of an archive

public DateTime ArchiveSvc.GetTimeStampInternal(string archive)	
Parameter	**Meaning**
Archives	Name and path of the archive
Return value	Creation date of the archive. This is the internal time stamp of the archive and not the system time stamp of the archive file.
Possible exceptions	ArgumentException, ArchiveSvcException, ArchiveSvcBusyException

Reading-out the NCU version of an archive

Table 8-17: Reading-out the NCU version of an archive

public string ArchiveSvc.GetNcuSystemSoftwareVersion(string archive)	
Parameter	**Meaning**
Archives	Name and path of the archive
Return value	NCU version of the archive
Possible exceptions	ArgumentException, ArchiveSvcException, ArchiveSvcBusyException

8.3.7 Specifications for implementing delegates

Notification about the progress of a request

Table 8-18: Notification about the progress of a request (delegate)

public delegate void ArchiveSvc.ArchiveSvcActionProgress(Guid guid, int percentage, string archiveItem)	
Parameter	**Meaning**
guid	GUID, which was returned when issuing a request. It is used to identify the request that is being executed.
percentage	Progress as a percentage
archiveItem	Current file, which is backed up

Notification about the end of a request

Table 8-19: Notification about the end of a request (delegate)

public delegate void ArchiveSvc.ArchiveSvcActionCompleted(Guid guid, ArchiveSvcStatus status)	
Parameter	**Meaning**
guid	GUID, which was returned when issuing a request. It is used to identify the request that is being executed.
status	Status of the request.

Unrestricted © Siemens AG 2024 All Rights Reserved

Notification about a status change or a query

Table 8-20: Notification about a status change / query (delegate)

public delegate void ArchiveSvc.ArchiveSvcStatusChanged(Guid guid, ArchiveSvcStatus status, ref bool confirm)	
Parameter	**Meaning**
guid	GUID, which was returned when issuing a request. It is used to identify the request that is being executed.
status	Request status, this can also be a query, e.g. whether or not a file should be overwritten.
confirm	Response of the status change (e.g. whether the file should be overwritten.

Details on the prompts / status changes are provided in Chapter 8.4

Notification about data that must be entered (PLC password protection)

Table 8-21: Notification about data that must be entered/prompt (delegate)

public delegate void ArchiveSvcDataRequired(Guid guid, ArchiveSvcStatus status, ref bool confirm, ref string data);	
Parameter	**Meaning**
guid	GUID, which was returned when issuing a request. It is used to identify the request that is being executed.
status	Status of the request.
confirm	Response, whether data was transferred.
data	Transferred data (PLC password)

Notification, if a request that is being executed has been completely canceled

Table 8-22: Notification, if a request that is being executed has been completely canceled

public delegate void ArchiveSvc.ArchiveSvcActionCanceled(Guid guid)	
Parameter	**Meaning**
guid	GUID, which was returned when issuing a request. It is used to identify the request that is being executed.

© Siemens AG 2024 All Rights Reserved

8.4 Reference ArchiveSvcStatus

The ArchiveSvcStatus class contains properties, which on the one hand, are transferred when exiting a request, and on the other hand, for status changes or for queries.

8.4.1 Properties

Ok

Overall result of the request. If the value "false" is transferred here, then details on the errors are available in the "Errors" property.

Table 8-23:Ok

public bool ArchiveSvcStatus.Ok { get; }

Errors

List of any errors that have occurred. The structure of the "error" type includes both the error number, and the associated item that had generated an error.

Table 8-24:Errors

public error[] ArchiveSvcStatus.Errors { get; }

ArchiveItem

For end request, the file name of the archive generated/read-in is located here.

Table 8-25:ArchiveItem

public string ArchiveSvcStatus.ArchiveItem { get; }

Percentage

Progress of the request as a percentage.

Table 8-26:Percentage

public int ArchiveSvcStatus.Percentage { get; }

Request

The Request property must only be evaluated in the implementation of the delegate ArchiveSvcStatusChanged. Interface queries are issued through this.

Table 8-27:Request

public ArchiveSvcRequest ArchiveSvcStatus.Request { get; }

Unrestricted © Siemens AG 2024 All Rights Reserved

Queries are identified using an Enumerator (ArchivSvcRequest), that can have the following values:

Table 8-28: Creating a series commissioning archive

public enum ArchiveSvcRequest	
Parameter	**Meaning**
NoRequest	No information is contained.
Progress	Progress. The actual value is in the "Percentage" property.
ListCompleted	Request has been executed.
About	Additional Information while a request is being processed. Additional data is in the "ArchiveItem" property.
Error	Error occurred. Data on the error that has occurred is provided in the "Errors" property.
Canceled	Request completed
NewFileName	Queries whether a file name or file path should be changed when reading-in an archive. The file name is in the "ArchiveItem" property.
Confirm	General query. Currently there is only one query that can occur when reading-in an archive (queries whether "CFG_RESET_INI" should be read in).
ConfirmNcuVersion	Queries whether an archive should be read-in although it is the incorrect version or type. Additional information is provided in the "ArchiveItem" property.
ConfirmOverwrite	Queries whether an existing file should be overwritten when reading-in an archive. The file name is in the "ArchiveItem" property.
ConfirmReadArchive	Queries whether an archive should be imported.

Archive service in 1:N constellations

By default, the archiving function is executed on the NCU which Operate is viewing. In a 1:N constellation, the archiving function can be executed on a specific NCU (an NCU other than the one displayed in Operate) as follows:

1. Create ArchiveSvc object with NCU name.
   By specifying the name of an NCU (example: NCU2) when an ArchiveSvc object is created (see ArchiveSvc constructors), access to the specified NCU will always take place via this ArchiveSvc instance.

Note

NCU names are listed in the MMC.INI file.
Entry:
[GLOBAL]
NcddeMachineNames=*NCU1,NCU2*

© Siemens AG 2024 All Rights Reserved

8.5 Reference ArchiveSvcNotifier

8.5.1 Definitions

Overview

Using ArchiveSvcNotifier objects, asynchronous archive service functions that are being executed can be monitored so that you can specifically respond to a series commissioning in the standard operating area "Commissioning".

8.5.2 Creating an ArchiveSvcNotifier object

Several constructors are available to create an ArchiveSvcNotifier object.

Table 8-29: Canceling a request that is being executed

public ArchiveSvcNotifier() **public ArchiveSvcNotifier(string server)**	
Parameter	**Meaning**
server	Currently not relevant
Return value	-
Possible exceptions	ArgumentException, ArchiveSvcException, ArchiveSvcBusyException

8.5.3 Activating the monitoring function

Table 8-30: Activating the monitoring function

public Guid ArchiveSvcNotifier.Subscribe(ArchiveSvcNotification cb)	
Parameter	**Meaning**
cb	Delegate implementation for the message when the status of the archive service changes.
Return value	GUID, which can be used to identify a running request.
Possible exceptions	ArgumentException, ArchiveSvcException, ArchiveSvcBusyException

Example: See Chapter: 8.2.5

8.5.4 Deactivating the monitoring function

Table 8-31: Deactivating the monitoring function

public void ArchiveSvcNotifier.UnSubscribe(ArchiveSvcNotification cb)	
Parameter	**Meaning**
cb	Delegate implementation for the message when the status of the archive service changes.
Return value	-
Possible exceptions	ArgumentException, ArchiveSvcException, ArchiveSvcBusyException

Example: See Chapter: 8.2.5

Unrestricted © Siemens AG 2024 All Rights Reserved

8.5.5 Specifications for implementing delegates

Notification about a change in the status of the archive service

Table 8-32: Notification about a change in the status of the archive service

public delegate void ArchiveSvcNotification(Guid guid, ArchiveSvcNotificationType notification, bool active, string info)	
Parameter	**Meaning**
guid	GUID, which when issuing the subscribe request was returned. It is used to identify the request that is being executed.
Notification	Current activity, for further details see below
active	True = activity started, False = activity completed
info	Additional information

The type of activity is described in the enumerator **ArchiveSvcNotificationType** in a coded fashion:

Table 8-33: Enumerator ArchiveSvcNotificationType

public enum ArchiveSvcNotificationType	
Parameter	**Meaning**
Idle	No action
CreateArchiveInputList	File list is generated
CreateArchive	Archive is created
ReadArchive	Archive is read-in (imported)
ConvertArchive	Archive is converted
ReadArchiveEntries	Archive entries are read
DeleteArchiveEntries	Archive entries are deleted
ProcessJobList	Job list is executed

Example: See Chapter: 8.2.5

© Siemens AG 2024 All Rights Reserved

9

9 Drive machine data service

Objective of the chapter

This chapter describes the interface of the drive machine data service. This service can be used to read, write, and save SINAMICS drive data, as well as to trigger an NCU reset or a reset of the SINAMICS drives.

Note

Do not read drive parameters faster than in a 1 second cycle, slower is better. Otherwise, an excessively high communication load is created - that can result in blocking communication between the HMI and realtime system, and can therefore block the HMI itself.

Unrestricted © Siemens AG 2024 All Rights Reserved

9.1 Introduction

9.1.1 Class model

Overview

The class model of the drive machine data service consists primarily of the following classes:

- Drive
- DrivesSvc
- DrivesSvcStatus

Namespace

The classes are defined in the "Siemens.Sinumerik.Operate.Services" namespace. Consequently, this namespace must be integrated into the C# project at the start of the associated file:

```
using Siemens.Sinumerik.Operate.Services;
```

class DrivesSvc

The DrivesSvc class is used to create objects, list the available Drive objects, and select a specific drive. The class also supports functions for saving drive parameters to the CF card on the NCU and for triggering various resets of the NCU and the control units.

class Drive

The Drive class is used to generate Drive objects which represent one of the following SINAMICS components:

- Control Unit
- Com Module
- Drive
- Line Module
- I/O

The class contains attributes which describe the corresponding object in more detail, along with functions for reading, writing, and saving drive parameters.

class DrivesSvcStatus

The DrivesSvcStatus class is used to generate objects containing information about the status of a request. These are an error number and a flag as to whether the request was successfully completed.

© Siemens AG 2024 All Rights Reserved

9.2 Step-by-step examples

Overview

The following chapters show the various applications of the drive machine data service step-by-step. An executable example is included in the SINUMERIK Integrate Create MyHMI /3GL for each "step-by-step example".

These examples are shown in the following overview of examples.

Table 9-1: Overview of the examples

Application	Example	Chapter
Asynchronous listing of the drive objects of a control	DrivesSvcListDrivesAsync	9.2.2
Reading and writing of drive parameters	DrivesSvcReadWrite	9.2.3
Asynchronous saving of drive objects	DrivesSvcSaveControlUnitsAsync	9.2.4

Only the points relevant to DriveService are described. The comments for the sources contain more detailed information.

9.2.1 Preparation

Overview

Preparatory measures are described in Chapter 2.1 ➔ "New Project". All of the items described there must be executed first.

9.2.2 Asynchronous listing of the drive objects of a control

Overview

The following example provides a step-by-step guide to listing the drive objects of a control asynchronously.

In the example, the type of drive objects to be listed is selected in a selection box. The list is created by clicking the "Get list of drive objects async" button. The status bar indicates whether the call has been successful.

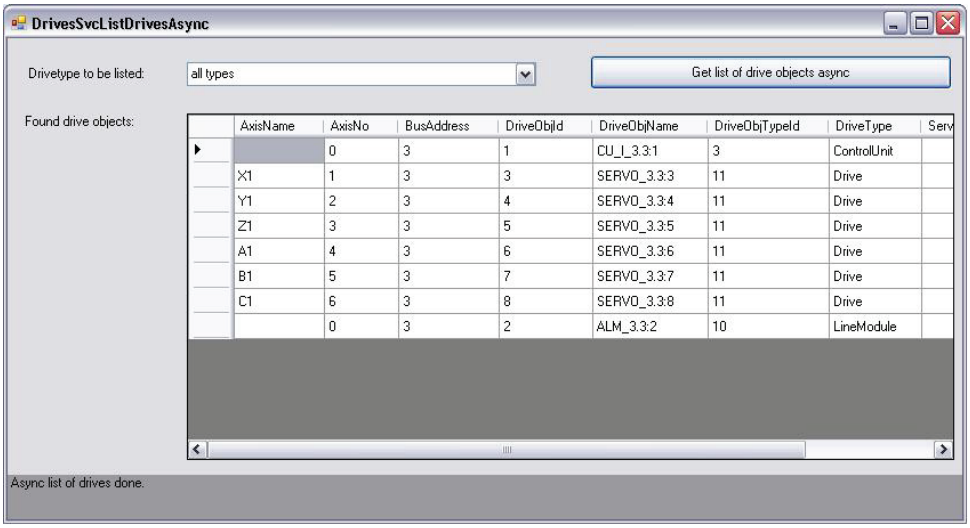


Fig. 9-1: Asynchronous listing of drive objects

Unrestricted © Siemens AG 2024 All Rights Reserved

Step 1

Create the required member variables. A DrivesSvc type variable and a System.Guid type variable are required.

```
DrivesSvc m_DrivesSvc = null;
System.Guid m_GuidListAsync;
```

Step 2

Generate the DrivesSvc object in the function frmMain().

```
m_DrivesSvc = new DrivesSvc();
```

Step 3

Trigger asynchronous listing of the drive objects that are available in the control. The type of drive objects to be included in the list is selected in a selection box. Therefore, there are two calls, one with and one without type restriction. The call returns a GUID, which is subsequently used to identify the delegates implemented in the calls. The OnCompleted() delegate is transferred; this delegate is called when the request has been completed.

```
//get all types of drive objects
if ("all types" == cbxDriveType.SelectedItem)
{
    m_GuidListAsync = m_DrivesSvc.ListDrivesAsync(OnCompleted);
}
// get only selected type of drive objects
else
{
    m_GuidListAsync = m_DrivesSvc.ListDrivesAsync(
                                        (DriveType)cbxDriveType.SelectedItem,
                                        OnCompleted);
}
```

Step 4

Implement the delegate.

! Important

Because the delegate is called from another thread, not all display elements can be accessed. Consequently, it is desirable to call a second delegate with the same signature with which changes can be performed on the user interface.

In the implemented delegate function, a change is sent to the same delegate signature, now in the GUI thread. The drive objects found are listed here. First, a check is made to ascertain whether the transferred GUID is the same as the one returned when ListDrivesAsync was called. The properties of the drive objects are entered in a DataGridView for display purposes.

© Siemens AG 2024 All Rights Reserved

```
private void OnCompleted(Guid guid, Drive[] drives)
{
    // are we in GUI thread (main) ?
    // No we are not in GUI thread => send message to GUI thread
    if (gridDrives.InvokeRequired)
    {
        ListDrivesCompleted completedDelegate = new
                                        ListDrivesCompleted(OnCompleted);
        BeginInvoke(completedDelegate, new Object[] { guid, drives });
    }
    else
    {
        if (m_GuidListAsync == guid)
        {
           // clear grid, create columns
           gridDrives.Rows.Clear();
           gridDrives.ColumnCount = 9;

           // initialize grid
           gridDrives.AllowUserToAddRows = false;
           gridDrives.AutoSizeColumnsMode =
                                DataGridViewAutoSizeColumnsMode.AllCells;
           gridDrives.ColumnHeadersDefaultCellStyle.Alignment =
                               DataGridViewContentAlignment.MiddleCenter;

           // set captions for columns
           gridDrives.Columns[0].HeaderText = "AxisName";
           ...
           gridDrives.Columns[8].HeaderText = "SlaveAddress";

           // fill grid with properties of found drive objects
           for (int i = drives.GetLowerBound(0); i <=
                                          drives.GetUpperBound(0); i++);
           {
               string[] rowContent = {drives[i].AxisName,
                                drives[i].AxisNumber.ToString(),
                                drives[i].BusAddress.ToString(),
                                drives[i].DriveObjectId.ToString(),
                                drives[i].DriveObjectName,
                                drives[i].DriveObjectTypeId.ToString(),
                                drives[i].DriveType.ToString(),
                                drives[i].Server,
                                drives[i].SlaveAddress.ToString()};
               gridDrives.Rows.Add(rowContent);
           }
        }
    }
}
```

9.2.3 Reading and writing of drive parameters

Overview

The next example provides a step-by-step guide to reading and writing drive parameters.

In the example, the drive object in which parameters are to be read or written is selected via a selection box. Three different examples have been implemented for addressing parameters. In the "Parameter named by String" row, the parameter is addressed using a string; in the "Parameter named by int" row, the parameter is addressed using an integer value; and in the "Parameter named by int, int" row, the parameter is addressed using integer values for the parameter number and the index. Values for the parameters which are being written by pressing the corresponding "write" buttons can be specified in the "Value" text boxes. Press the "read" buttons to read each of the parameter values and write an entry to the corresponding "Value" output field. The status bar indicates whether the call has been successful.

Unrestricted © Siemens AG 2024 All Rights Reserved

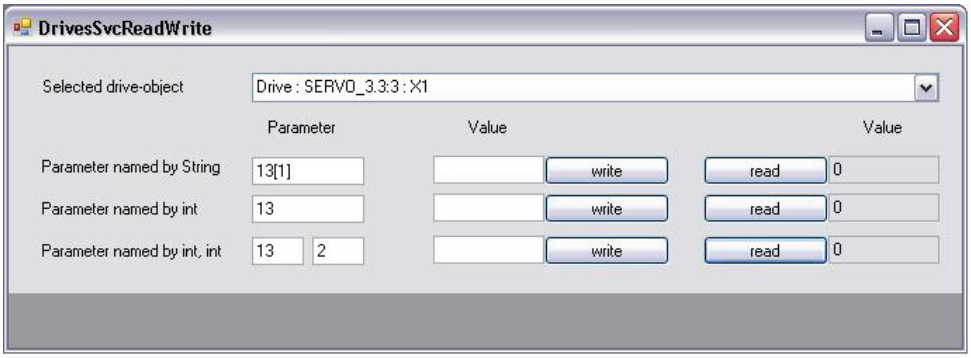


Fig. 9-2: Reading and writing of drive parameters

Step 1

Create the required member variables. A DrivesSvc type variable is required. A Drive type variable contains the drive object currently selected in the application. An array with Drive type variables contains the drive objects identified on the control.

```
DrivesSvc m_DrivesSvc = null;
Drive m_SelectedDrive = null;
Drive[] m_DrivesList;
```

Step 2

Generate the DrivesSvc object in the function frmMain().

```
m_DrivesSvc = new DrivesSvc();
```

Step 3

Identify the drive objects available on the control with the ListDrives() function, which is executed synchronously and returns an array of the Drive objects found.

```
m_DrivesList = m_DrivesSvc.ListDrives();
```

Step 4

In the example, the drive objects found are entered in a selection box. The currently selected drive object is identified as follows:

```
m_SelectedDrive = m_DrivesList[cbxDriveObject.SelectedIndex];
```

Step 5

Write the parameter values. The parameter addresses (number and if applicable index) and the values to be set are saved in text boxes.

Parameter address (and index) defined as string:

```
m_SelectedDrive.Write(txtParameterString.Text,
                      Convert.ToDouble(txtWriteString.Text));
```

Parameter address defined as integer value:

```
m_SelectedDrive.Write(Convert.ToInt32(txtParameterInt.Text),
                      Convert.ToDouble(txtWriteInt.Text));
```

Parameter address and index defined as integer values:

```
m_SelectedDrive.Write(Convert.ToInt32(txtParameterIntIntPara.Text),
                      Convert.ToInt32(txtParameterIntIntIndex.Text),
                      Convert.ToDouble(txtWriteIntInt.Text));
```


© Siemens AG 2024 All Rights Reserved

Step 6

Read the parameter values. The parameter addresses (number and if applicable index) are saved in text boxes.

Parameter address (and index) defined as string:

```
double dRes = 0.0
dRes = m_SelectedDrive.Read(txtParameterString.Text);
txtReadString.Text = dRes.ToString();
```

Parameter address defined as integer value:

```
double dRes = 0.0
dRes = m_SelectedDrive.Read(Convert.ToInt32(txtParameterInt.Text));
txtReadInt.Text = dRes.ToString();
```

Parameter address and index defined as integer values:

```
double dRes = 0.0;
dRes = m_SelectedDrive.Read(Convert.ToInt32(txtParameterIntIntPara.Text),
                            Convert.ToInt32(txtParameterIntIntIndex.Text));
txtReadIntInt.Text = dRes.ToString();
```

9.2.4 Asynchronous saving of drive objects

Overview

The next example shows how to save drive objects asynchronously to the CF card on the NCU.

Click the "Save control units async" button to start the save operation. Click the "Cancel" button to abort the action for the application.

Important

Even after a Cancel call, saving of drive data to the card is completed by the system to avoid inconsistencies in the drive data.

The progress of the save operation is indicated by a percentage indicator and a progress bar. The status bar indicates whether the call has been successful.

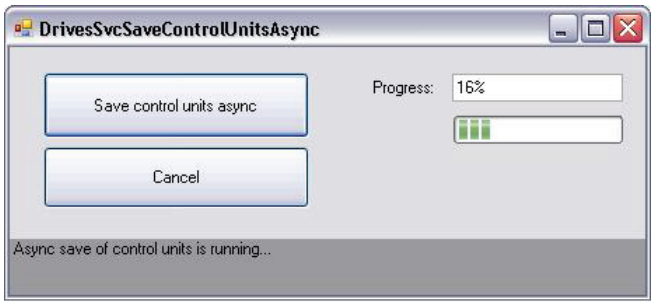


Fig. 9-3: Asynchronous saving of drive objects

Unrestricted © Siemens AG 2024 All Rights Reserved

Step 1

Create the required member variables. A DrivesSvc type variable and a System.Guid type variable are required.

```
DrivesSvc m_DrivesSvc = null;
System.Guid m_GuidListAsync;
```

Step 2

Generate the DrivesSvc object in the function frmMain().

```
m_DrivesSvc = new DrivesSvc();
```

Step 3

Trigger asynchronous saving of the drive objects. Here, a GUID is returned, which is subsequently used to identify the delegates implemented in the calls. Two delegates are transferred with the call. The onProgressCUSaveAsync() function cyclically provides information about progress. The onCompletedCUSaveAsync() function is called when the request is complete.

```
m_GuidCUSaveAsync = m_DrivesSvc.SaveControlUnitsAsync(onProgressCUSaveAsync,
                                                      onCompletedCUSaveAsync);
```

Step 4

Implement the onProgressCUSaveAsync() delegate.

Important

Because the delegate is called from another thread, not all display elements can be accessed. Consequently, it is desirable to call a second delegate with the same signature with which changes can be performed on the user interface.

In the implemented delegate function, a change is sent to the same delegate signature, now in the GUI thread. Here, progress is indicated in the form of a percentage value displayed in a text field and also by a progress bar. First, a check is made to ascertain whether the GUID is the same as the one returned when SaveControlUnitsAsync() was called. Next, the values are set accordingly in the display.

```
private void onProgressCUSaveAsync(Guid guid, int percentage)
{
    // are we in GUI thread (main) ?
    // No we are not in GUI thread => send message to GUI thread
    if (txtProgress.InvokeRequired)
    {
        SaveProgress progressCUSaveAsync = new
                                        SaveProgress(onProgressCUSaveAsync);
        BeginInvoke(progressCUSaveAsync, new Object[] { guid, percentage });
    }
    else
    {
        if (m_GuidCUSaveAsync == guid)
        {
            txtProgress.Text = percentage.ToString() + "%";
            progressBarSave.Value = percentage;
        }
    }
}
```


© Siemens AG 2024 All Rights Reserved

Step 5

Implement the onCompletedCUSaveAsync() delegate.

Important

Because the delegate is called from another thread, not all display elements can be accessed. Consequently, it is desirable to call a second delegate with the same signature with which changes can be performed on the user interface.

In the implemented delegate function, a change is sent to the same delegate signature, now in the GUI thread. Information can be output here following completion of the action. First, a check is made to ascertain whether the GUID is the same as the one returned when SaveControlUnitsAsync() was called. Output is then made to the user interface. In this case, text in the status bar.

```
private void onCompletedCUSaveAsync(Guid guid, DrivesSvcStatus status)
{
    // are we in GUI thread (main) ?
    // No we are not in GUI thread => send message to GUI thread
    if (txtProgress.InvokeRequired)
    {
        SaveCompleted completedDelegate = new
                                    SaveCompleted(onCompletedCUSaveAsync);
        BeginInvoke(completedDelegate, new Object[] { guid, status });
    }
    else
    {
        if (m_GuidCUSaveAsync == guid)
        {
            setStatus("Control units saved to card.");
        }
    }
}
```

Unrestricted © Siemens AG 2024 All Rights Reserved

9.3 Reference Drive

9.3.1 Definitions

Overview

A Drive object represents one of the following SINAMICS components:

- Control Unit
- Com Module
- Drive
- Line Module
- I/O

The Drive class contains attributes which describe the corresponding object in more detail. The attributes are provided as properties. The class supports functions for reading, writing, and saving drive parameters.

This class is defined in the "Siemens.Sinumerik.Operate.Services" namespace.

9.3.2 Creating a Drive object

Two constructors are available for creating a Drive object.

Table 9-2: Creating a Drive object

public Drive() **public Drive(string server)**	
Parameter	**Meaning**
server	In 1:N constellations, specifies from/to which NCU access operations are executed. **NOTICE:** Function is not supported.

9.3.3 Releasing assigned resources

This function releases the resources used by a Drive object again. If this is not done, the system will release the resources at some point once the object has become invalid.

Table 9-3: Re-releasing occupied resources

public void Dispose()	
Parameter	**Meaning**
-	-

9.3.4 Reading drive parameters

The Read function is used for synchronous reading of drive parameters out of the control.

Synchronous reading of a drive parameter (Read)

The parameter numbers required and if applicable the required index of an array can be specified as integer values.

© Siemens AG 2024 All Rights Reserved

Table 9-4: Reading an individual drive parameter (parameter specified as integer)

public double Read(int parameter) **public double Read(int parameter, int index)**	
Parameter	**Meaning**
parameter	Number of the drive parameter
index	Index of the drive parameter for arrays
Return value	Value of the drive parameter

Examples:
double dRes1 = m_Drive.Read(35);
double dRes2 = m_Drive.Read(13,5);

It is also possible to transfer the parameter address as a string.

Table 9-5: Reading an individual drive parameter (parameter specified as string)

public double Read(string parameter)	
Parameter	**Meaning**
parameter	Number of the drive parameter (with or without index)
Return value	Value of the drive parameter

Examples:
double dRes1 = m_Drive.Read("13[5]");
double dRes2 = m_Drive.Read("35");

Synchronous reading of an array of drive parameters (Read)

To read out multiple consecutive values of a drive parameter array, the required area can be transferred to the Read function with the parameter number, the start index, and the end index.

Table 9-6: Reading an array of drive parameters

public double[] Read(int parameter, int startIndex, int endIndex)	
Parameter	**Meaning**
parameter	Number of the drive parameter
startIndex	Index where reading starts
endIndex	Last index to be read
Return value	Array of values of drive parameters

Example:
double[] dRes = m_Drive.Read(13,0,10);

Synchronous reading of multiple drive parameters (Read)

By specifying the required parameter numbers and if applicable indices in a string array, it is possible to read out multiple different drive parameters at once.

Table 9-7: Reading of multiple drive parameters

public double[] Read(string[] parameters)	
Parameter	**Meaning**
parameters	Array of numbers of drive parameters (with or without index)
Return value	Array of values of drive parameters

Unrestricted © Siemens AG 2024 All Rights Reserved

9.3.5 Writing drive parameters

Synchronous writing to individual drive parameters (Write)

The Write function is used for synchronous writing of drive parameters.

The parameter number and if applicable the parameter index can be transferred as integer values.

Table 9-8: Write – writing drive parameters

public void Write(int parameter, double value) public void Write(int parameter, int index, double value)	
Parameter	**Meaning**
parameter	Number of the drive parameter
index	Index of the drive parameter for arrays
value	Value to be written

Examples:
m_Drive.Write(952, 0);
m_Drive.Write(13,2,1.23);

The drive parameter can also be addressed using a string with parameter and index.

Table 9-9: Writing drive parameters

public void Write(string parameter, double value)	
Parameter	**Meaning**
parameter	Number (and index) of drive data
value	Value to be written

Examples:
m_Drive.Write("13[2]", 1.23);
m_Drive.Write("952", 0);

9.3.6 Saving a drive object

Synchronous saving of Drive object data

The Save function saves the drive data of the drive object synchronously to the CF card on the NCU, in the /user/sinamics/data directory.

Table 9-10: Synchronous saving of Drive object

public void Save()	
Parameter	**Meaning**
-	-

Asynchronous saving of Drive object data

The SaveAsync function saves data asynchronously to the CF card on the NCU.

The functions call the implementation of the following delegates:

- SaveProgress()
- SaveCompleted()

© Siemens AG 2024 All Rights Reserved

Table 9-11: Asynchronous saving of Drive object

public System.Guid SaveAsync(SaveProgress onProgress, SaveCompleted onCompleted)	
Parameter	**Meaning**
onProgress	Delegate implementation for the progress message
onCompleted	Delegate implementation for the message when the request completes
Return value	GUID to identify the asynchronous operation.

9.3.7 Reset the drive object

The Reset function triggers a reset of the component which the Drive object represents.

Table 9-12: Resetting the Drive object

public void Reset()	
Parameter	**Meaning**
-	-

The ResetWithNcu function triggers a reset of the NCU in addition to a reset of the component which the Drive object represents.

Table 9-13: Resetting of the Drive object together with the NCU

public void ResetWithNcu()	
Parameter	**Meaning**
-	-

9.3.8 Canceling an asynchronous function

This function cancels active asynchronous calls.

Table 9-14: Canceling an asynchronous function

public void Cancel()	
Parameter	**Meaning**
-	-

9.3.9 Properties

AxisName

Axis name which is assigned to the Drive object.

Table 9-15: AxisName

public string AxisName { get; }

AxisNumber

Axis number which is assigned to the Drive object.

Table 9-16: AxisNumber

public long AxisNumber { get; }

Unrestricted © Siemens AG 2024 All Rights Reserved

BusAddress

Bus address of the drive object.

Table 9-17: BusAddress

public long BusAddress { get; }

DriveObjectId

Identifier of the Drive object.

Table 9-18: DriveObjectId

public long DriveObjectId { get; }

DriveObjectName

Name of the Drive object.

Table 9-19: DriveObjectName

public string DriveObjectName { get; }

DriveObjectTypeId

Identifier of the Drive object type.

Table 9-20: DriveObjectTypeId

public long DriveObjectTypeId { get; }

Enumerator DriveType

The DriveType enumerator contains the values of the possible drive component types which a Drive object can represent.

Table 9-21: Values of the DriveType enumerator

Value
ComModule
ControlUnit
Drive
I/O
LineModule
NotSpecified

DriveType

Type of the Drive object. This can be set to the values defined in the DriveType enumerator.

Table 9-22: DriveType

public DriveType DriveType { get; }

© Siemens AG 2024 All Rights Reserved

Server

Name of the NCU in a 1:N configuration.

Table 9-23: Server

public string Server { get; }

SlaveAddress

Device address of the drive object.

Table 9-24: SlaveAddress

public long SlaveAddress { get; }

9.3.10 Specifications for implementing delegates

SaveProgress

This function is called cyclically during active asynchronous saving of the Drive object in order to indicate progress.

Table 9-25: SaveProgress – specification for delegate implementation

public delegate void SaveProgress(System.Guid guid, int percentage)	
Parameter	**Meaning**
guid	GUID that was returned after calling asynchronous saving. It is used to identify the call.
percentage	Percentage value indicating the progress of the save operation.

SaveCompleted

This function is called once active asynchronous saving of the Drive object has been completed.

Table 9-26: SaveCompleted – specification for delegate implementation

public delegate void SaveCompleted(System.Guid guid, DrivesSvcStatus status)	
Parameter	**Meaning**
guid	GUID that was returned after calling asynchronous saving. It is used to identify the call.
status	Backup status, see Chapter: 9.5 Reference DriveSvcStatus

Unrestricted © Siemens AG 2024 All Rights Reserved

9.4 Reference DrivesSvc

9.4.1 Definitions

Overview

DriveSvc objects are used to list available Drive objects. Functions for resetting the NCU and the drives, as well as for saving the drive parameters to the CF card on the NCU are also supported.

Some of the calls are available in both synchronous and asynchronous forms. Synchronous calls block the thread in which the call takes place until the request has been completed. For asynchronous calls, after completion of the operation, an implementation of the associated delegate is called.

The class is defined in the "Siemens.Sinumerik.Operate.Services" namespace.

The class has two classes for exception handling:

- DrivesSvcBusyException
- DrivesSvcException

DrivesSvcBusyException is triggered if the service is already busy processing another call when the function is called. DrivesSvcException returns general problems associated with the use of the drive machine data service.

9.4.2 Creating a DrivesSvc object

The DrivesSvc class has two constructors.

Table 9-27: Constructors of the DrivesSvc class

public DrivesSvc() **public DrivesSvc(string server)**	
Parameter	**Meaning**
server	In 1:N constellations, specifies from/to which NCU access operations are executed. **NOTICE:** Function is not supported.

9.4.3 Releasing assigned resources

This function releases the resources used by a DriveSvc object again. If this is not done, the system will release the resources at some point once the object has become invalid.

Table 9-28: Release of resources being used

public void Dispose()	
Parameter	**Meaning**
-	-

© Siemens AG 2024 All Rights Reserved

9.4.4 Creating a list of Drive objects

Synchronous creation of a list of Drive objects

The ListDrives function returns an array with the Drive objects found in the NCU. The driveType parameter is used to define the corresponding drive type for the array.

Table 9-29: Synchronous creation of list of Drive objects

public Drive[] ListDrives() **public Drive[] ListDrives(DriveType driveType)**	
Parameter	**Meaning**
driveType	Drive object type to be associated with the drive objects in the list
Return value	Array with Drive objects
Possible exceptions	DrivesSvcException

Asynchronous creation of a list of Drive objects

The ListDrivesAsync is the asynchronous version of the ListDrives() function.

Its return value is a GUID which is subsequently used to identify the call.

Once the list has been created, the implementation of the following delegate is called:

- ListDrivesCompleted()

Table 9-30: Asynchronous creation of list of Drive objects

public System.Guid ListDrivesAsync(ListDrivesCompleted onCompleted) **public System.Guid ListDrivesAsync(DriveType driveType, ListDrivesCompleted onCompleted)**	
Parameter	**Meaning**
onCompleted	Delegate implementation for the message when the request completes
driveType	Drive object type to be associated with the drive objects in the list
Return value	GUID to identify the asynchronous operation.
Possible exceptions	DrivesSvcBusyException, DrivesSvcException

9.4.5 Accessing an individual drive object

The SelectDrive function is used for direct access to a Drive type Drive object. Identification is via the associated axis number or the associated axis name.

Table 9-31: Access to a specific drive

public Drive SelectDrive(int machineAxisNumber) **public Drive SelectDrive(string machineAxisName)**	
Parameter	**Meaning**
machineAxisNumber	Axis number assigned to the drive
machineAxisName	Axis name assigned to the drive
Return value	Drive object assigned to the transfer parameter
Possible exceptions	DrivesSvcException

Unrestricted © Siemens AG 2024 All Rights Reserved

9.4.6 Backing up drive objects

Synchronous saving of drive data

The Save and SaveControlUnits functions are used to save the data of all Drive objects synchronously to the CF card on the NCU, in the /user/sinamics/data directory.

Table 9-32: Synchronous saving of drive data

public void Save() **public void SaveControlUnits()**	
Parameter	**Meaning**
-	-

Asynchronous saving of drive data

The SaveAsync and SaveControlUnitsAsync functions are the asynchronous versions for saving drive data to the CF card.

Its return value is a GUID which is subsequently used to identify the call.

The functions call the implementation of the following delegates:

- SaveProgress()
- SaveCompleted()

! Important

To avoid inconsistencies when saving data, aborting the function with Cancel will not interrupt saving of the drive data to the card. Saving is only completed for the calling application.

Table 9-33: Asynchronous saving of drive data

public System.Guid SaveAsync(SaveProgress onProgress, SaveCompleted onCompleted) **public System.Guid SaveControlUnitsAsync(SaveProgress onProgress, SaveCompleted onCompleted)**	
Parameter	**Meaning**
onProgress	Delegate implementation for the progress message
onCompleted	Delegate implementation for the message when the request completes
Return value	GUID to identify the asynchronous operation.
Possible exceptions	DrivesSvcBusyException, DrivesSvcException

9.4.7 Triggering resets

Reset of the NCU

The ResetNCU function triggers a reset of the NCU.

Table 9-34:

public void ResetNcu()	
Parameter	**Meaning**
-	-

© Siemens AG 2024 All Rights Reserved

Reset of the NCU and the control units

The ResetNcuAndControlUnits function triggers a reset of the NCU and of the control units.

Table 9-35:

public void ResetNcuAndControlUnits()	
Parameter	**Meaning**
-	-

Preparing the control units for a reset

The PrepareControlUnitsForReset sets the control units so that they are also reset the next time the NCU is reset.

Table 9-36:

public void PrepareControlUnitsForReset()	
Parameter	**Meaning**
-	-

Reset of the control units

The ResetControlUnits function triggers a reset of the control units.

Table 9-37:

public void ResetControlUnits()	
Parameter	**Meaning**
-	-

9.4.8 Canceling asynchronous functions

This function cancels active asynchronous calls.

Table 9-38: Canceling asynchronous functions

public void Cancel()	
Parameter	**Meaning**
-	-

9.4.9 Specifications for implementing delegates

ListDrivesCompleted

The function is called if the ListDrivesAsync() function has been completed.

Table 9-39: ListDrivesCompleted – specification for delegate implementation

public delegate void ListDrivesCompleted(System.Guid guid, Drive[] drives)	
Parameter	**Meaning**
guid	GUID, which was returned when calling ListDrivesAsync(). It is used to identify the call.
drives	Array with Drive objects, list of found objects

Unrestricted © Siemens AG 2024 All Rights Reserved

SaveProgress

This function is called cyclically during active asynchronous saving of drive data objects in order to indicate progress.

Table 9-40: SaveProgress – specification for delegate implementation

public delegate void SaveProgress(System.Guid guid, int percentage)	
Parameter	**Meaning**
guid	GUID that was returned after calling asynchronous saving. It is used to identify the call.
percentage	Percentage value indicating the progress of the save operation.

SaveCompleted

This function is called once active asynchronous saving of the drive data objects has been completed.

Table 9-41: SaveCompleted – specification for delegate implementation

public delegate void SaveCompleted(System.Guid guid, DrivesSvcStatus status)	
Parameter	**Meaning**
guid	GUID that was returned after calling asynchronous saving. It is used to identify the call.
status	Backup status, see Chapter: 9.5 Reference DriveSvcStatus

© Siemens AG 2024 All Rights Reserved

9.5 Reference DriveSvcStatus

9.5.1 Definitions

Overview

A DriveSvcStatus object contains the result of a completed asynchronous request. The object only contains properties.

9.5.2 Properties

Ok

For a completed asynchronous request, indicates whether an error has occurred.

Values:
True: Job successfully executed
False: A problem occurred while executing the request. The problem can be analyzed in more detail via the "ErrorNo" property.

Table 9-42: Ok

public bool Ok { set; get; }

ErrorNo

Error number in the case of a problem for completed asynchronous requests.

Table 9-43: ErrorNo

public int ErrorNo { set; get; }

Unrestricted © Siemens AG 2024 All Rights Reserved

© Siemens AG 2024 All Rights Reserved

10

10 TraceDataRecorder service

Objective of the chapter

This chapter describes the interface of the trace recorder. The Trace recorder is used if control data needs to be recorded at high frequency. It is based on the "sITrace" component included in SINUMERIK Operate.

Unrestricted © Siemens AG 2024 All Rights Reserved

10.1 Introduction

10.1.1 Class model

The class model of the recorder consists of the following classes:

- TraceDataRecorder
- TraceDataRecorderBinData
- TraceDataRecorderDataset
- TraceDataRecorderErrorInterval
- TraceDataRecorderError
- TraceDataRecorderException

Two scenarios are possible with the recorder.

Procedure for scenario 1:

- Start data recording
- Recorded data is buffered
- Feedback after recording is completed
- Recorded data can be read

Procedure for scenario 2:

- Start data recording
- Data is received continuously during recording, the data is not buffered
- Feedback after recording is completed

Scenario 1 can be used for recording smaller data volumes. Scenario 2 is suitable for long recording sessions.

An object of the class TraceDataRecorder might be in status "Stopped", "Waiting for Trigger", or "Recording". The transition from one state to the next is shown in the figure below:

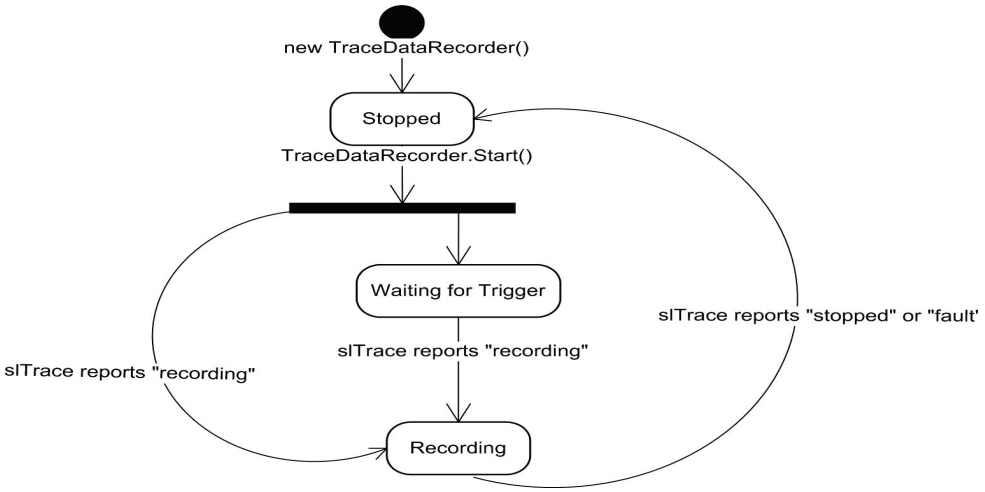


Fig. 10-1: State graph

© Siemens AG 2024 All Rights Reserved

Namespace

The classes are defined in the "Siemens.Sinumerik.Operate.Services" namespace. Consequently, this namespace must be integrated into the C# project at the start of the associated file:

```
using Siemens.Sinumerik.Operate.Services;
```

TraceDataRecorder

Objects that permit high-frequency recording of the time characteristics of NC variables and PLC variables can be created with class TraceDataRecorder.

TraceDataRecorderBinData

Class TraceDataRecorderBinData provides a data structure for binary data recording. This structure contains the data recorded in the time scale and any errors that occured during tracing.

TraceDataRecorderDataset

Class TraceDataRecorderDataset provides a structure for the binary recorded data recorded at any one particular time.

TraceDataRecorderErrorInterval

Class TraceDataRecorderErrorInterval provides a structure for errors that occured at any one particular time during binary recording.

TraceDataRecorderError

Class TraceDataRecorderError provides a structure for the errors that are returned if the trace procedure is aborted.

TraceDataRecorderException

Class TraceDataRecorderException offers a data structure for the exceptions that can occur in a TraceDataRecorder object.

Unrestricted © Siemens AG 2024 All Rights Reserved

10.2 Step-by-step examples

Overview

The following chapters provide a step-by-step demonstration of how the recorder is used in the two application scenarios.

These examples are shown in the following overview of examples.

Table 10-1: Overview of the examples

Application	Example	Chapter
Recording of data with buffering according to scenario 1	TraceDataRecorder_Scenario1	10.2.2
Continuous recording of data according to scenario 2	TraceDataRecorder_Scenario2	10.2.3

Only the points relevant to trace recording are described. The comments in the source code contain additional information.

10.2.1 Preparation

Overview

Preparatory measures are described in Chapter 2.1 ➔ "New Project". All of the items described there must be executed first.

Determining the XML setup

An XML setup file is required to parameterize the data recording. It can be created using the trace application in the "Diagnosis" area of the SINUMERIK Operate user interface. A session with the required parameters is configured and stored for this (e.g. "TenSeconds_Session.xml").

© Siemens AG 2024 All Rights Reserved

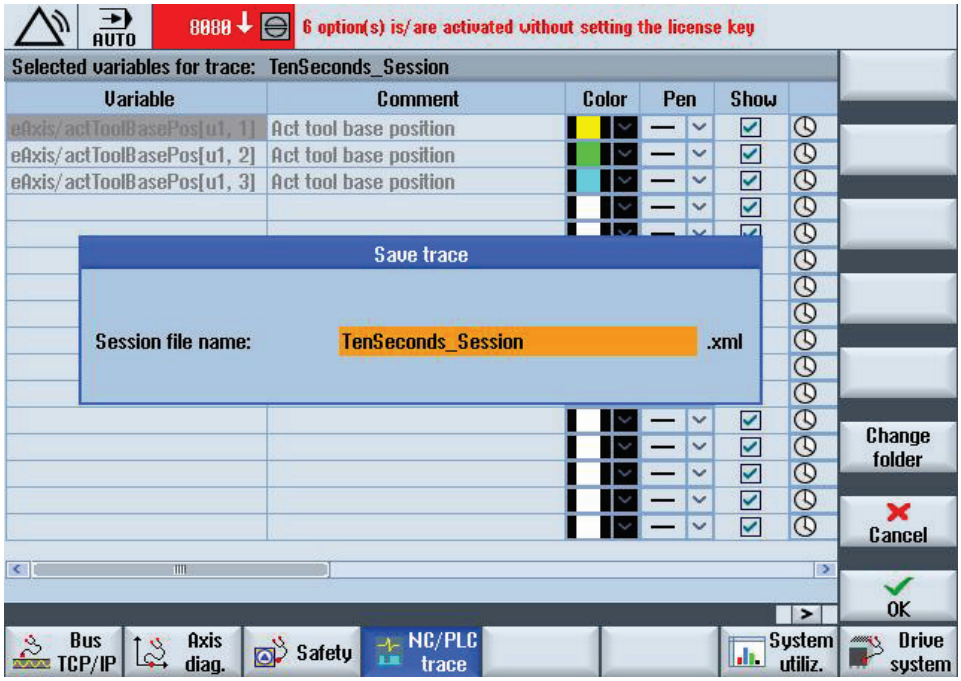


Fig. 10-2: Generating an XML structure for the trace recorder

The node "traceCaptureSetup_V20" is required from this session file.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!--XML file generated by SINUMERIK Solution Line Trace  (http://www.siemens.com)-->
<traceSession version="01.00.03.02">
<traceCaptureSetup_V20>
 <collectionSettings endless="false" autoRestart="false" appendData="false" atCapacity="discard" capacity="71680" deferOffload="false"
 <sessionSettings expirationPeriod="0" clientPrivilegeRead="password" clientPrivilegeManage="exclusive" sessionName="TenSeconds_Sessio
 <sampleRateSettings servoSampleRate="0.00200" driveSampleRate1="0.000125" ipo2SampleRate="0.00800" ipo1SampleRate="0.00800"/>
 <signalSettings>
  <signal dataType="double" displayRes="5" domain="NCU_1" eventChannel="1" name="/Channel/!SMA/actToolBasePos [u1, 1]" unitsType="posn
  <signal dataType="double" displayRes="5" domain="NCU_1" eventChannel="1" name="/Channel/!SMA/actToolBasePos [u1, 2]" unitsType="posn
  <signal dataType="double" displayRes="5" domain="NCU_1" eventChannel="1" name="/Channel/!SMA/actToolBasePos [u1, 3]" unitsType="posn
 </signalSettings>
 <triggerSettings useStopTrigger="false" useStartTrigger="false">
  <startTrigger patternTimeLimit="0.0100" combineLogic="and" delay="0.0000" patternQualifier="entered" repeatCount="1">
   <triggerCondition dataType="none" displayRes="0" mode="equalTo" level="0" level2="0.0" domain="" eventChannel="1" name="" key="t1"
  </startTrigger>
  <stopTrigger patternTimeLimit="0.0100" combineLogic="and" delay="0.5000" patternQualifier="entered" repeatCount="1">
   <triggerCondition dataType="none" displayRes="0" mode="equalTo" level="0" level2="0.0" domain="" eventChannel="1" name="" key="t1"
  </stopTrigger>
 </triggerSettings>
</traceCaptureSetup_V20>

<traceDisplaySetup_V20>
 <signals>
  <signal compositeDef="" key="s4" name="/Channel/!SMA/actToolBasePos [u1, 1]" unitsType="posn" lineWidth="1" waveformKey="w4" customU
  <signal compositeDef="" key="s1" name="/Channel/!SMA/actToolBasePos [u1, 2]" unitsType="posn" lineWidth="1" waveformKey="w1" customU
  <signal compositeDef="" key="s2" name="/Channel/!SMA/actToolBasePos [u1, 3]" unitsType="posn" lineWidth="1" waveformKey="w2" customU
 </signals>
 <graphWindows activeWindowKey="g1">
  <graphWindow title="Graph Window 1" annotations="0" multiYAxes="0" persistence="0" key="g1" tabContent="text">
   <position top="-345" state="maximized" height="6180" width="8940" left="-60"/>
   <colors labelColor="FFFFFF" backColor="000000" gridMajorColor="4F4F4F" gridMinorColor="2F2F2F"/>
   <xAxis timeScale="continuous" gridLines="true" scalingMode="linear"/>
   <yAxis gridLines="true" scalingMode="linear"/>
   <plot xMin="0" xMax="0" visible="true" key="w4" yMin="0.000000000000000" yMax="1.000000000000000" userOffset="0"/>
   <plot xMin="0" xMax="0" visible="true" key="w1" yMin="0.000000000000000" yMax="1.000000000000000" userOffset="0"/>
```

Fig. 10-3: Example of a stored trace session

Unrestricted © Siemens AG 2024 All Rights Reserved

The remaining entries must be deleted. After this, the file should look like this:

```
<traceCaptureSetup_V20>
 <collectionSettings endless="false" autoRestart="false" appendData="false" atCapacity="discard" capacity="71680" deferOffload="false" powerOnStart="false" timeLimit="10.0000"/>
 <sessionSettings expirationPeriod="0" clientPrivilegeRead="password" clientPrivilegeManage="exclusive" sessionName="TenSeconds_Session" archiveSetting="NCU"/>
 <sampleRateSettings servoSampleRate="0.00200" driveSampleRate1="0.000125" ipo2SampleRate="0.00800" ipo1SampleRate="0.00800"/>
 <signalSettings>
  <signal dataType="double" displayRes="5" domain="NCU_1" eventChannel="1" name="/Channel/!SMA/actToolBasePos [u1, 1]" unitsType="posn" key="s4" event="1" description="Aktueller Sollwe
  <signal dataType="double" displayRes="5" domain="NCU_1" eventChannel="1" name="/Channel/!SMA/actToolBasePos [u1, 2]" unitsType="posn" key="s1" event="1" description="Aktueller Sollwe
  <signal dataType="double" displayRes="5" domain="NCU_1" eventChannel="1" name="/Channel/!SMA/actToolBasePos [u1, 3]" unitsType="posn" key="s2" event="1" description="Aktueller Sollwe
 </signalSettings>
 <triggerSettings useStopTrigger="false" useStartTrigger="false">
  <startTrigger patternTimeLimit="0.0100" combineLogic="and" delay="0.0000" patternQualifier="entered" repeatCount="1">
   <triggerCondition dataType="none" displayRes="0" mode="equalTo" level="0" level2="0.0" domain="" eventChannel="1" name="" key="t1" event="1" description="" bitMask="00000000"/>
  </startTrigger>
  <stopTrigger patternTimeLimit="0.0100" combineLogic="and" delay="0.5000" patternQualifier="entered" repeatCount="1">
   <triggerCondition dataType="none" displayRes="0" mode="equalTo" level="0" level2="0.0" domain="" eventChannel="1" name="" key="t1" event="1" description="" bitMask="00000000"/>
  </stopTrigger>
 </triggerSettings>
</traceCaptureSetup_V20>
```

Fig. 10-4: XML setup for use in the TraceDataRecorder

Important

In order to be able to receive data while recording, parameter "deferOffload" = false must be set in the XML configuration.

Such an XML setup (TenSeconds_Def.xml) is included with the examples. With it, the positions of the tool adapters with axis indices 1, 2, and 3 are recorded in the IPO cycle clock in channel 1 for ten seconds.

Note

The following points should be observed:

1. The attributes "ipo1SampleRate" and "ipo2SampleRate" in the XML setup must be an integer multiple of the IPO clock cycle of the control.
2. The attribute "servoSampleRate" in the XML setup must be an integer multiple of the servo clock cycle of the control.

10.2.2 Example 1

Overview

The following example shows step-by-step how to trace and buffer control data with the trace recorder and subsequently read out data after completing trace operation.

In the example, you choose whether the data is to be traced as XML or binary format via the two option buttons. If you choose the option binary, the data will be processed as a CSV file, for example. In two text fields, you define the setup XML that contains the definitions for the trace operation and the destination directory for the file containing the traced data. Using the six buttons on the left-hand side of the form, you can execute the individual steps of the trace procedure.

The status bar at the lower edge informs you about the success of the calls.

© Siemens AG 2024 All Rights Reserved

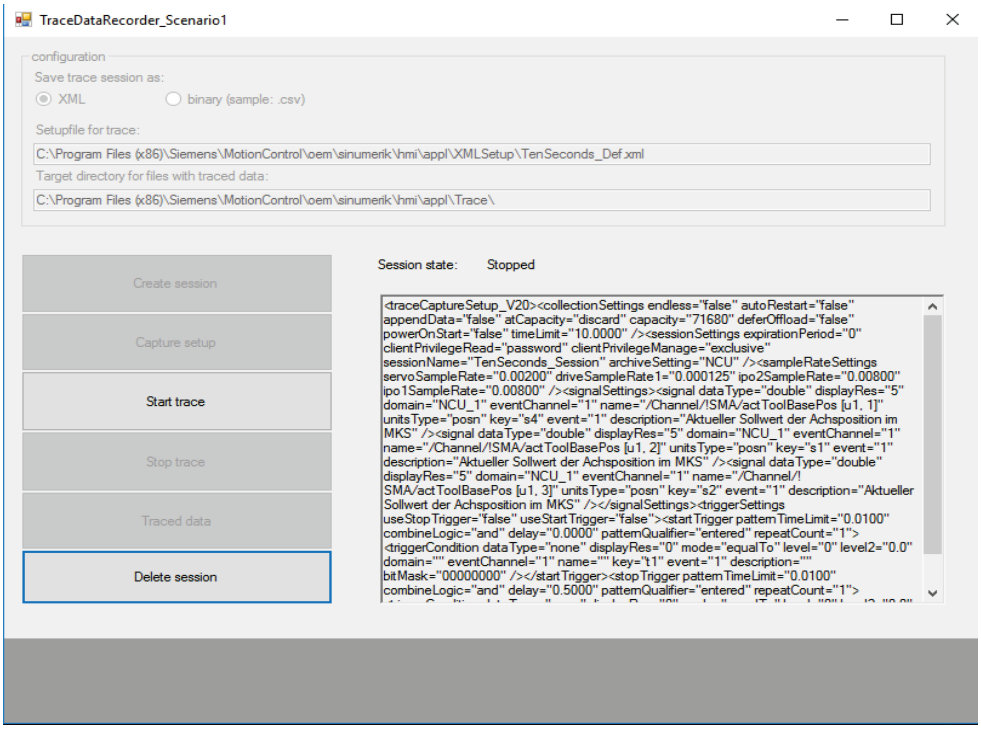


Fig. 10-5: Data recording with the trace recorder according to scenario 1

The diagram below shows the sequence of calls in example 1:

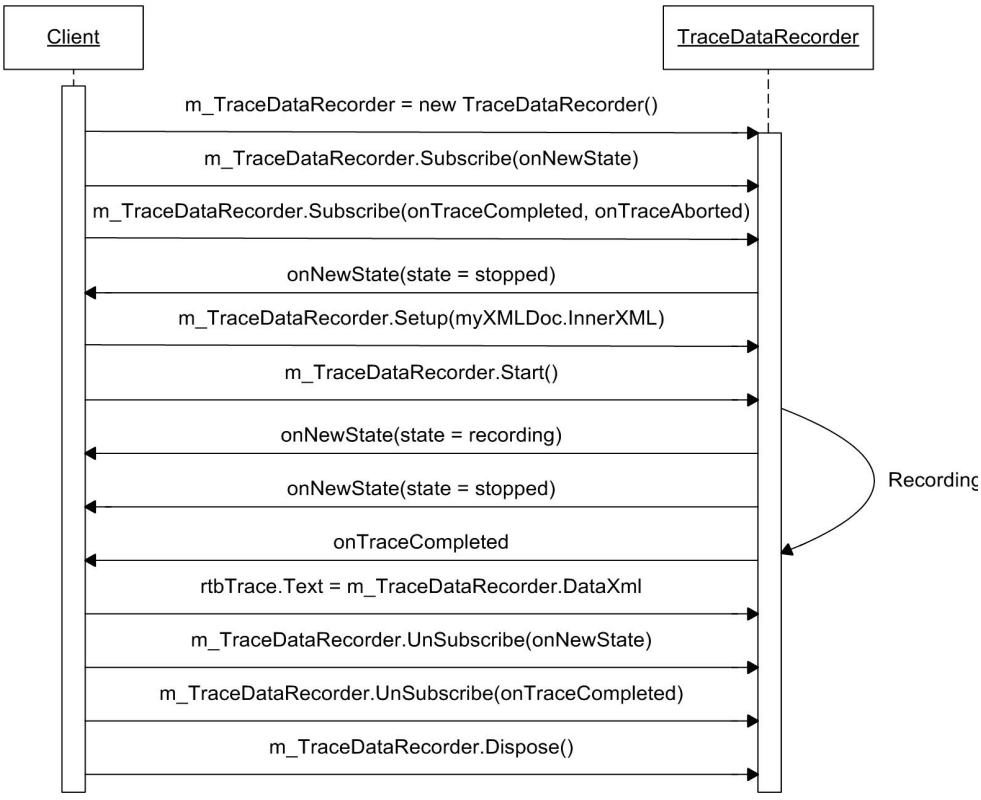


Fig. 10-6: Sequence of calls in the example of scenario 1

Unrestricted © Siemens AG 2024 All Rights Reserved

Step 1

Create the required member variables. A variable of type TraceDataRecorder is required.

```
TraceDataRecorder m_TraceDataRecorder = null;
```

Step 2

Create the TraceDataRecorder object in function cmdCreateSession_Click.

```
m_TraceDataRecorder = new TraceDataRecorder();
```

Step 3

Specify the format (XML structure or binary format) with which the data is to be recorded.

```
if (lblXML.Checked == true)
{
   m_TraceDataRecorder.ReportsXml = true;
}
else
{
   m_TraceDataRecorder.ReportsXml = false;
}
```

Step 4

Subscribe to the required events. The delegates are called via these events. In the example, state changes are indicated by delegate onNewState. The delegates onTraceCompleted and onTraceAborted are called as soon as the trace action has been completed or is aborted.

```
m_TraceDataRecorder.Subscribe(onNewState);
m_TraceDataRecorder.Subscribe(onTraceCompleted, onTraceAborted);
```

Step 5

Implement the delegates required for processing the events.

Important

Because the delegate is called from another thread, not all display elements can be accessed. Consequently, it is desirable to call a second delegate with the same signature with which changes can be performed on the user interface.

In the implemented delegate function, a change is sent to the same delegate signature, now in the GUI thread.

Implementation of delegate onNewState(). This is called for a state change of the trace recorder and outputs the current state of the TraceDataRecorder object as a label text.

© Siemens AG 2024 All Rights Reserved

```
private void onNewState(TraceDataRecorderState state)
{
    // are we in GUI thread (main) ?
    // No we are not in GUI thread => send message to GUI thread
    if (lblSessionState.InvokeRequired)
    {
        StateChanged stateChangedDelegate = new StateChanged(onNewState);
        BeginInvoke(stateChangedDelegate, new Object[] { state });
    }
    else
    {
        lblSessionState.Text = state.ToString();
    }
}
```

Implementation of delegate onTraceCompleted(). This delegate is called when data recording is complete. The buffered recording values are read out and stored in a file. If the data was recorded in XML format it will be stored directly in an XML file. If data was recorded in binary format, the supplied data structure will be resolved and the data stored in a CSV file, for example.

```
private void onTraceCompleted()
{
  // are we in GUI thread (main) ?
  // No we are not in GUI thread => send message to GUI thread
  if (cmdCreateSession.InvokeRequired)
  {
      DataCollectionCompleted traceCompletedDelegate = new
                                  DataCollectionCompleted(onTraceCompleted);
      BeginInvoke(traceCompletedDelegate, new Object[] { });
  }
  else
  {
    if (m_TraceDataRecorder.ReportsXml == true)
    {
      //save XML-file with traced data
      string sDestination = m_sTracePath +
                        DateTime.Now.ToString("yyyyMMddhhmmssfff") + ".xml";
      XmlDocument myTraceDoc = new XmlDocument();
      myTraceDoc.InnerXml = m_TraceDataRecorder.DataXml;
      myTraceDoc.Save(sDestination);
    }
    else
    {
      if (m TraceDataRecorder.DataBin.Error.Count() == 0)
      {
        //no error occurred during execution of trace
        if (m_TraceDataRecorder.DataBin.Data.Count() > 0)
        {
          //save binary traced data as a CSV-file
           ...

          //write traced data to file
          for (int i = 0; i < m TraceDataRecorder.DataBin.Data.Count(); i++)
          {
            s2Write =
                  m_TraceDataRecorder.DataBin.Data[i].TimeOffset.ToString();

            for (int iCountIds = 0; iCountIds <
               m TraceDataRecorder.DataBin.Data[i].Ids.Count(); iCountIds++)
            {
              s2Write = s2Write + ";" + m_TraceDataRecorder.DataBin.Data[i].
                                                  Ids[iCountIds].ToString();
            }

            for (int iCountValues = 0; iCountValues <
                     m_TraceDataRecorder.DataBin.Data[i].Values.Count();
                     CountValues++)
            {
              s2Write = s2Write + ";" + m_TraceDataRecorder.DataBin.Data[i].
                                               Values[iCountValues].ToString();
            }
```

Unrestricted © Siemens AG 2024 All Rights Reserved

```
            ...
          }
        }
      }
      else
      {
        //error during trace
        for (int i = 0; i < m_TraceDataRecorder.DataBin.Error.Count();i++)
        {
          sError = sError + "TimeOffset: " +
          m_TraceDataRecorder.DataBin.Error[i].TimeOffset.ToString() +
             " ErrorNo: " +
            m_TraceDataRecorder.DataBin.Error[i].ErrorNo.ToString() + "\n";
         }

         setStatus(sError);
      }
    }
  }
}
```

Implementation of delegate onTraceAborted(). This delegate is called if data recording is aborted by the system. If the trace operation is aborted, the delegate enters the returned error number and the associated text in the status bar.

```
private void onTraceAborted(TraceDataRecorderError err)
{
    // are we in GUI thread (main) ?
    // No we are not in GUI thread => send message to GUI thread
    if (cmdCreateSession.InvokeRequired)
    {
        DataCollectionAborted traceAbortedDelegate = new
                                    DataCollectionAborted(onTraceAborted);
        BeginInvoke(traceAbortedDelegate, new Object[] { err });
    }
    else
    {
        String sReason = "";
        sReason = "Tracing was aborted with errornumber " + err.ErrorNo +
                                  " and following message: " + err.Message;
        setStatus(sReason);
    }
}
```

Step 6

Read-in setup file with the trace definitions in function cmdCaptureSetup_Click().

```
m_TraceDataRecorder.Setup(myXMLDoc.InnerXML);
```

Step 7

Start data recording in function cmdStartTrace_Click().

```
m_TraceDataRecorder.Start();
```

Step 8

Stop data recording in function cmdStopTrace_Click(). With this method, data recording can be stopped at any time irrespective of the setup file.

```
m_TraceDataRecorder.Stop();
```

Step 9

Read-out buffered recorded data in function cmdTracedData_Click().

```
rtbTrace.Text = m_TraceDataRecorder.DataXml.ToString();
```


© Siemens AG 2024 All Rights Reserved

Step 10

Unsubscribe the subscribed events in function cmdDeleteSession_Click().

```
m_TraceDataRecorder.UnSubscribe(onNewState);
m_TraceDataRecorder.UnSubscribe(onTraceCompleted, onTraceAborted);
```

Step 11

Destroy the TraceDataRecorder object in function cmdDeleteSession_Click().

```
m_TraceDataRecorder.Dispose();
```

10.2.3 Example 2

Overview

The following example shows how to continuously record control data with the trace recorder. Example 2 is structured in the same way as the previous example 1. Therefore, only the additional steps that deal with continuous recording are described below.

In the example, you choose whether the data is to be traced as XML or binary format via the two option buttons. If you choose the option binary, the data will be processed as a CSV file, for example. In two text fields, you define the setup XML that contains the definitions for the trace operation and the destination directory for the file containing the traced data. Using the five buttons on the left-hand side of the form, you can execute the individual steps of the trace procedure.

The status bar at the lower edge informs you about the success of the calls.

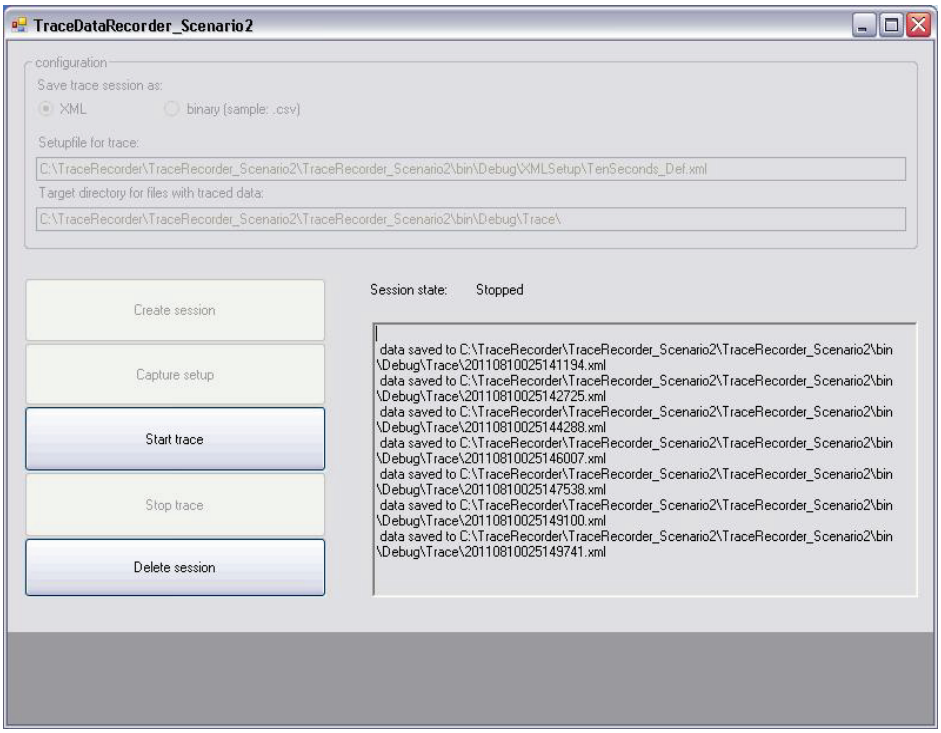


Fig. 10-7: Continuous data recording with the trace recorder according to scenario 2

Unrestricted © Siemens AG 2024 All Rights Reserved

The diagram below shows the sequence of calls in example 2:

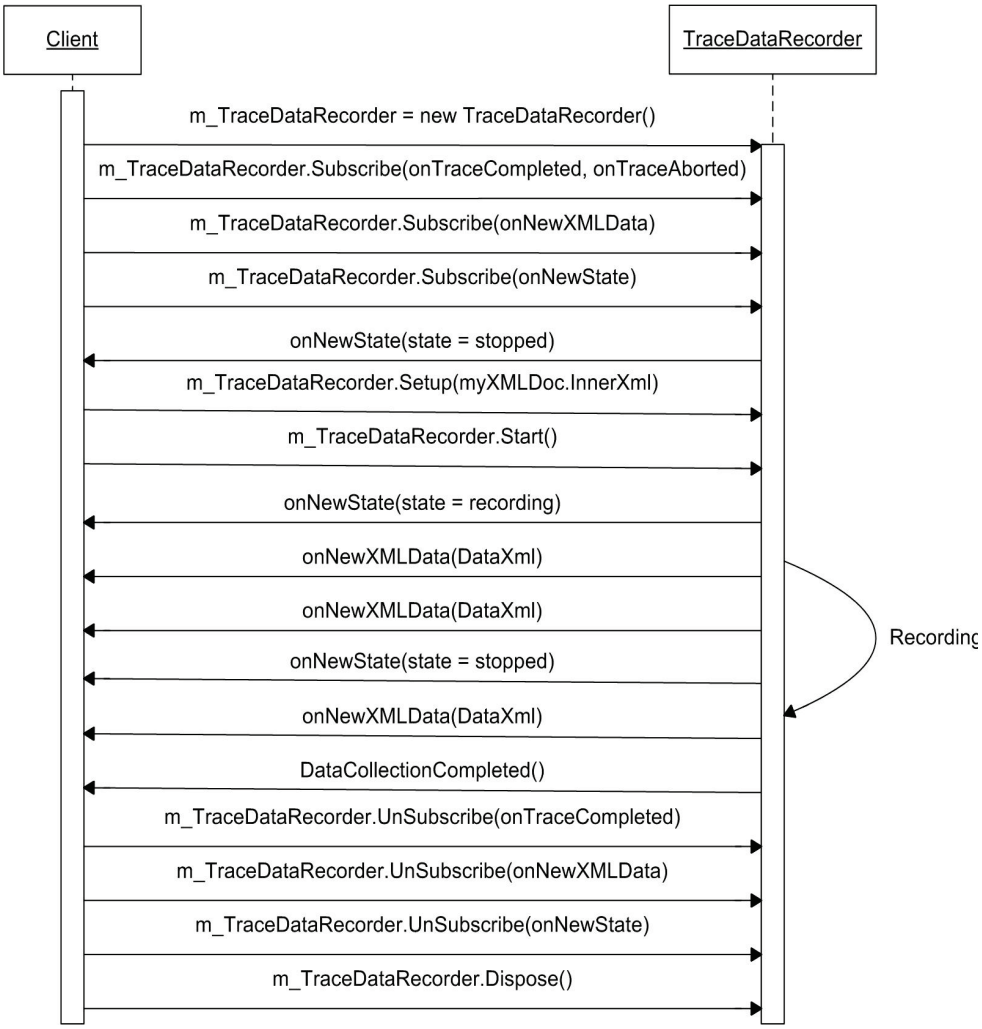


Fig. 10-8: Sequence of calls in the example of scenario 2

Step 1

Additional subscription to event NewXMLData or NewBinaryData in function cmdCreateSession_Click(). Delegates are called via theses events if new data is pending from a recording. If the data is recorded in XML format, you need the NewXMLData event, otherwise you need the NewBinaryData event. In the example, the events are triggered if approximately 200 new data records are pending.

```
if (lblXML.Checked == true)
{
   m_TraceDataRecorder.Subscribe(onNewXMLData,200);
}
else
{
   m_TraceDataRecorder.Subscribe(onNewBinaryData,200);
}
```


© Siemens AG 2024 All Rights Reserved

Step 2

Implement the delegates that were called via the NewData events.

Important

Because the delegate is called from another thread, not all display elements can be accessed. Consequently, it is desirable to call a second delegate with the same signature with which changes can be performed on the user interface.

In the implemented delegate function, a change is sent to the same delegate signature, now in the GUI thread.

Implementation of delegate onNewXMLData(). This delegate is called if new data is available that was recorded in XML format. The data is provided in the DataXML parameter and written to an XML file.

```
private void onNewXMLData(string DataXml)
{
    // are we in GUI thread (main) ?
    // No we are not in GUI thread => send message to GUI thread
    if (cmdCreateSession.InvokeRequired)
    {
        NewXmlData newDataXMLDelegate = new NewXmlData(onNewXMLData);
        BeginInvoke(newDataXMLDelegate, new Object[] { DataXml });
    }
    else
    {
        //save XML-file with traced data
        string sDestination = m_sTracePath +
                      DateTime.Now.ToString("yyyyMMddhhmmssffff") + ".xml";
        XmlDocument myTraceDoc = new XmlDocument();
        myTraceDoc.InnerXml = DataXml;
        myTraceDoc.Save(sDestination);
        rtbTrace.Text = rtbTrace.Text + "\n data saved to " + sDestination;
    }
}
```

Implementation of delegate onNewBinaryData(). This delegate is called if new data is available that was recorded in binary format. The data is provided in the newData parameter and written, for example, to a CSV file.

```
private void onNewBinaryData(TraceDataRecorderBinData newData)
{
  // are we in GUI thread (main) ?
  // No we are not in GUI thread => send message to GUI thread
  if (cmdCreateSession.InvokeRequired)
  {
    NewBinaryData newBinaryDataDelegate = new
                                          NewBinaryData(onNewBinaryData);
    BeginInvoke(newBinaryDataDelegate, new Object[] { newData });
  }
  else
  {
    if (newData.Error.Count() == 0)
    {
      //no error occurred during execution of trace
      if (newData.Data.Count() > 0)
      {
         //save binary traced data as a CSV-file
      }
    }
    else
    {
```

Unrestricted © Siemens AG 2024 All Rights Reserved

```
      //error during trace
      string sError = "There were errors during the trace of binary data.";

      for (int i = 0; i < m_TraceDataRecorder.DataBin.Error.Count(); i++)
      {
        sError = sError + "TimeOffset: " +
             m_TraceDataRecorder.DataBin.Error[i].TimeOffset.ToString() +
             " ErrorNo: " +
             m_TraceDataRecorder.DataBin.Error[i].ErrorNo.ToString() + "\n";
      }

      setStatus(sError);
  }
}
```

Step 3

Unsubscribe the NewData events in function cmdDeleteSession_Click().

```
if (m_TraceDataRecorder.ReportsXml == true)
{
   m_TraceDataRecorder.UnSubscribe(onNewXMLData);
}
else
{
   m_TraceDataRecorder.UnSubscribe(onNewBinaryData);
}
```


© Siemens AG 2024 All Rights Reserved

10.3 Reference TraceDataRecorder

Overview

The TraceDataRecorder class provides the option of recording control data at high frequency via the trace server.

This class is defined in the "Siemens.Sinumerik.Operate.Services" namespace.

10.3.1 Creating an instance of TraceDataRecorder

Three constructors are available for creating an instance of TraceDataRecorder.

Table 10-2: Creating a TraceDataRecorder object

public TraceDataRecorder() **public TraceDataRecorder(string xmlSetup)** **public TraceDataRecorder(string xmlSetup, string ncu)**	
Parameter	**Meaning**
xmlSetup	Parameterization of a recording in XML format
ncu	currently not used

10.3.2 Releasing assigned resources

The Dispose function releases the resources assigned by a TraceDataRecorder object again.

Table 10-3: Re-releasing occupied resources

public void Dispose()	
Parameter	**Meaning**
Possible exceptions	TraceDataRecorderException

10.3.3 Resetting to initial state

The Reset function sets the recorder back to its initial state, i.e. an object without parameterization and without data is available.

Table 10-4: Reset to initial state

public void Reset()	
Parameter	**Meaning**
Possible exceptions	TraceDataRecorderException

10.3.4 Transferring the parameterization

With the setup function, an XML parameterization is passed for recording a trace.

Table 10-5: Transferring the parameterization

public void Setup(string xmlSetup)	
Parameter	**Meaning**
xmlSetup	Parameterization of a recording in XML format
Possible exceptions	TraceDataRecorderException

Unrestricted © Siemens AG 2024 All Rights Reserved

10.3.5 Starting the recording

The trace recording is started by calling the function Start.

Table 10-6: Starting the trace

public void Start()	
Parameter	**Meaning**
Possible exceptions	TraceDataRecorderException

10.3.6 Stopping the recording

The current trace recording is stopped by calling the function Stop.

Table 10-7: Stopping the trace

public void stop()	
Parameter	**Meaning**
Possible exceptions	TraceDataRecorderException

10.3.7 Subscribing to events

Notifications from the trace recorder are subscribed with the subscribe functions.

Table 10-8: Subscribing to events

public void Subscribe(NewBinaryData onNewData) **public void Subscribe(NewBinaryData onNewData, int MinimumRecords)** **public void Subscribe(NewXmlData onNewData)** **public void Subscribe(NewXmlData onNewData, int MinimumRecords)** **public void Subscribe(StateChanged onNewState)** **public void Subscribe(DataCollectionCompleted onCompleted,** **DataCollectionAborted onAborted)**	
Parameter	**Meaning**
onNewData	Delegate implementation for the event that is initiated when new data is available. Available in variants for XML and for binary data.
MinimumRecords	Defines the number of data records that should exist when an onNewData is initiated
onNewState	Delegate implementation for the event on a state change
onCompleted	Delegate implementation for the event after end of recording
onAborted	Delegate implementation for the event when a recording is aborted
Possible exceptions	TraceDataRecorderException

10.3.8 Unsubscribing events

Existing subscriptions to messages from the recorder are ended with the unsubscribe functions.

© Siemens AG 2024 All Rights Reserved

Table 10-9: Unsubscribing a subscription to events

public void UnSubscribe(NewBinaryData onNewData) public void UnSubscribe(NewXmlData onNewData) public void UnSubscribe(StateChanged onNewState) public void UnSubscribe(DataCollectionCompleted onCompleted, DataCollectionAborted onAborted)	
Parameter	**Meaning**
onNewData	Delegate implementation for the event that is initiated when new data is available. Available in variants for XML and for binary data.
onNewState	Delegate implementation for the event on a state change
onCompleted	Delegate implementation for the event after end of recording
onAborted	Delegate implementation for the event when a recording is aborted
Possible exceptions	TraceDataRecorderException

10.3.9 Specifications for implementing delegates

DataCollectionAborted

This function is called if the trace is aborted by the system.

Table 10-10: DataCollectionAborted – specification for delegate implementation

public delegate void DataCollectionAborted(TraceDataRecorderError error)	
Parameter	**Meaning**
error	Structure with the returned error that caused the data recording to be aborted.

DataCollectionCompleted

This function is called when data recording is complete.

Table 10-11: DataCollectionCompleted – specification for delegate implementation

public delegate void DataCollectionCompleted()	
Parameter	**Meaning**
-	-

NewBinaryData

This function is called if new data recorded in binary format is available.

Table 10-12: NewBinaryData – specification for delegate implementation

public delegate void NewBinaryData(TraceDataRecorderBinData DataBin)	
Parameter	**Meaning**
DataBin	New data returned from the data recording in binary format.

NewXmlData

This function is called if new data recorded in XML format is available.

Table 10-13: NewXmlData – specification for delegate implementation

public delegate void NewXmlData(string DataXml)	
Parameter	**Meaning**
DataXml	New data returned from the data recording in XML format.

Unrestricted © Siemens AG 2024 All Rights Reserved

StateChanged

This function is called if the status of the trace recorder changes.

Table 10-14: StateChanged – specification for delegate implementation

public delegate void StateChanged(TraceDataRecorderState state)	
Parameter	**Meaning**
state	New status of the trace recorder.

10.3.10 Properties

DataBin

Recorded data in binary format. Contains the data after a recording according to scenario 1.

Table 10-15: Buffered data in binary format

public TraceDataRecorderBinData DataBin { get; }

DataXml

Recorded data in an XML structure. Contains the data after a recording according to scenario 1.

Table 10-16: Buffered data in XML format

public string DataXML { get; }

Ncu

Name of the NCU. Not yet used.

Table 10-17: Name of the Ncu

public string Ncu { set; get; }

ReportsXml

Specifies the format of the result data. The default value "true" stands for data in XML format, the value "false" stands for data in binary format. The binary data format is more powerful and is recommended for applications with high volumes of data.

Table 10-18: Specifying the output format for trace

public bool ReportsXml { set; get; }

© Siemens AG 2024 All Rights Reserved

Enumerator TraceDataRecorderState

The TraceDataRecorderState enumerator contains the values of the possible states that the trace recorder can assume.

Table 10-19: Values of the enumerator TraceDataRecorderState

Parameter
Recording
Stopped
WaitingForTrigger

State

Contains the current state of the trace recorder.

Table 10-20: State

public TraceDataRecorderState State { get; }

XmlSetup

XML definition of the data to be recorded by the recorder.

Table 10-21: Definition file for trace

public string XmlSetup { get;}

Unrestricted © Siemens AG 2024 All Rights Reserved

10.4 Reference TraceDataRecorderBinData

Overview

If data is recorded in binary format, it is stored in an object of class TraceDataRecorderBinData.

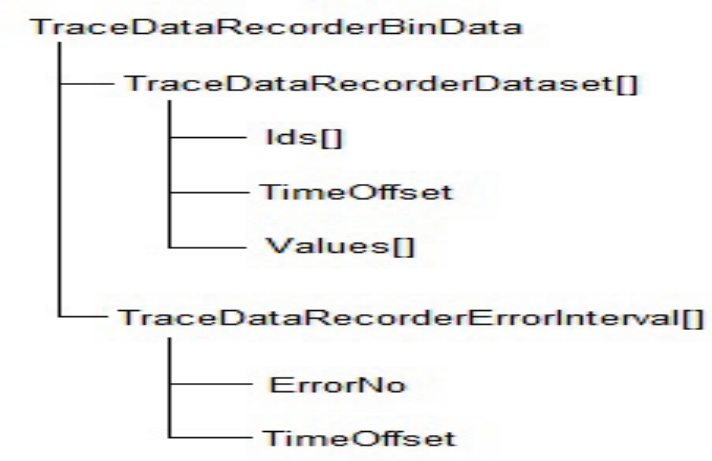


Fig. 10-9: Structure of the binary recorded data

10.4.1 Creating a TraceDataRecorderBinData object

A constructor is provided for creating a TraceDataRecorderBinData object.

Table 10-22: Constructor of class TraceDataRecorderBinData

public TraceDataRecorderBinData	
Parameter	**Meaning**
-	-

10.4.2 Array with recorded values

The class has a one-dimensional array of type TraceDataRecorderDataset, which contains the recorded values.

Table 10-23: Data in TraceDataRecorderBinData

public TraceDataRecorderDataset[] Data

10.4.3 Array with errors that have occurred

A one-dimensional array of type TraceDataRecorderErrorInterval contains any error messages that might have occurred during a recording.

Table 10-24: Error in TraceDataRecorderBinData

public TraceDataRecorderErrorInterval[] Error

© Siemens AG 2024 All Rights Reserved

10.5 Reference TraceDataRecorderDataset

Overview

If data is recorded in binary format, a TraceDataRecorderDataset object containing the recorded data is created for each recording instant.

Table 10-25: Constructor of class TraceDataRecorderDataset

public TraceDataRecorderDataset()	
Parameter	**Meaning**
-	-

Identifier

The variable Ids is a one-dimensional array with the identifiers of items recorded at a specific time.

Table 10-26: Array with identifiers

public int[] Ids

Time frame

The TimeOffset variable contains the time difference determined for the values in the TraceDataRecorderDataset object relative to the start of recording.

Table 10-27: Time frame of the recording

public double TimeOffset

Recorded values

The variable Values is a one-dimensional array containing the values for the identifiers recorded at a specific time.

Table 10-28: Array with recorded values

public object[] Values

Unrestricted © Siemens AG 2024 All Rights Reserved

10.6 Reference TraceDataRecorderErrorInterval

Overview

If errors occur during recording of data in binary format, they are stored in a TraceDataRecorderErrorInterval object.

Table 10-29: Constructor of class TraceDataRecorderErrorInterval

public TraceDataRecorderErrorInterval()	
Parameter	**Meaning**
-	-

Error number

The ErrorNo variable contains the error number that occurred on instant TimeOffset during binary recording.

Table 10-30: Error number

public long ErrorNo

Time frame

The TimeOffset variable contains the time difference determined for the values in the TraceDataRecorderDataset object relative to the start of recording.

Table 10-31: Time frame of the recording

public double TimeOffset

© Siemens AG 2024 All Rights Reserved

10.7 Reference TraceDataRecorderError

Overview

An object of type TraceDataRecorderError is passed as a parameter to delegate DataCollectionAborted. It contains the error message that caused the data recording to be aborted.

Table 10-32: Constructor of class TraceDataRecorderError

public TraceDataRecorderError	
Parameter	**Meaning**
-	-

Properties

The ErrorNo property contains the error number of the problem that occurred.

Table 10-33: Error number

public int ErrorNo

The property message contains the message text for the problem that occurred during data recording.

Table 10-34: Message text

public string Message

Unrestricted © Siemens AG 2024 All Rights Reserved

10.8 Reference TraceDataRecorderException

Overview

The TraceDataRecorderException class is used for the exceptions that an object of type TraceDataRecorder can report.

Table 10-35: Exception for TraceDataRecorder

public TraceDataRecorderException(string message, int errorNo, System.Exception innerException) **public TraceDataRecorderException(string message)**	
Parameter	**Meaning**
message	Error message for exception
errorNo	Error number for exception
innerException	Exception of SITrace that caused the TraceDataRecorderException

Properties

Error number that is output by a TraceDataRecorder object.

Table 10-36: Error number

public int ErrorNumber { get; }

The trace recorder sometimes outputs its own internal error numbers as follows:

TRACE_DATA_RECORDER_SETUP_FAILED = 180000
TRACE_DATA_RECORDER_START_FAILED = 180001
TRACE_DATA_RECORDER_STOP_FAILED = 180002
TRACE_DATA_RECORDER_RESET_FAILED = 180003

If the cause of the error is to be studied in more detail, it may be useful to evaluate the InnerException that is output by SITrace.
The InnerException.Message property, in particular may contain further error codes in its texts that are explained in more detail in the SITrace documentation (Referenz_SITrace.pdf) or the error message header file of SITrace (SItraceerrenumgrp.h).

© Siemens AG 2024 All Rights Reserved

11

11 ToolManagement service

Objective of the chapter

This chapter describes the interface of the ToolManagement Service. It enables changes to be tracked with regard to tools.

Note

Only one instance of the ToolManagement service is permissible per process.

Unrestricted © Siemens AG 2024 All Rights Reserved

11.1 Introduction

11.1.1 Class model

The class model of the ToolManagement service primarily consists of the following class:

- ToolMngmntSvc

Namespace

The class is defined in the "Siemens.Sinumerik.Operate.Services" namespace. Consequently, this namespace must be integrated into the C# project at the start of the associated file:

```
using Siemens.Sinumerik.Operate.Services;
```

class ToolMngmntSvc

It is possible to subscribe to changes with regard to tools using ToolMngmntSvc objects:

- Creating/deleting tools
- Value or status changes on tools
- Loading/unloading/relocating tools

© Siemens AG 2024 All Rights Reserved

11.2 Step-by-step examples

Overview

The following chapters show the use of the ToolManagement service step-by-step.

Table 11-1: Overview of the examples

Application	Example	Chapter
Tracking changes	ToolMngmntSvcToolAction	11.2.2

Only the points relevant for the ToolManagement service are described. The comments in the source code contain more detailed information.

11.2.1 Preparation

Overview

Preparatory measures are described in Chapter 2.1 ➔ "New Project". All of the items described there must be executed first.

11.2.2 Tracking changes

Overview

The following example shows step-by-step how changes with regard to tools can be tracked with the ToolManagement service.

In the example, the notifications are activated after "Subscribe" has been pressed. All changes with regard to tools are displayed in a list and updated accordingly. Pressing "Unsubscribe" ends the subscription.

The status bar at the lower edge informs you about the success of the calls.

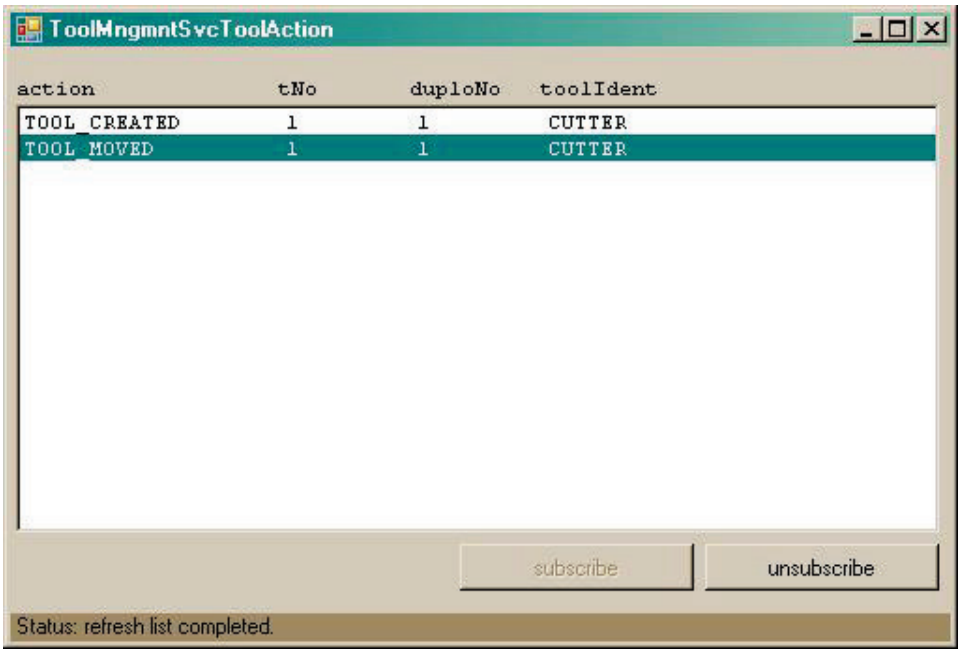


Fig. 11-1: ToolMngmntSvcToolAction

Unrestricted © Siemens AG 2024 All Rights Reserved

Step 1

Create the required member variables.

```
ToolMngmntSvc m_ToolMngmntSvc = null;
Guid m_ToolMngmntListGuid = Guid.Empty;
```

Step 2

Create the ToolMngmntSvc object in the constructor of the form. The TO area to be monitored is required as parameter.

```
m_ToolMngmntSvc = new ToolMngmntSvc(1);
```

Step 3

Call the `Subscribe` function to receive changes with regard to tools. In this case, only the delegate is required as the transfer parameter, in which the changes are to be signaled (CbNewToolInfo).

```
m_ToolMngmntListGuid = m_ToolMngmntSvc.Subscribe(CbNewToolInfo);
```

Step 4

Implement the delegate function.

Important

Because the delegate is called from another thread, not all display elements can be accessed. Consequently, it is desirable to call a second delegate with the same signature with which changes can be performed on the user interface.

In the implemented delegate function, a change is sent to the same delegate signature, now in the GUI thread. The changes with regard to tools are output here. They are written to a list box.

```
private void CbNewToolInfo(Guid guid, ToolMngmntSvc.ToolInfo[] toolInfo)
{
   // are we in GUI thread (main) ?
   // No we are not in GUI thread => send message to GUI thread
   if (lstAction.InvokeRequired)
   {
       ToolMngmntSvc.NewToolInfo newToolInfoDelegate = new
                                  ToolMngmntSvc.NewToolInfo(CbNewToolInfo);
       BeginInvoke(newToolInfoDelegate, new Object[] { guid, toolInfo });
   }
   else
   {
       foreach (ToolMngmntSvc.ToolInfo myToolInfo in toolInfo)
       {
           // add a new tool information
           lstAction.Items.Add(myToolInfo.action.ToString().PadRight(20) +
                               myToolInfo.tNo.ToString().PadRight(10) +
                               myToolInfo.duploNo.ToString().PadRight(10) +
                               myToolInfo.toolIdent);
       }
   }
}
```

Step 5

If the functionality is no longer required, `UnSubscribe` is called.

```
m_ToolMngmntSvc.UnSubscribe(CbNewToolInfo);
```


© Siemens AG 2024 All Rights Reserved

11.3 Reference ToolMngmntSvc

Overview

It is possible to subscribe to changes with regard to tools using ToolMngmntSvc objects:

- Creating/deleting tools
- Value or status changes on tools
- Loading/unloading/relocating tools

This class is defined in the "Siemens.Sinumerik.Operate.Services" namespace.

11.3.1 Creating a ToolMngmntSvc object

The following constructor is available to create a ToolMngmntSvc object.

Table 11-2: Creating a ToolMngmntSvc object

public ToolMngmntSvc(int ToAreaNo);	
Parameter	**Meaning**
ToAreaNo	TO area
Possible exceptions	ToolMngmntSvcException

11.3.2 Releasing assigned resources

The Dispose function releases the resources assigned by a ToolMngmntSvc object again.

Table 11-3: Re-releasing occupied resources

public void Dispose()	
Parameter	**Meaning**
Possible exceptions	ToolMngmntSvcException

11.3.3 Functions

Subscribe to the tool events

Table 11-4: Subscribe to the tool events

public Guid ToolMngmntSvc.Subscribe(ToolMngmntSvc.NewToolInfo cb);	
Parameter	**Meaning**
cb	Delegate implementation for the signaled tool events.
Return value	GUID, which can be used to identify a running request.
Possible exceptions	ToolMngmntSvcException

Unsubscribing the tool events

Table 11-5: Unsubscribing the tool events

public Guid ToolMngmntSvc.UnSubscribe(ToolMngmntSvc.NewToolInfo cb);	
Parameter	**Meaning**
cb	Delegate implementation for the signaled tool events.
Return value	-
Possible exceptions	ToolMngmntSvcException

Unrestricted © Siemens AG 2024 All Rights Reserved

11.3.4 Specifications for implementing delegates

Notification for the signaled tool events

Table 11-6: NewToolInfo – specification for delegate implementation

public delegate void NewToolInfo(Guid guid, ToolMngmntSvc.ToolInfo[] toolInfo);	
Parameter	**Meaning**
guid	GUID, which when issuing the subscribe request was returned. It is used to identify the request that is being executed.
toolInfo	List of the change information.

Structure of the checkback data (`ToolMngmntSvc.ToolInfo`):

```
public struct ToolInfo
{
   public ToolMngmntSvc.ToolAction action;
   public long duploNo;
   public long tNo;
   public string toolIdent;
}
```

Table 11-7: Checkback data

Value	Meaning
action	Type of performed action
toolIdent	Tool name
duploNo	Duplo number of the tool
tNo	T number of the tool

Actions that can be monitored (`ToolMngmntSvc.ToolAction`):

```
public enum ToolAction
{
   NO_ACTION = 0,
   TOOL_CREATED = 1,
   TOOL_DELETED = 2,
   TOOL_DATA_CHANGED = 3,
   TOOL_MOVED = 4,
}
```

Table 11-8: Actions

Value	Meaning
NO_ACTION	No tool-related activity
TOOL_CREATED	New tool created
TOOL_DELETED	Tool deleted
TOOL_DATA_CHANGED	Tool data has been changed
TOOL_MOVED	Tool loaded, unloaded or relocated

© Siemens AG 2024 All Rights Reserved

12

12 Action log service

Objective of the chapter

This chapter describes the interface of the action log service. The action log service allows the action log of SINUMERIK Operate to be accessed.

Further, for the user it is possible to extend this log by several entries. As a consequence, there is a link with respect to time between the actual entries and the events monitored by the action log.
For instance, user IDs can be included in the action log and therefore operator actions can be precisely assigned to a user.

Unrestricted © Siemens AG 2024 All Rights Reserved

12.1 Introduction

12.1.1 Class model

Overview

Using the action log it is possible to log various operator sequences in order to subsequently be able to track them. In so doing, various events can be monitored:

- Alarm state changes
- Keystrokes
- Channel state changes
- Window switchover
- Writing NCK/PLC data
- File access
- Function calls
- Current program status

The class model of the action log service primarily consists of the following class:

- TripRecorder

Namespace

All classes are defined in the "Siemens.Sinumerik.Operate.Services" namespace. As a consequence, this namespace can be integrated into the C# project at the start of the particular file:

```
using Siemens.Sinumerik.Operate.Services;
```

Class TripRecorder

Using TripRecorder objects, you can read-out the current action log. It is also possible to add your own entries to the log.

To read-out the action log, there is a synchronous as well as an asynchronous call.

12.1.2 Explanation of terms

Synchronous calls

Synchronous calls return only after the request is performed, i.e. the calling thread is blocked in the meantime. This can interfere with event processing since control inputs and displays are withheld during a synchronous call. For this reason, calls that may take a long time should be performed asynchronously.

Asynchronous calls

Asynchronous calls return as soon as the request has been transferred to the archive service. The returned error code cannot therefore indicate whether the request has been completed successfully; it can only indicate whether or not the request has been sent successfully. For example, an error occurs if call parameters are not correctly assigned. The actual request status is supplied in the callbacks of the action log service (call of the implemented delegate).

© Siemens AG 2024 All Rights Reserved

Log types

Table 12-1: Log types

Log type	Description
Current log	This log contains the current entries. It is implemented as ring buffer; this means that old entries are automatically overwritten.
Crash log	For critical events, the action log saves the contents of the current log in the so-called crash log. The entries backed up can therefore no longer be overwritten by new entries. The crash log can only be overwritten if an additional critical event occurs.

.

Unrestricted © Siemens AG 2024 All Rights Reserved

12.2 Step-by-step examples

Overview

The following chapters show the various applications of the action log service step-by-step. An executable example is included in the SINUMERIK Integrate Create MyHMI /3GL for each "step-by-step example".

Table 12-2: Overview of the examples

Application	Example	Chapter
Synchronously reading out the action log	TripRecorderSync	12.2.2
Asynchronously reading out the action log	TripRecorderAsync	12.2.3

Only the points relevant for the action log service are explained or described in each example. The comments for the sources contain more detailed information (e.g. screen layout, outputs, etc.).

12.2.1 Preparation

Overview

Preparatory measures are described in Chapter 2.1 ➔ "New Project". All of the items described there must be executed first.

12.2.2 Synchronously reading out the action log

The following example shows, step-by-step, which actions are required to start and stop the action log, synchronously read out the log and add your own entry. The matching executable example in the SINUMERIK Integrate Create MyHMI /3GL is the example "TripRecorderSync".

The action log can be started and stopped in the example. You can add your own entries to the action log using the "Make TripEntry" button. The current log is displayed with "GetTripLog". The status bar displays the status of the action log.

© Siemens AG 2024 All Rights Reserved

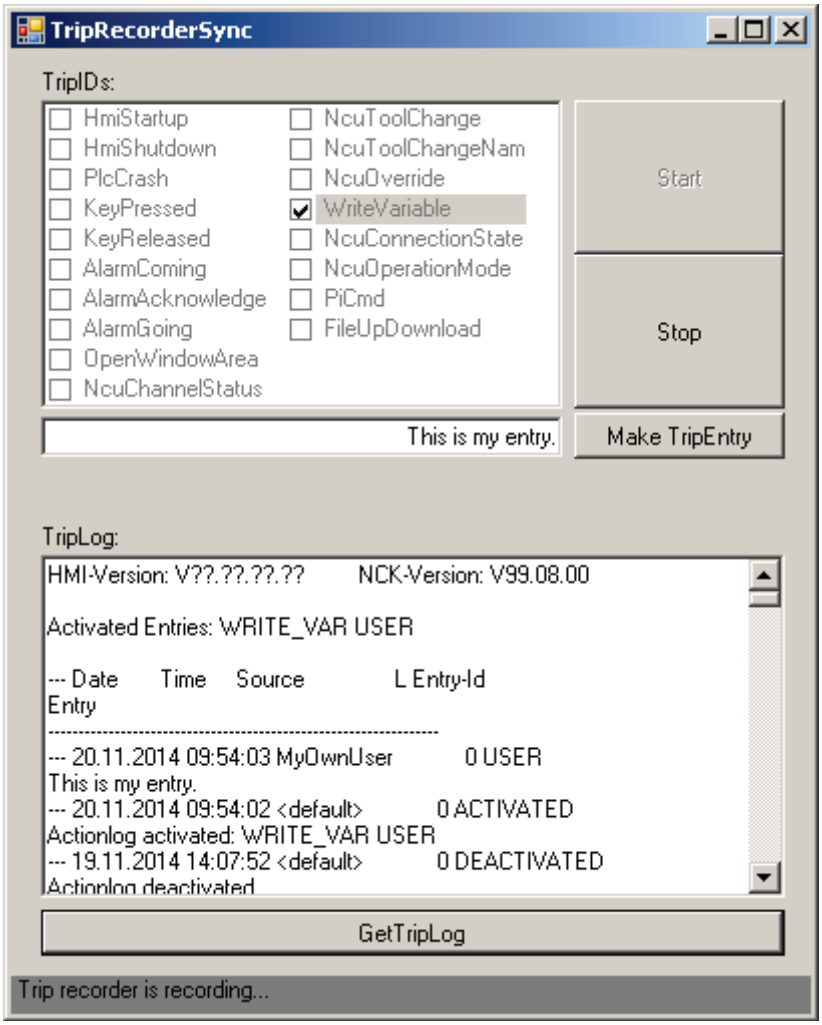


Fig. 12-1: TripRecorderSync example

Step 1

Create the required member variables.

```
TripRecorder m_TripRecorder = null;
```

Step 2

Create the TripRecorder object when loading the form. The user name under which own entries are entered subsequently is specified as parameter.

```
m_TripRecorder = new TripRecorder("MyOwnUser");
```

Unrestricted © Siemens AG 2024 All Rights Reserved

Step 3

Install the notification (subscription) for the status changes of the action log. In this case, only the delegate is required as the transfer parameter, in which the status changes of the archive service should be received (onStatusChanged).

```
m_TripRecorder.Subscribe(onStatusChanged);
```

Step 4

Implement the delegates.

Important

Because the delegate is called from another thread, not all display elements can be accessed. Consequently, it is desirable to call a second delegate with the same signature with which changes can be performed on the user interface.

In the implemented delegate function, a change is sent to the same delegate signature, now in the GUI thread. The status change is now output here.

```
private void onStatusChanged(bool isRecording)
{
   // are we in GUI thread (main) ?
   // No we are not in GUI thread => send message to GUI thread
   if (lblStatus.InvokeRequired)
   {
       TripRecorder.StatusChanged internalDelegate = new
                              TripRecorder.StatusChanged(onStatusChanged);
       BeginInvoke(internalDelegate, new Object[] { isRecording });
   }
   else
   {
       // enable/disable gui elements
       if (true == isRecording)
       {
           setStatus("Trip recorder is recording...");
           cmdStart.Enabled = false;
           ...
       }
       else
       {
           setStatus("");
           cmdStart.Enabled = true;
           ...
       }
   }
}
```

Step 5

Start the action log. The events to be logged are transferred in the form of an array. If an empty array is transferred, only your own entries are logged.

```
TripEvents[] tripEvents = new TripEvents[2];
tripEvents[0] = TripEvents.AlarmComing;
tripEvents[1] = TripEvents.WriteVariable;

m_TripRecorder.Start(tripEvents);
```


© Siemens AG 2024 All Rights Reserved

Step 6

Add your own entry to the action log. The user name used when creating the TripRecorder object is taken as user name.

```
m_TripRecorder.Write("This is my entry.");
```

Step 7

Stop the action log.

```
m_TripRecorder.Stop();
```

Step 8

Write the current action log to a file. In this example, the contents of the log file are directly loaded into a rich text box.

```
m_TripRecorder.GetTripLog("C:\\tmp\\log.txt");

// load written trip log file to textbox
rtbTripLog.LoadFile("C:\\tmp\\log.txt", RichTextBoxStreamType.PlainText);
```

Note

The synchronous function "getTripLog" can take a very long time to return depending on the size of the current action log file and therefore block the complete SINUMERIK Operate. An alternative to this is the asynchronous version "GetTripLogAsync".

Step 9

If the notification for the status change of the action log is no longer required, `UnSubscribe` is called.

```
m_TripRecorder.UnSubscribe(onStatusChanged);
```

Unrestricted © Siemens AG 2024 All Rights Reserved

12.2.3 Asynchronously reading out the action log

The following example shows, step-by-step, which actions are required to start and stop the action log, asynchronously read out the log and add your own entry. The matching executable example in the SINUMERIK Integrate Create MyHMI /3GL is the example "TripRecorderAsync".

The action log can be started and stopped in the example. You can add your own entries to the action log using the "Make TripEntry" button. The current log is displayed with "GetTripLogAsync". The status bar displays the status of the action log.

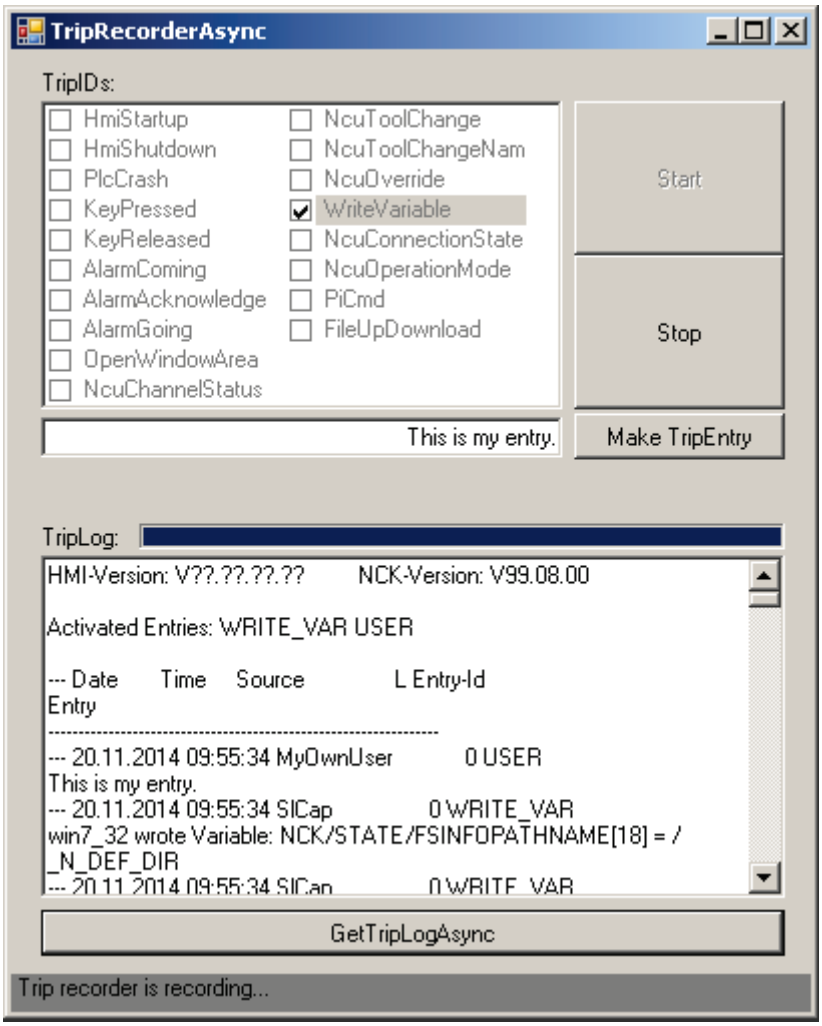


Fig. 12-2: TripRecorderAsync

Note

Apart from Step 8, the procedure is identical to that of example "TripRecorderSync" and is therefore not explained again.

© Siemens AG 2024 All Rights Reserved

Steps 1 to 7

See example "Synchronously reading out the action log" (TripRecorderSync)

Step 8a

Create the required member variables for the identification of the asynchronous call.

```
Guid m_guidGetTripLogAsync = Guid.Empty;
```

Step 8b

Request the current action log. In this case, only the delegates are required as transfer parameters, in which the progress (`onTripLogProgress`) and completion (`onTripLogCompleted`) of the request are to be signaled.

```
m_guidGetTripLogAsync = m_TripRecorder.GetTripLogAsync("C:\\tmp\\log.txt",
                                                       onTripLogCompleted,
                                                       onTripLogProgress);
```

Step 8c

Implement the delegates.

Important

Because the delegate is called from another thread, not all display elements can be accessed. Consequently, it is desirable to call a second delegate with the same signature with which changes can be performed on the user interface.

In the implemented delegate function, a change is sent to the same delegate signature, now in the GUI thread. It is possible to react to the progress and completion here. In the latter, the contents of the log file are directly loaded into a rich text box.

```
private void onTripLogProgress(Guid guid, int value)
{
   if (lblStatus.InvokeRequired)
   {
       // are we in GUI thread (main) ?
       // No we are not in GUI thread => send message to GUI thread
       TripRecorder.TripLogProgress internalDelegate = new
                        TripRecorder.TripLogProgress(onTripLogProgress);
       BeginInvoke(internalDelegate, new Object[] { guid, value });
   }
   else
   {
       // check that guid is correct
       if (guid == m_guidGetTripLogAsync)
           prgProgress.Value = value;
   }
}
```

Unrestricted © Siemens AG 2024 All Rights Reserved

```
private void onTripLogCompleted(Guid guid)
{
   if (lblStatus.InvokeRequired)
   {
       // are we in GUI thread (main) ?
       // No we are not in GUI thread => send message to GUI thread
       TripRecorder.TripLogCompleted internalDelegate = new
                          TripRecorder.TripLogCompleted(onTripLogCompleted);
       BeginInvoke(internalDelegate, new Object[] { guid });
   }
   else
   {
       // check that guid is correct
       if (guid == m_guidGetTripLogAsync)
       {
           // load written trip log file to textbox
           rtbTripLog.LoadFile("C:\\tmp\\log.txt",
                                        RichTextBoxStreamType.PlainText);
       }
   }
}
```

Step 9

See example "Synchronously reading out the action log" (TripRecorderSync)

© Siemens AG 2024 All Rights Reserved

12.3 Reference TripRecorder

12.3.1 Definitions

Overview

Using TripRecorder objects, you can read-out the current action log. It is also possible to add your own entries to the log.

To read-out the action log, there is a synchronous as well as an asynchronous call.

The class is defined in the "Siemens.Sinumerik.Operate.Services" namespace.

The class has two classes for exception handling:
- TripRecorderBusyException
- TripRecoderException

TripRecorderBusyException is triggered if the service is already busy processing another call when the function is called.
TripRecoderException returns general problems associated with the use of the action log service.

12.3.2 Creating a TripRecorder object

A number of constructors are available for creating a TripRecorder object.

Table 12-3: Creating a TripRecorder object

public TripRecorder(); **public TripRecorder(string username);** **public TripRecorder(string username, TripEvents[] events);**	
Parameter	**Meaning**
username	User name for own entries
events	Events to be recorded

A user name already set when creating the TripRecorder object can be queried with the following property.

Table 12-4: Set user name

public string Username;

12.3.3 Releasing assigned resources

This function releases the resources used by a TripRecorder object again. If this is not done, the system will release the resources at some point once the object has become invalid.

Table 12-5: Release of resources being used

public void Dispose()

Unrestricted © Siemens AG 2024 All Rights Reserved

12.3.4 Starting and stopping the recording

Starting the recording

Starts a recording.

Table 12-6: Starting the recording

public void Start(); **public void Start(TripEvents[] events);**	
Parameter	**Meaning**
events	Events to be recorded

If the events to be recorded were already specified when creating the object, the function is used without parameters. The already set events to be recorded can be queried with the following property.

Table 12-7: Set events

public TripEvents;

Stopping the recording

Stops an already started recording.

Table 12-8: Stopping the recording

public void Stop();	
Parameter	**Meaning**

12.3.5 Generation of log entries

Writing an entry to the action log.

Table 12-9: Generation of log entries

public void Write(string info); **public void Write(string info, string username);**	
Parameter	**Meaning**
info	Log entry
username	User name for own entries

12.3.6 Reading out the log

Request current log synchronously

Generates an ASCII file with the current entries of the action log. The call is synchronous and blocks SINUMERIK Operate
until the file has been completely generated.

Table 12-10: Request current log synchronously

public void GetTripLog(string logFilename);	
Parameter	**Meaning**
logFilename	File path and file name of the file to be generated

© Siemens AG 2024 All Rights Reserved

Request current log asynchronously

Generates an ASCII file with the current entries of the action log. The call is an asynchronous call and returns immediately. The transferred delegate implementation contains information about the progress and completion.

Table 12-11: Request current log asynchronously

public Guid GetTripLogAsync(string logFilename, TripRecorder.TripLogCompleted onComplete);	
public Guid GetTripLogAsync(string logFilename, TripRecorder.TripLogCompleted onCompleted, TripRecorder.TripLogProgress onProgress);	
Parameter	**Meaning**
logFilename	File path and file name of the file to be generated
onProgress	Delegate implementation for the progress message.
onCompleted	Delegate implementation for the message, if the request has been completed.
Return value	GUID, which can be used to identify a running request.

Request crash log synchronously

Generates an ASCII file with the current entries of the crash log. The call is synchronous and blocks SINUMERIK Operate until the file has been completely generated.

Table 12-12: Request crash log asynchronously

public void GetCrashLog(string crashlogFilename);	
Parameter	**Meaning**
crashlogFilename	File path and file name of the file to be generated

Request crash log asynchronously

Generates an ASCII file with the current entries of the crash log. The call is an asynchronous call and returns immediately. The transferred delegate implementation contains information about the progress and completion.

Table 12-13: Request crash log synchronously

public Guid GetCrashLogAsync(string crashlogFilename, TripRecorder.CrashLogCompleted onCompleted);	
public Guid GetCrashLogAsync(string crashlogFilename, TripRecorder.CrashLogCompleted onCompleted, TripRecorder.CrashLogProgress onProgress);	
Parameter	**Meaning**
crashlogFilename	File path and file name of the file to be generated
onProgress	Delegate implementation for the progress message.
onCompleted	Delegate implementation for the message, if the request has been completed.
Return value	GUID, which can be used to identify a running request.

Unrestricted © Siemens AG 2024 All Rights Reserved

12.3.7 Status of the recording

Subscribe to the status of the recording

Table 12-14: Subscribe to the status of the recording

public void Subscribe(TripRecorder.StatusChanged onStatusChanged);	
Parameter	**Meaning**
onStatusChanged	Delegate implementation for the status change

Unsubscribing

Table 12-15: Unsubscribing

public void UnSubscribe(TripRecorder.StatusChanged onStatusChanged);	
Parameter	**Meaning**
onStatusChanged	Delegate implementation for the status change

Querying the status of the recording

Table 12-16: Querying the status of the recording

public bool IsRecording;

12.3.8 Canceling a request that is being executed

Canceling an already started asynchronous request.

Table 12-17: Canceling a request that is being executed

public void Cancel(Guid asyncGuid);	
Parameter	**Meaning**
asyncGuid	GUID, which was returned when issuing a request. It is used to identify the request that is being executed.

12.3.9 Specifications for implementing delegates

Notification for status of the recording

Table 12-18: Notification for status of the recording

public delegate void StatusChanged(bool isRecording);	
Parameter	**Meaning**
isRecording	True if currently a recording is in progress.

Notification about the progress of an asynchronous request

Table 12-19: Notification about the progress of an asynchronous request

public delegate void TripLogProgress(Guid guid, int value); **public delegate void CrashLogProgress(Guid guid, int value);**	
Parameter	**Meaning**
guid	GUID, which was returned when issuing a request. It is used to identify the request that is being executed.
value	Progress as a percentage

© Siemens AG 2024 All Rights Reserved

Notification about the end of an asynchronous request

Table 12-20: Notification about the end of an asynchronous request

public delegate void TripLogCompleted(Guid guid); **public delegate void CrashLogCompleted(Guid guid);**	
Parameter	**Meaning**
guid	GUID, which was returned when issuing a request. It is used to identify the request that is being executed.

12.3.10 Events

The following events can be logged. They are transferred in the form of an array (`TripEvents[]`) either in the constructor or when starting a recording.

Table 12-21: TripEvents

TripEvents	Description
HmiStartup	Logs the HMI startup
HmiShutdown	Logs the HMI shutdown
PlcCrash	Logs the PLC crash (setting of DB19.DBX0.6 generates an entry with the current program status and then writes the crash log)
KeyPressed	Logs when a key is pressed
KeyReleased	Logs when a key is released
AlarmComing	Logs an alarm coming event
AlarmAcknowledge	Logs an alarm acknowledge event
AlarmGoing	Logs an alarm going event
OpenWindowArea	Logs when a window is opened
NcuChannelStatus	Logs the channel status
NcuToolChange	Logs a tool change
NcuToolChangeName	Logs a tool change
NcuOverride	Logs override changes
WriteVariable	Logs write accesses of NCK/PLC variables
NcuConnectionState	Logs the status of the NC connection
NcuOperationMode	Logs an operating mode change
PiCmd	Logs PI services
FileUpDownload	Logs file transfers in the NC

Unrestricted © Siemens AG 2024 All Rights Reserved

© Siemens AG 2024 All Rights Reserved

13 COM clients and .NetApi

Objective of the chapter

This chapter shows, as example, the creation of what is known as a ComCallableWrapper (CCW).

Unrestricted © Siemens AG 2024 All Rights Reserved

13.1 Introduction

13.1.1 Overview

.NET components can be used by COM clients (COM → Component Object Model). This is specifically realized for programming languages such as Visual Basic 6, Delphi etc., for instance using what is known as a **C**om**C**allable**W**rapper (CCW). This CCW is a DLL that must have some specific features.

Using this technology, under SINUMERIK Operate, for example Visual Basic 6 can access the .NET interface. This is realized by generating your own InterfaceDLL (CCW).

This possibility will now be shown as example in the following. Additional information on using .NET components based on COM technology can be taken from the links listed below.

13.1.2 Links

Com interoperability:
http://msdn.microsoft.com/de-de/library/bd9cdfyx.aspx

ComWrapper launch:
http://msdn.microsoft.com/de-de/library/5dxz80y2.aspx

ComCallableWrapper
http://msdn.microsoft.com/de-de/library/f07c8z1c.aspx

Raising Events Handled by a COM Sink
http://msdn.microsoft.com/en-us/library/dd8bf0x3(v=vs.71).aspx

13.1.3 Disclaimer of liability

The solutions shown are examples, and must be seen as such, especially regarding the topic of error handling. This was not discussed here for reasons of transparency.

Any liability on the part of Siemens AG is completely excluded, unless there is a legally binding liability, e.g. in cases of intent, gross negligence, injury of life, body or health of a person.

© Siemens AG 2024 All Rights Reserved

13.2 Creating an interface DLL

The following steps are required to create an interface DLL:

1) Create a new project, and under C#, select the "ClassLibrary" project type.
2) Rename the default class, e.g. "ccwDataSvc", and implement the necessary methods.
3) Insert additional "using" directives:

```
using System.Runtime.InteropServices;
using Siemens.Sinumerik.Operate.Services;
```

4) Declare one delegate that has the signature of the callback events.

```
public delegate void ValueChangedDelegate(String value);
```

5) Define a public interface that contains the methods that can be accessed from COM.

```
// Methods called from COM Client
// Methods reachable via COM
public interface IDataSvc
{
    String Read(String varName);
    void Write(String varName, String value);
    void Subscribe(String varName);
    void UnSubscribe();
}
```

6) Declare methods for hotlinks as interface.

```
// Subscriptions (Hotlinks)
// Step 1: Defines an event sink interface (IDataSvcEvents) to be
// implemented by the COM sink (COM Client).
[GuidAttribute("646B0F67-372A-4834-9744-E49A7B62CAE1")]
// <-- Make sure to specify your own unique GUID.
[InterfaceTypeAttribute(ComInterfaceType.InterfaceIsIDispatch)]
public interface IDataSvcEvents
{
    void ValueChanged(String value);
}
```

7) The class must
   - be allocated the attributes "ClassInterface" and "ComSourceInterfaces"
   - be inherited from "public Interface"
   - define an event that corresponds to the delegate declared previously
   - have a Default Constructor.

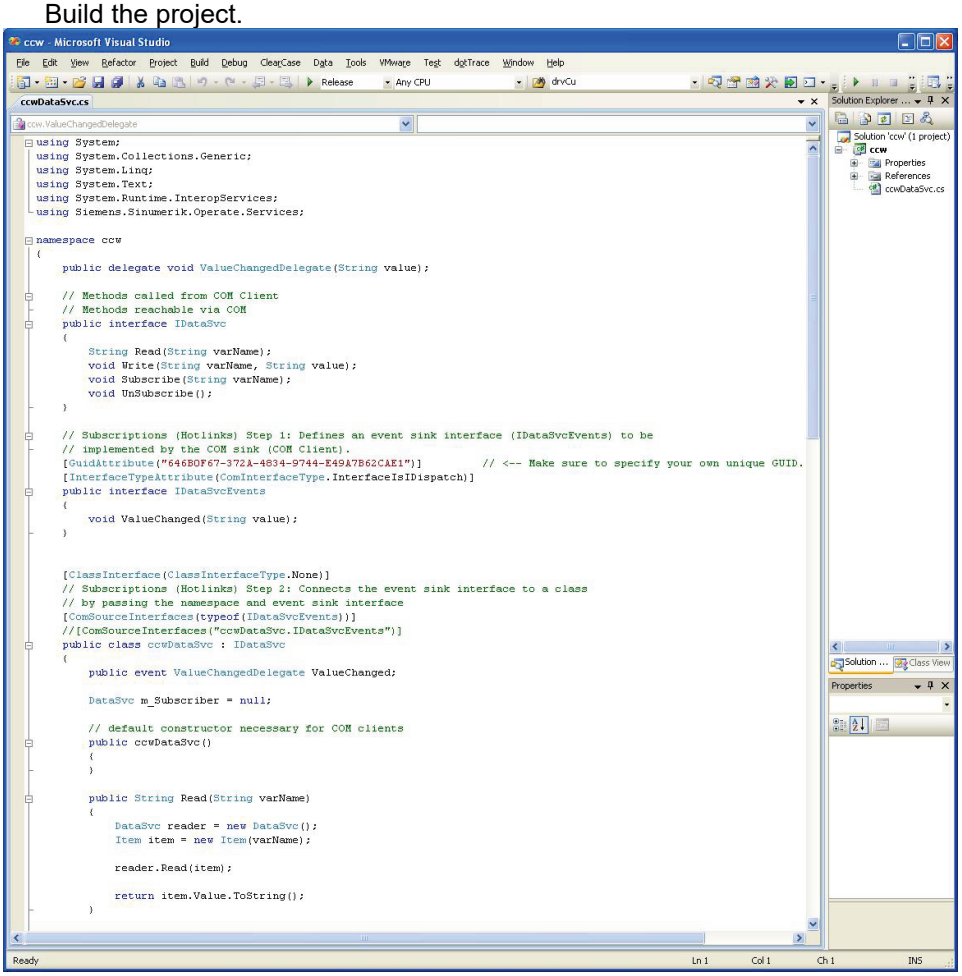
```
[ClassInterface(ClassInterfaceType.None)]
// Subscriptions (Hotlinks)
// Step 2: Connects the event sink interface to a class
// by passing the namespace and event sink interface
[ComSourceInterfaces(typeof(IDataSvcEvents))]
//[ComSourceInterfaces("ccwDataSvc.IDataSvcEvents")]
public class ccwDataSvc : IDataSvc
{
    public event ValueChangedDelegate ValueChanged;

    // default constructor necessary for COM clients
    public ccwDataSvc()
    {
    }
}
```

Unrestricted © Siemens AG 2024 All Rights Reserved

8) Build the project and the project settings (-properties)
   - Application
     ➔ Button "Assembly Information"
     ➔ "Checkmark" for "Make Assembly COM-Visible"
   - Build
     ➔"Checkmark" for "Register for COM interop"

Build the project.




```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.InteropServices;
using Siemens.Sinumerik.Operate.Services;

namespace ccw
{
    public delegate void ValueChangedDelegate(String value);

    // Methods called from COM Client
    // Methods reachable via COM
    public interface IDataSvc
    {
        String Read(String varName);
        void Write(String varName, String value);
        void Subscribe(String varName);
        void UnSubscribe();
    }

    // Subscriptions (Hotlinks) Step 1: Defines an event sink interface (IDataSvcEvents) to be
    // implemented by the COM sink (COM Client).
    [GuidAttribute("646B0F67-372A-4834-9744-E49A7B62CAE1")]         // <-- Make sure to specify your own unique GUID.
    [InterfaceTypeAttribute(ComInterfaceType.InterfaceIsIDispatch)]
    public interface IDataSvcEvents
    {
        void ValueChanged(String value);
    }


    [ClassInterface(ClassInterfaceType.None)]
    // Subscriptions (Hotlinks) Step 2: Connects the event sink interface to a class
    // by passing the namespace and event sink interface
    [ComSourceInterfaces(typeof(IDataSvcEvents))]
    //[ComSourceInterfaces("ccwDataSvc.IDataSvcEvents")]
    public class ccwDataSvc : IDataSvc
    {
        public event ValueChangedDelegate ValueChanged;

        DataSvc m_Subscriber = null;

        // default constructor necessary for COM clients
        public ccwDataSvc()
        {
        }

        public String Read(String varName)
        {
            DataSvc reader = new DataSvc();
            Item item = new Item(varName);

            reader.Read(item);

            return item.Value.ToString();
        }
```

Fig. 11-1: CCW sources

An example of the handling can be found at
_ComUsesNetApi\ComCallableWrapper in the example directory.

© Siemens AG 2024 All Rights Reserved

13.3 Using the interface DLL by the COM client

In Visual Basic, include a reference (Project -> References) to the corresponding DLL (in case of doubt, navigate with browse to the TLB file that has been created).

Using the object browser, the DLL and its methods can be viewed and used in the source code.

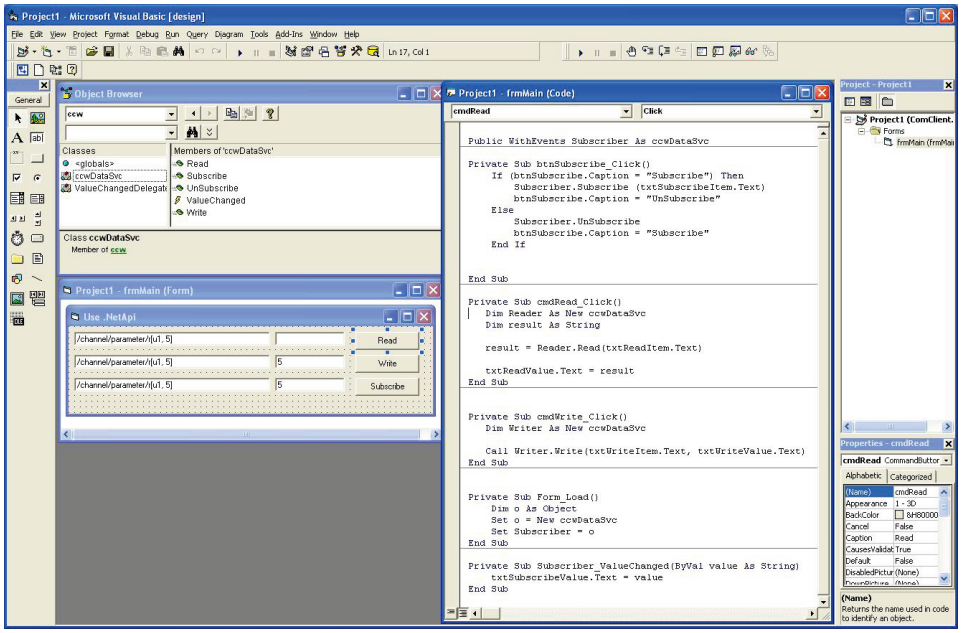


Fig. 11-2: VB6 project

An example of the handling can be found at
_ComUsesNetApi\ComClient in the example directory.

Unrestricted © Siemens AG 2024 All Rights Reserved

© Siemens AG 2024 All Rights Reserved

14

14 Display Manager

Objective of the chapter

You can find a description of the Display Manager in the "SinumerikOperate_QT.pdf" document in Chapter 4.27.

If frames are assigned OEMFrame applications, which use the SINUMERIK Operate .Net Api described in this document, then the following options are available to influence the Display Manager.

Unrestricted © Siemens AG 2024 All Rights Reserved

14.1 Introduction

14.1.1 Class model

Overview

The class model of the Display Managers essentially comprises the DisplayManager class.

Namespace

The DisplayManager object is defined in namespace "Siemens.Sinumerik.Operate.Services". As a consequence, this namespace can be integrated into the C# project at the start of the particular file:

```
using Siemens.Sinumerik.Operate.Services;
```

DisplayManager class

The following actions can be executed using DisplayManager objects:

- Display, change and hide application components
- Display, change and hide menu components
- Activate/deactivate the mirror mode
- Switch over displays
- Notification for the mirror mode, actual display and actual display configuration

© Siemens AG 2024 All Rights Reserved

14.2 Reference DisplayManager

14.2.1 Definitions

Overview

The following actions can be executed using DisplayManager objects:

- Display, change and hide application components
- Display, change and hide menu components
- Activate/deactivate the mirror mode
- Switch over displays
- Notification for the mirror mode, actual display and actual display configuration

The class is defined in the "Siemens.Sinumerik.Operate.Services" namespace.

14.2.2 Creating a SIDisplayManager object

The DisplayManager class only has a default constructor.

Table 14-1: DisplayManager constructor

public DisplayManager(MirrorModeChanged onMirrorModeChanged, DisplayChanged onDisplayChanged, DisplayConfigurationChanged onDisplayConfigurationChanged)	
Parameter	**Meaning**
onMirrorModeChanged	Delegate implementation for the mirror mode
onDisplayChanged	Delegate implementation for display change
onDisplayConfigurationChanged	Delegate implementation for display configuration change

14.2.3 API functions

Table 14-2: showDisplay

Changes to another display.	
void ShowDisplay(string display) **void ShowDisplay(string display, bool noRefresh)**	
Parameter	**Meaning**
display	Target display (from the same display configuration)
noRefresh	true ➔ The display is not directly shown – but only after `refreshDisplay()` is called.

Table 14-3:ActiveDisplay

Supplies the actual display names.
string ActiveDisplay

Table 14-4:MirrorMode

Delivers the status of the mirror mode
bool MirrorMode

Unrestricted © Siemens AG 2024 All Rights Reserved

Table 14-5: showApp

Displays an application component in a certain frame. Application exchange: If the application component is already visible in another frame of the display, it is moved from there to the frame specified here. If possible, the freed frame is filled with the application displaced from *szDestFrame*. You can also attach a display name in front of a frame name (*displayname*.*framename*), therefore changing the frame assignment into invisible displays. *framename*.currentApp or *displayname*.*framename*.currentApp can be used. Here, the application component displayed in the particular frame is shifted into a new frame.	
void ShowApp(string destinationFrame, string applicationName); **void ShowApp(string destinationFrame, string applicationName, bool noRefresh);**	
Parameter	**Meaning**
destinationFrame	Target frame
applicationName	Application component
noRefresh	true ➔ The display is not directly shown – but only after `refreshDisplay()` is called.

Table 14-6: showPopup

Essentially corresponds to the `showApp()` function. However, the target frame cannot be located in a display. It is automatically displayed when `showPopup()` is called - and placed over the currently active display. Only your own HMI dialogs can be used as *appname*. The window to be displayed is declared as top-level window (Qt::WindowStaysOnTopHint). Only one popup can be open at a time.	
void ShowPopUp(string destinationFrame, string applicationName); **void ShowPopUp(string destinationFrame, string applicationName, bool noRefresh)**	
Parameter	**Meaning**
destinationFrame	Target frame
applicationName	Application component
noRefresh	true ➔ The display is not directly shown – but only after `refreshDisplay()` is called.

Table 14-7: hidePopup

Hides a component displayed with `showPopup()` or `togglePopup()`.	
void HidePopUp(string applicationName) **void HidePopUp(string applicationName, bool noRefresh)**	
Parameter	**Meaning**
applicationName	Application component
noRefresh	true ➔ The display is not directly shown – but only after `refreshDisplay()` is called.

© Siemens AG 2024 All Rights Reserved

Table 14-8: togglePopup

If the popup frame is already displayed as a popup, this frame is hidden. Otherwise, it is displayed. This function is therefore a combination of `showPopup()` and `hidePopup()`.	
void TogglePopup(string PopupFrameName, string applicationName) **void TogglePopup(string popupFrameName, string applicationName, bool noRefresh)**	
Parameter	**Meaning**
PopupFrameName	Popup frame
applicationName	Application component
noRefresh	true ➔ The display is not directly shown – but only after `refreshDisplay()` is called.

Table 14-9: showMenu

Displays a menu in the specified frame. Otherwise, this command corresponds to the `showApp()` function.	
void ShowMenu(string destinationFrame, string menu) **void ShowMenu(string destinationFrame, string menu, bool noRefresh)**	
Parameter	**Meaning**
destinationFrame	Target frame
menu	menu to be displayed
noRefresh	true ➔ The display is not directly shown – but only after `refreshDisplay()` is called.

Table 14-10: showPopupMenu

Displays a menu as popup in the specified frame. Otherwise, this command corresponds to the `showPopup()` function.	
void ShowPopupMenu(string destinationFrame, string menu) **void ShowPopupMenu(string destinationFrame, string menu, bool noRefresh)**	
Parameter	**Meaning**
destinationFrame	Target frame
menu	menu to be displayed
noRefresh	true ➔ The display is not directly shown – but only after `refreshDisplay()` is called.

Table 14-11: hidePopupMenu

Hides a menu displayed with `showPopupMenu()` or `togglePopup()`.	
void HidePopupMenu(string menu) **void HidePopupMenu(string menu, bool noRefresh)**	
Parameter	**Meaning**
menu	menu to be hidden
noRefresh	true ➔ The display is not directly shown – but only after `refreshDisplay()` is called.

Table 14-12: togglePopupMenu

Hides a menu already displayed as popup. Otherwise, it is displayed. This function is therefore a combination of `showPopupMenu()` and `hidePopupMenu()`.	
void TogglePopupMenu(string popupFrameName, string menu) **void TogglePopupMenu(string popupFrameName, string menu, bool noRefresh)**	
Parameter	**Meaning**
popupFrameName	Popup frame
menu	Menu
noRefresh	true ➔ The display is not directly shown – but only after `refreshDisplay()` is called.

Unrestricted © Siemens AG 2024 All Rights Reserved

Table 14-13: setMirrorMode

Activates or deactivates the mirror mode. When activated, it means that all display frames are displayed horizontally mirrored, if `enableMirrorMode: is set = true` for the display.	
void SetMirrorMode(bool mirror) **void SetMirrorMode(bool mirror, bool noRefresh)**	
Parameter	**Meaning**
mirror	true ➔ Activates the mirror mode false ➔ Deactivates the mirror mode
noRefresh	true ➔ The display is not directly shown – but only after `refreshDisplay()` is called.

14.2.4 Notifications

Table 14-14: displayConfigurationChanged

This delegate is initiated if, after a TCU switchover, the active display configuration changes.	
delegate void DisplayConfigurationChanged(string displayConfig)	
Parameter	**Meaning**
displayConfig	Name of the display configuration (empty, if a display configuration has not been defined for the actual resolution)

Table 14-15: displayChanged

This delegate is initiated if the actual display and/or the application components visible there change.	
delegate void DisplayChanged(string display, string[] visibleApps, string[] visibleFrames)	
Parameter	**Meaning**
display	New display names
visibleApps	Name of the visible components
visibleFrames	Name of the visible frame

Table 14-16: mirrorModeChanged

This delegate is initiated if the mirror mode is switched over.	
delegate void MirrorModeChanged(bool mirrorMode)	
Parameter	**Meaning**
mirrorMode	New mirroring mode

© Siemens AG 2024 All Rights Reserved

I Index

I.1 Keyword index

.

.NET-Framework 2-18

2

20-second timeout 2-42

A

Access levels 6-132, 7-178
Action log service 12-265
AlarmService 5-99
Architecture 1-13
Archive service 8-185
Archive types 8-187
Array access DataSvc 3-49

C

COM clients 13-281

D

DataService 3-47
Debugging 2-39
Drive machine data service 9-213

E

Error evaluation 2-21

F

File paths 6-147
FileService 6-129

I

infrastructure service 7-171
Integrating in HMI 2-22

L

Language selection 7-178

N

Namespace 2-20
New project 2-17

O

oemframe.ini 2-28
Operating area menu 2-27

P

PI service status 4-91
PiService 4-91
PLC access 4-98
PLC accesses 3-76

S

systemconfiguration.ini 2-22

T

Target platform 2-19
ToolManagement Service 11-259
TraceDataRecorder Service 10-235
Trip IDs 12-279
two-dimensional GUDs 3-89

V

Versioning 2-36

Unrestricted © Siemens AG 2024 All Rights Reserved

© Siemens AG 2024 All Rights Reserved