

SIMATIC WinCC

Unified SCADA Runtime - Open Development Kit (ODK)




System Manual

Basics	1
Creating a minimal ODK client	2
Authorizing users	3
Startup and shutdown behavior of an ODK application	4
Syntax of the alarm filter	5
Locale IDs of the supported languages	6
Code samples	7
Description of the C# interfaces	8
Description of the C++ interfaces	9

Legal information

Warning notice system

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

 DANGER
indicates that death or severe personal injury will result if proper precautions are not taken.
 WARNING
indicates that death or severe personal injury may result if proper precautions are not taken.
 CAUTION
indicates that minor personal injury can result if proper precautions are not taken.
NOTICE
indicates that property damage can result if proper precautions are not taken.


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

Qualified Personnel

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

Proper use of Siemens products

Note the following:

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

Trademarks

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

Disclaimer of Liability

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

Table of contents

1	Basics	9
2	Creating a minimal ODK client	11
3	Authorizing users	15
4	Startup and shutdown behavior of an ODK application	17
4.1	Autostart of an ODK application	17
4.2	Shutdown behavior	17
4.3	Restart behavior	18
5	Syntax of the alarm filter	19
6	Locale IDs of the supported languages	21
7	Code samples	25
8	Description of the C# interfaces	27
8.1	Releasing objects	27
8.2	Interfaces of the Runtime environment	28
8.2.1	IRuntime	28
8.2.2	IProduct	32
8.2.3	IOption	33
8.2.4	IVersionInfo	35
8.3	Error-handling interfaces	36
8.3.1	IErrorResult	36
8.3.2	IErrorInfo	38
8.3.3	OdkException	38
8.4	Interfaces of the tags	40
8.4.1	IProcessValue	40
8.4.2	ITag	42
8.4.3	ITagSet	44
8.4.4	ITagSetQCD	52
8.4.5	ITagSetQCItem	55
8.4.6	ILoggedTagValue	56
8.4.7	ILoggedTag	57
8.4.8	ILoggedTagSet	58
8.4.9	ITags	62
8.4.10	ITagAttributes	64
8.4.11	ILoggingTags	66
8.4.12	ILoggingTagAttributes	67
8.5	Interfaces of the alarms	68
8.5.1	IAlarmResult	68
8.5.2	IAlarm	76
8.5.3	IAlarmSet	78
8.5.4	IAlarmSetResult	86

8.5.5	IAlarmTrigger	87
8.5.6	ITextList	94
8.5.7	IAlarmSubscription	94
8.5.8	ILoggedAlarmResult	99
8.5.9	IAlarmLogging	104
8.5.10	IAlarmLoggingSubscription	106
8.6	Interfaces for connections	109
8.6.1	IConnectionResult	109
8.6.2	IConnectionStatusResult	111
8.6.3	IConnection	113
8.6.4	IConnectionSet	114
8.7	Interfaces of the Plant Model	120
8.7.1	IPlantModel	120
8.7.2	IPlantObject	122
8.7.3	IPlantObjectProperty	125
8.7.4	IPlantObjectPropertyValue	127
8.7.5	IPlantObjectPropertySet	129
8.7.6	IPlantObjectAlarmSubscription	136
8.8	Interfaces of the Calendar option	139
8.8.1	ISHCCalendar	139
8.8.2	ISHCCategory	140
8.8.3	ISHCCategoryProvider	141
8.8.4	ISHCCalendarSettings	142
8.8.5	ISHCTimeSlice	143
8.8.6	ISHCDay	144
8.8.7	ISHCDayProvider	145
8.8.8	ISHCDayTemplate	149
8.8.9	ISHCDayTemplatesProvider	151
8.8.10	ISHCShiftTemplate	155
8.8.11	ISHCShiftTemplatesProvider	157
8.8.12	ISHCShift	161
8.8.13	ISHCAction	164
8.8.14	ISHCActionElement	165
8.8.15	ISHCActionTemplate	166
8.8.16	ISHCActionTemplateElement	168
8.8.17	ISHCActionTemplatesProvider	168
8.9	Interfaces of the contexts	172
8.9.1	IContextLogging	172
8.9.2	IContextDefinition	182
8.9.3	ILoggedContext	184
8.9.4	IContextError	185
8.9.5	IContextFilter	185
9	Description of the C++ interfaces	189
9.1	Error codes of the C++ interfaces	189
9.2	Interfaces of the Runtime environment	189
9.2.1	IOdkRt	189
9.2.2	IRuntime	191
9.2.3	IProduct	195
9.2.4	IOption	196

9.2.5	IOptionEnumerator	198
9.2.6	IVersionInfo	200
9.2.7	IErrorResult	201
9.2.8	IErrorResultEnumerator	204
9.2.9	IErrorInfo	206
9.3	Interfaces of the tags.....	208
9.3.1	IProcessValue	208
9.3.2	IProcessValueEnumerator	210
9.3.3	ITag	212
9.3.4	ITagCallback	216
9.3.5	ITagSet.....	221
9.3.6	ITagSetQCD.....	229
9.3.7	ITagSetQCDItem.....	234
9.3.8	ILoggedTagValue	236
9.3.9	ILoggedTagValueEnumerator	238
9.3.10	ILoggedTagCallback / ILoggedTagSetCallback	239
9.3.11	ILoggedTag	244
9.3.12	ILoggedTagSet.....	247
9.3.13	ITags	251
9.3.14	ITagAttributes	252
9.3.15	ITagAttributesEnumerator	255
9.3.16	ITagAttributesCallback	256
9.3.17	ILoggingTags	256
9.3.18	ILoggingTagAttributes	257
9.3.19	ILoggingTagAttributesEnumerator	258
9.3.20	ILoggingTagAttributesCallback	259
9.4	Interfaces of the alarms.....	260
9.4.1	IAlarmResult.....	260
9.4.2	IAlarmResultEnumerator	269
9.4.3	IAlarm	270
9.4.4	IAlarmCallback	273
9.4.5	IAlarmSourceCommandCallback	277
9.4.6	IAlarmSet	281
9.4.7	IAlarmSetResult	285
9.4.8	IAlarmSetResultEnumerator	286
9.4.9	IAlarmTrigger	287
9.4.10	ITextList.....	294
9.4.11	IAlarmSubscription	295
9.4.12	ILoggedAlarmResult.....	300
9.4.13	ILoggedAlarmResultEnumerator	306
9.4.14	IAlarmLogging	307
9.4.15	IAlarmLoggingCallback	310
9.4.16	IAlarmLoggingSubscription	311
9.5	Interfaces for connections	314
9.5.1	IConnectionResult.....	314
9.5.2	IConnectionResultEnumerator	317
9.5.3	IConnectionStatusResult.....	318
9.5.4	IConnectionStatusResultEnumerator	321
9.5.5	IConnection	322
9.5.6	IConnectionReadNotification.....	323
9.5.7	IConnectionStateChangeNotification	326

9.5.8	IConnectionSet.....	328
9.6	Interfaces of the Plant Model	333
9.6.1	IPlantModel	333
9.6.2	IPlantObject.....	336
9.6.3	IPlantObjectProperty	341
9.6.4	IPlantObjectPropertyValue	343
9.6.5	IPlantModelPropertySubscriptionNotification	344
9.6.6	IPlantObjectPropertyValueEnumerator	347
9.6.7	IPlantObjectPropertySet.....	348
9.6.8	IPlantObjectPropertySetReadReply	352
9.6.9	IPlantObjectPropertySetWriteReply	354
9.6.10	IPlantObjectEnumerator	356
9.6.11	IPlantObjectAlarmSubscription	358
9.6.12	IPlantObjectAlarmCallback	360
9.6.13	IPlantObjectAlarmSubscriptionCallback.....	362
9.7	Interfaces of the Calendar option	363
9.7.1	ISHCCalendarOption	363
9.7.2	ISHCCalendar	367
9.7.3	ISHCCalendarSettings	369
9.7.4	ISHCCategory	370
9.7.5	ISHCCategoryEnumerator	372
9.7.6	ISHCCategoryProvider.....	374
9.7.7	ISHCTimeSlice	375
9.7.8	ISHCTimeSliceEnumerator	376
9.7.9	ISHCDay	378
9.7.10	ISHCDayEnumerator	380
9.7.11	ISHCDayProvider	383
9.7.12	ISHCDayTemplate	393
9.7.13	ISHCDayTemplatesProvider	395
9.7.14	ISHCShiftTemplate	405
9.7.15	ISHCShiftTemplateEnumerator.....	407
9.7.16	ISHCShiftTemplatesProvider	410
9.7.17	ISHCShift	420
9.7.18	ISHCShiftEnumerator	428
9.7.19	ISHCAction.....	431
9.7.20	ISHCActionEnumerator.....	432
9.7.21	ISHCActionElement	435
9.7.22	ISHCActionElementEnumerator	436
9.7.23	ISHCActionTemplate	439
9.7.24	ISHCActionTemplateEnumerator	441
9.7.25	ISHCActionTemplatesProvider	443
9.7.26	ISHCActionTemplateElement	452
9.7.27	ISHCActionTemplateElementEnumerator	453
9.8	Interfaces of the contexts	456
9.8.1	IContextLogging	456
9.8.2	IContextLoggingCallBack.....	469
9.8.3	IContextDefinitionEnumerator	472
9.8.4	IContextErrorEnumerator	473
9.8.5	ILoggedContextEnumerator	473
9.8.6	IContextDefinition	474
9.8.7	ILoggedContext.....	478

9.8.8	IContextError	480
9.8.9	IContextFilter	480

Basics

Task of the runtime API

Runtime API describes the open programming interface of WinCC Unified Scada RT. With the Runtime API, you can use the internal functions of WinCC in your own applications. With an ODK client, you can read out all the objects of the Runtime system and change their Runtime attributes, for example, for tags or alarms.

The ODK is optimized for the processing of mass data when special objects are used, for example the reading or writing of 1000 tags in one pass.

Note

Siemens is not liable for and does not guarantee the compatibility of the data and information transported via the API interfaces with third-party software.

We expressly point out that improper use of the API interface can result in data loss or production downtimes.

Requirement

- Programming environment is installed, e.g. MS Visual Studio
- WinCC Runtime Unified Scada RT is installed.

Application of C++ and .NET

The Runtime API makes all the interfaces available for the access to the runtime system in the languages C++ and C#.

Name-based addressing of objects

The objects of the Runtime system are addressed by their name and the full name path.

The name path of objects consists of several components and has the following syntax:

```
[SystemName::] [ObjectName] [.ElementPath] [:SubElementName]
```

- `SystemName`
Name of a Runtime systems (optional)
If the "SystemName" is omitted, the object is searched for on the local runtime system.
- `ObjectName`
Name of a tag or a structure
- `ElementPath`
Element of a structure
- `SubElementName`
Subelement of an object, e.g. alarm or logging tag of a tag.

Examples for access to different object types:

- Simple tag: `MyRTSystem::MySimpleTag`
- Structure tag: `MyRTSystem::Motor.Temperature`
- Alarm of a simple tag: `MyDiscreteTag:MyDiscreteAlarm`
- Alarm of a structure tag: `Motor.Temperature:MyAnalogAlarm`
- Logging tag: `MySimpleTag:MyLoggingTag`
- Connection: `MyRTSystem::MyHmiConnection`

Creating a minimal ODK client

Introduction

An ODK client uses the ODK API to access objects of the WinCC Unified system.

In the following, an ODK client is created for use of the Runtime API in the C# and C++ languages.

The programs only contain the most needed components of a simple client. They form the framework for all the subsequent runtime code examples in this documentation. See also section Code samples (Page 25).

Note

You will find additional programming examples on the installation medium in the file "Support\Openness\Siemens.Unified.Openness_SDK_<version number>.zip" in the subdirectory "ODK\samples".

Requirement

- Development environment is installed.
- The ODK SDK was extracted locally on your computer. You will find the ODK SDK in the "Support\Openness" folder on the WinCC Unified DVD in the file "Siemens.Unified.Openness_SDK_<version number>.zip".

Note

If you create a C++ ODK client, you must set the system tag "PATH=C:\Program Files\Siemens\Automation\WinCCUnified\bin" and perform a restart.

Procedure C# client

1. Create a new .NET project in the development environment.
2. Carry out the following project settings:
 - Target framework is .NET 4.6.
 - ODK client is "Release" version for the x64 platform.
3. Create references to the following assembly: Siemens.Runtime.HmiUnified.Interfaces.dll (Copy Local = False)
You will find the assembly in the local folder to which you have extracted Openness_SDK.zip, in the subfolder "ODK\bin".
4. Create a program with the following code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Siemens.Runtime.HmiUnified;

namespace Siemens.Runtime.HmiUnified.TestClient
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                using (IRuntime runtime = Runtime.Connect())
                {
                    //do runtime operations
                }
            }
            catch (Exception ex)
            {
                System.Console.WriteLine(string.Format("Exception occurred
{0}", ex.Message));
            }
        }
    }
}
```

Procedure C++ client

1. Create a new C++ project in the development environment.
2. Set the following include directories for required headers:
 - <Local folder to which you have extracted the ODK SDK>\ODK\include\ODK
 - <Local folder to which you have extracted the ODK SDK>\ODK\include\ODK\include\CF
3. Create references to the following libraries in "<Local folder to which you have extracted the ODK SDK>\ODK\include\ODK\lib":
 - HmiUnifiedRt.lib
 - CfCore.lib
4. Create a reference to the following directory as "Additional Library Directory":
 - <Local folder to which you have extracted the ODK SDK>\ODK\lib
5. Create a program with the following code:

```
#include <CfTL>

#include "IOdkRt.h"
#include "IOdkRtTag.h"
#include "IOdkRtTagLogging.h"
#include "IOdkRtAlarm.h"
#include "IOdkRtAlarmLogging.h"
#include "IOdkRtCpm.h"
#include "IOdkRtConnection.h"
#include "IOdkRtUmc.h"

#include <stdio.h>
#include <tchar.h>
#include <iostream>

using namespace Siemens::Runtime::HmiUnified;
using namespace Siemens::Runtime::HmiUnified::Common;
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    CCfString projectName = L"";
    IRuntimePtr pRuntime;

    if(CF_SUCCEEDED(Connect(projectName, &pRuntime))
    {
        // do runtime operations here
    }
    return 0;
}
```

Result

The program core of an ODK client is created.

You can complete the program with the fragments from the following code examples for ODK-API.

Authorizing users

Introduction

Before the ODK client can be used, all users that run the ODK client must be authorized.

Note

Only authenticated users can access the data of a project over the Runtime API with an ODK client.

Requirement

WinCC Unified RT setup has been carried out completely on the Runtime computer.

Procedure

Add the user who corresponds to the logged-on Windows user on the Runtime computer to the Windows user group "SIMATIC HMI" in the Windows user administration.

Result

When the ODK client is connected to the Runtime system, the logged-on Windows user is authenticated via the user administration of the Runtime computer.

If one of the checks fails, the ODK client does not establish a connection (error: "Authentication error" or "User has no access right").

Startup and shutdown behavior of an ODK application

4.1 Autostart of an ODK application

You have the possibility of starting ODK applications automatically on start-up of the device.

Requirement

Runtime is configured in such a way that it automatically started on start-up of the device without a user having to be logged on. (Default setting)

Procedure

In the Windows Task Scheduler, create a task which starts the ODK application on start-up of the device.

Note

User Service Mode

To use the ODK application in Service Mode, configure the security options in the dialog "Create task" in such a way that a user does not have to be logged on for starting the task.

Under Windows 10 activate the option "Run whether user is logged on or not".

Result

The "Connect" method of IRuntime wait for a maximum of ten minutes after the start of the ODK application until the Runtime has started up.

4.2 Shutdown behavior

You have the option to be notified by the system on shutdown of Runtime, for example, to start cleanup work on the client.

For this purpose, subscribe the system tag "@SystemActivationState" for monitoring. "@SystemActivationState" signals whether Runtime is active and can have the following values:

- System startup in progress (1)
- System started (activated) (2)
- System stopped (3)
- System shutdown in progress (4)
- System restart in progress (5)

4.3 Restart behavior

Value 4 is the trigger to start cleanup work.

Note

Interface calls on shutdown

Do not call any functions of the ODK interfaces while the system is shut down.

4.3 Restart behavior

Tags subscribed for monitoring

After Runtime is restarted, you can continue to use the existing subscriptions.

Syntax of the alarm filter

With an AlarmSubscription, a filter can be transferred so that not all active alarms of the alarm system are notified, but only those which match the filter. The filter syntax is based on SQL syntax. However, only the WHERE instruction is relevant. The keyword "WHERE" must be omitted.

Operators

The following operators can be used in the filter string of the alarm filter:

Operator	Description	Example
=	equal to	Name = 'Recipe246'
<>	not equal	Value <> 0.0
>	greater than	Value > 25.0
<	less than	Value < 75.0
>=	greater than or equal to	Value >= 25.0
<=	less than or equal to	Value <= 75.0
OR,	logical OR	State = 1 OR State = 3
AND, &&	logical AND	Value >= 25.0 AND Value <= 75.0
BETWEEN	within a range	Value BETWEEN 25.0 AND 75.0
NOT BETWEEN	outside a range	Value NOT BETWEEN 25.0 AND 75.0
LIKE string	corresponds to the string <i>string</i>	Name LIKE 'Motor*'
NOT LIKE string	does not correspond to the string <i>string</i>	Name NOT LIKE 'Valve*'
IN (v1, v2, ...)	corresponds to one or more values	State IN (1, 4, 7)
NOT IN (v1, v2, ...)	does not correspond to one or more values	State NOT IN (0, 2, 3, 5, 6)
(...)	brackets expressions	Value <= 75.0 AND (State = 1 OR State = 3)

Precedence of the operators:

Rank	Operators
1	<ul style="list-style-type: none"> Relational operators: =, <>, >, <, >=, <= LIKE IN BETWEEN
2	NOT
3	AND, &&

Rank	Operators
4	OR,
5	

Permitted wildcards:

Wildcard	Description	Example
*	Replaces 0 to more characters	Name LIKE 'Motor*' Reference = <1.*.15>1
?	Replaces 1 character	Name = 'Recipe?'

See also

[IAlarmSubscription \(Page 94\)](#)

[IPlantObjectAlarmSubscription \(Page 136\)](#)

[IPlantObjectAlarmSubscription \(Page 358\)](#)

[IAlarmSubscription \(Page 295\)](#)

Locale IDs of the supported languages

At the AlarmSubscription, there is a Language property which defines the language of the alarm filter and the language of the alarm texts. In this case, a locale ID from the table below must be entered.

The following table contains the Microsoft locale IDs of the languages supported in the TIA Portal:

Language	Country/Region	Locale ID
Afrikaans	South Africa	1078
Albanian	Albania	1052
Armenian	Armenia	1067
Azerbaijani (Cyrillic)	Azerbaijan	2092
Azerbaijani (Latin)	Azerbaijan	1068
Basque	Basque country	1069
Belarusian	Belarus	1059
Bulgarian	Bulgaria	1026
Chinese	Hong Kong S.A.R.	3076
Chinese	Macao S.A.R.	5124
Chinese	Singapore	4100
Chinese	Taiwan	1028
Chinese	PR China	2052
Danish	Denmark	1030
German	Germany	1031
German	Liechtenstein	5127
German	Luxembourg	4103
German	Austria	3079
German	Switzerland	2055
English	Australia	3081
English	Belize	10249
English	United Kingdom	2057
English	Ireland	6153
English	Jamaica	8201
English	Canada	4105
English	Caribbean	9225
English	New Zealand	5129
English	Philippines	13321
English	Zimbabwe	12297
English	South Africa	7177
English	Trinidad and Tobago	11273
English	USA	1033
Estonian	Estonia	1061

Language	Country/Region	Locale ID
Faroese	Faroe Islands	1080
Finnish	Finland	1035
French	Belgium	2060
French	France	1036
French	Canada	3084
French	Luxembourg	5132
French	Monaco	6156
French	Switzerland	4108
Galician	Galicia	1110
Georgian	Georgia	1079
Greek	Greece	1032
Hindi	India	1081
Indonesian	Indonesia	1057
Icelandic	Iceland	1039
Italian	Italy	1040
Italian	Switzerland	2064
Japanese	Japan	1041
Kazakh	Kazakhstan	1087
Catalan	Catalonia	1027
Kyrgyz	Kyrgyzstan	1088
Konkani	India	1111
Korean	Korea	1042
Croatian	Croatia	1050
Latvian	Latvia	1062
Malay	Brunei Darussalam	2110
Malay	Malaysia	1086
Macedonian	Macedonia, FYRM	1071
Mongolian (Cyrillic)	Mongolia	1104
Dutch	Belgium	2067
Dutch	Netherlands	1043
Norwegian (Bokmal)	Norway	1044
Norwegian (Nynorsk)	Norway	2068
Polish	Poland	1045
Portuguese	Brazil	1046
Portuguese	Portugal	2070
Romanian	Romania	1048
Russian	Russia	1049
Sanskrit	India	1103
Swedish	Finland	2077
Swedish	Sweden	1053
Serbian (Cyrillic)	Serbia and Montenegro (formerly)	3098

Language	Country/Region	Locale ID
Serbian (Latin)	Serbia and Montenegro (formerly)	2074
Slovakian	Slovakia	1051
Slovenian	Slovenia	1060
Spanish	Argentina	11274
Spanish	Bolivia	16394
Spanish	Chile	13322
Spanish	Costa Rica	5130
Spanish	Dominican Republic	7178
Spanish	Ecuador	12298
Spanish	El Salvador	17418
Spanish	Guatemala	4106
Spanish	Honduras	18442
Spanish	Columbia	9226
Spanish	Mexico	2058
Spanish	Nicaragua	19466
Spanish	Panama	6154
Spanish	Paraguay	15370
Spanish	Peru	10250
Spanish	Puerto Rico	20490
Spanish (modern)	Spain	3082
Spanish	Uruguay	14346
Spanish	Venezuela	8202
Swahili	Kenya	1089
Tatar	Russia	1092
Thai	Thailand	1054
Czech	Czech Republic	1029
Turkish	Turkey	1055
Ukrainian	Ukraine	1058
Hungarian	Hungary	1038
Uzbek (Cyrillic)	Uzbekistan	2115
Uzbek (Latin)	Uzbekistan	1091
Vietnamese	Vietnam	1066

Code samples

ODK is supplied with code samples for using the Runtime interfaces. Open the local folder to which you have extracted the file "Support\Openness \Siemens.Unified.Openness_SDK_<version number>.zip".

You will find the code samples in the subfolder "\ODK\samples".

Using the code samples in the help

To reduce the complexity of the code samples and enable better readability, the examples in the help deliberately exclude troubleshooting and freeing up memory.

The application programmer must add these elements during programming.

Constructs affected under C#

- Exception handling with try...catch...finally

```
try
{
    ...
}
catch (Exception ex)
{
    ...
}
finally
{
}
```

- Freeing up memory with Dispose or using (...)
 - See also section Releasing objects (Page 27).

For a description of how to evaluate ODK-specific errors, see section Error-handling interfaces (Page 36).

Constructs affected under C++

- Freeing up allocated memory
- Null pointer check: `if (pObject != nullptr) {...}`
- Error code check: `if (CF_SUCCEEDED(errorCode) {...}`

For a description of how to evaluate ODK-specific errors, see section Error codes of the C++ interfaces (Page 189).

Description of the C# interfaces

8.1 Releasing objects

Creating objects with GetObject

In .NET-ODK, you create the objects with the "GetObject" method, for example:

```
ITagSet odkTagSet = runtime.GetObject<ITagSet>();
```

Memory is created internally for the object. In .NET, the Garbage Collector automatically releases the memory when an object is no longer needed. However, the memory is released at an indefinite time.

Note

The indefinite execution of the Garbage Collector may cause the memory to increase and appear as if it is not being released again. The real reason for this is that the Garbage Collector has not yet started!

Releasing objects created with GetObject

The ODK client should release the memory of objects created using the "GetObject" method as soon as the object is no longer needed.

The following cases must be distinguished here:

- Using the objects by synchronous ODK methods
- Using the objects by asynchronous ODK methods

Example when using synchronous ODK methods

When releasing objects used by synchronous ODK methods, use the keyword `using`:

Copy code

```
try
{
    using (ITag myTag = runtime.GetObject<ITag>("Tag1"))
    {
        IProcessValue value = myTag.Read(HmiReadType.Cache);    // Reads synchronous
    }
}
catch (OdkException ex)
{
    System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
}
```

Example when using asynchronous ODK methods

When you release objects that are used by asynchronous ODK methods, you can use the "Dispose" method. This can be called in the callback method:

Copy code

```
try
{
    ITagSet odkTagSet = runtime.GetObject<ITagSet>();
    odkTagSet.Add(new string[] { "Tag1", "Tag2" });

    // Assign callback function
    odkTagSet.OnReadResult += odkTagSet_OnReadResult;
    odkTagSet.ReadAsync(); // Reads asynchronous
}
catch (OdkException ex)
{
    System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
}

void odkTagSet_OnReadResult(ITagSet sender, IList<IProcessValue> values, bool completed)
{
    try
    {
        ...
    }
    catch (OdkException ex)
    {
        ...
    }
    finally
    {
        if (null != sender)
        {
            sender.Dispose(); // Release memory
        }
    }
}
```

8.2 Interfaces of the Runtime environment

8.2.1 IRuntime

Description

The C# interface "IRuntime" specifies properties and methods for handling the Runtime system. The "Connect" method of the "Runtime" class is called to establish the connection to the Runtime system.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

Members "Runtime"

The class implements the following method:

"Connect" method

Connect to a Runtime project.

- Connect to locally run Runtime project. Logged-on Windows user is authenticated.
`IRuntime Connect()`
- Connect to locally run Runtime project. The logged-on Windows user is not authenticated; instead, the user is specified as a parameter.

Note

Can only be used in a future version!

```
IRuntime Connect(string userName, string password)
```

- `user`
User name
- `password`
Password

- Connect to a specific Runtime project. Logged-on Windows user is authenticated.

Note

Can only be used in a future version!

```
IRuntime Connect(string value)
value
Name of a Runtime project
```

- Connect to a specific Runtime project. The logged-on Windows user is not authenticated; instead, the user is specified as a parameter.

Note

Can only be used in a future version!

```
IRuntime Connect(string value, string userName, string password)
```

- `value`
Name of a Runtime project
- `user`
User name
- `password`
Password

"Dispose" method

Enable Runtime system with all resources.

```
void Dispose()
```

Members "IRuntime"

The following properties and methods are specified in the interface:

"ProjectName" property

Name of the current project

Note

Can only be used in a future version!

```
string ProjectName { get; }
```

"UserName" property

Name of the logged-on user

```
string UserName { get; }
```

"Product" property

Return version information and installed options of the Runtime system as "IProduct" object.

```
IProduct Product { get; }
```

"GetObject" method

Create a new instance of an object type T in a project.

```
T GetObject<T>(params object[] parameters)
```

The object type T adopts the following values:

- ITag, ITagSet or ITagSetQCD
Access to tags
- IAlarm, IAlarmSet or IAlarmSubscription
Access to alarm logging
- ILoggedTag or ILoggedTagSet
Access to logging tags
- IAlsrnLogging or IAlarmLoggingSubscription
Access to logged alarms
- IUserManagement
Access to user management
- IConnection or IConnectionSet
Access to connections

parameters

Optional: A name or array with names of objects of the respective object type

"GetOption" method

Return an installed option of the Runtime system as "IOption" object using the name.

```
IOption GetOption(string optionName)
```

optionName
Name of the installed option

Example

Initialize the ODK and establish a connection to the active project of the Runtime system.

Copy code

```
public static IRuntime runtime = null;

public void Connect()
{
    try
    {
        // Connect to running project
        runtime = Siemens.Runtime.HmiUnified.Runtime.Connect();
    }
    catch (Exception ex)
    {
        System.Console.WriteLine(string.Format("OdkException occured {0}", ex.Message));
    }
}
```

Initialize an "IProduct" object and output the technical product version of the Runtime system:

```
public void GetVersionInfo(IRuntime runtime)
{
    IProduct product = runtime.Product;
    IVersionInfo version = product.Version;
    System.Console.WriteLine(string.Format("Product version: {0}.{1}.{2}.{3}",
    version.Major, version.Minor, version.ServicePack, version.Update));

    ...
}
```

Access a tag with the name "Tag1":

```
public void ReadSingleTagSync()
{
    try
    {
        using (ITag myTag = runtime.GetObject<ITag>("Tag1"))
        {
            ...//further tag processing
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occured {0}", ex.Message));
    }
}
```

See also

[IProduct](#) (Page 32)

[IErrorResult](#) (Page 36)

8.2.2 IProduct

Description

The C# interface "IProduct" specifies properties for handling product information of the Runtime system.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

Members

The following properties are specified in the interface:

"Options" property

Return installed options of the Runtime system as a list of "IOption" objects.

```
IList<IOption> Options { get; }
```

"Version" property

Return version structure of the Runtime system as "IVersionInfo" object.

```
IVersionInfo Version { get; }
```


Example

The version of the "IProduct" object is read out and iterated via the installed "IOption" objects:

Copy code

```
public void GetVersionInfo(IRuntime runtime)
{
    try
    {
        IProduct product = runtime.Product;
        IVersionInfo version = product.Version;

        if (product.Options.Count > 0)
        {
            foreach (IOption op in product.Options)
            {
                IVersionInfo opVersion = op.Version;

                // Iterate through options and get version
                ...
            }
        }
    }
    catch (Exception ex)
    {
        System.Console.WriteLine(string.Format("Exception occurred {0}", ex.Message));
    }
}
```

See also

[IRuntime \(Page 28\)](#)

[IOption \(Page 33\)](#)

[IVersionInfo \(Page 35\)](#)

8.2.3 IOption

Description

The C# interface "IOption" specifies properties and methods for handling installed product options of the Runtime system.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

Members

The following properties and methods are specified in the interface:

"Name" property

Name of an installed option of the Runtime system

```
String Name { get; }
```

"Version" property

Return version structure of an installed option of the Runtime system as "IVersionInfo" object.

```
IVersionInfo Version { get; }
```

"GetObject" method

Create new instance of an object type T of the option.

```
T GetObject<T>(params object[] parameters)
```

- T
The value defines a specific object type of the option.
- parameters
Optional: A parameter or array with parameters for the object type of the option

Example

Instantiate and use installed options with name "MyOptionName":

Copy code

```
public void GetOptionObject()
{
    //load option component by name
    IMyOption rtOption = (IMyOption)runtime.GetOption("MyOptionName");

    //create a instance of the option object IMyOptionObject
    IMyOptionObject optionObject = rtOption.GetObject<IMyOptionObject>();
    try
    {
        string strMethod = optionObject.MyMethod();
        string strProperty = optionObject.MyProperty;
    }
    catch (OdkException ex)
    {
        //It is an option error?
        if (ex.ErrorSubCategory == MyOptionConstants.MYOPTION_ERRORCATEGORY)
        {
            //Handle option specific error
            if (ex.ErrorCode == (uint)MyErrorCodes.E_UNKNOWN_NAME)
            {
                //get error description
                string errorDescription = ex.Message;
            }
        }
    }
}
```

See also

IProduct (Page 32)

IVersionInfo (Page 35)

8.2.4 IVersionInfo**Description**

The C# interface "IVersionInfo" specifies properties for handling version information of the Runtime system.

Members

The following properties are specified in the interface:

"Major" property

Main version of the Runtime system or of an installed option

```
uint16 Major { get; }
```

"Minor" property

Minor version of the Runtime system or of an installed option

```
uint16 Minor { get; }
```

"ServicePack" property

Service pack of the Runtime system or of an installed option

```
uint16 ServicePack { get; }
```

"Update" property

Update version of the Runtime system or of an installed option

```
uint16 Update { get; }
```

Example

Information about the installed "IOption" options of the runtime system is output:

Copy code

```
public void GetVersionInfo(IRuntime runtime)
{
    try
    {
        IProduct product = runtime.Product;
        IVersionInfo version = product.Version;
        System.Console.WriteLine(string.Format("Product version: {0}.{1}.{2}.{3}",
version.Major, version.Minor, version.ServicePack, version.Update));

        if (product.Options.Count > 0)
        {
            foreach (IOption op in product.Options)
            {
                IVersionInfo opVersion = op.Version;

                // Iterate through options and get version
                System.Console.WriteLine(string.Format("Option name: {0}", op.Name));
                System.Console.WriteLine(string.Format("Option version: {0}.{1}.{2}.{3}",
opVersion.Major, opVersion.Minor, opVersion.ServicePack, opVersion.Update));
            }
        }
    }
    catch (Exception ex)
    {
        System.Console.WriteLine(string.Format("Exception occurred {0}", ex.Message));
    }
}
```

See also

[IProduct \(Page 32\)](#)

[IOption \(Page 33\)](#)

8.3 Error-handling interfaces

8.3.1 IErrorResult

Description

The C# interface "IErrorResult" specifies properties of error results in runtime.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

Members

The following properties are specified in the interface:

"Error" property

Error code of an error

```
int32 Error { get; }
```

"Name" property

Name of the associated object of the data source

```
string Name { get; }
```

Example

Error output when writing a TagSet:

Copy code

```
public void WritePartlyNotExistingTagSetSync()
{
    try
    {
        using (ITagSet odkTagSet = runtime.GetObject<ITagSet>())
        {
            odkTagSet.Add("Tag1", 1);
            odkTagSet.Add("Tag2", 2);
            odkTagSet.Add("NotExistingTag1", 1);
            odkTagSet.Add("NotExistingTag2", 2);

            IList<IErrorResult> writeResult = odkTagSet.Write();

            foreach (var result in writeResult)
            {
                if (result.Error != 0)
                {
                    System.Console.WriteLine(string.Format("Write tag '{0}' failed, error
code {1}", result.Name, result.Error));
                }
            }
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}
```

See also

[IRuntime \(Page 28\)](#)

[IErrorInfo \(Page 38\)](#)

8.3.2 IErrorInfo

Description

The C# interface "IErrorInfo" specifies methods and properties for handling error codes.

Members

The following properties and methods are specified in the interface:

"Error" property

Error code of an error

```
int32 Error { get; }
```

"GetErrorDescription" method

Output an error description for the error code.

```
string GetErrorDescription(uint32 Error)
```

Error

Error code that is passed by the ODK client.

See also

IErrorResult (Page 36)

8.3.3 OdkException

Description

In the case of exceptions, the ODK triggers an OdkException in the .Net environment. The OdkException can be caught by try-catch blocks and evaluated.

The "OdkException" class inherits all properties and methods of the .NET class "Exception".

Members "OdkException"

The following objects and methods are also implemented in the "OdkException" class for all properties and methods of the .NET class "Exception".

"OdkException" method

- Trigger exception without message.
`OdkException()`
- Trigger exception with message and error description.

```
OdkException(string message)
```

message

Description of the error

- Trigger exception with message and error description. Trigger additional exception with reference to triggering exception.

```
OdkException(string message, Exception innerException)
```

- `message`
Description of the error

- `innerException`
Triggering exception

- Trigger exception with serialized data.

```
OdkException(SerializationInfo info, StreamingContext context)
```

- `info`
Serialized data of the exception

- `context`
Describes the origin or target of the serialized data.

"ErrorCode" property

Error code of the exception

```
UInt32 ErrorCode { get; }
```

"ErrorSubCategory" property

Subcategory of an error code

```
UInt32 ErrorSubCategory { get; }
```

8.4 Interfaces of the tags

Example

Exception handling using the example of optional components:

Copy code

```
public void GetOptionObject()
{
    //load option component by name
    IMyOption rtOption = (IMyOption)runtime.GetOption("MyOptionName");
    //create a instance of the option object IMyOptionObject
    IMyOptionObject optionObject = rtOption.GetObject<IMyOptionObject>();
    try
    {
        string strMethod = optionObject.MyMethod();
        string strProperty = optionObject.MyProperty;
    }
    catch (OdkException ex)
    {
        //It is an option error?
        if (ex.ErrorSubCategory == MyOptionConstants.MYOPTION_ERRORCATEGORY)
        {
            //Handle option specific error
            if (ex.ErrorCode == (uint)MyErrorCodes.E_UNKNOWN_NAME)
            {
                //get error description
                string errorDescription = ex.Message;
            }
        }
    }
}
```

8.4 Interfaces of the tags

8.4.1 IProcessValue

Description

The C# interface "IProcessValue" specifies properties and methods for values of process tags of the Runtime system. The "IProcessValue" interface provides values from the result of a read operation or monitoring.

Members

The following properties are specified in the interface:

"Name" property

Name of the tag

```
string Name { get; }
```


"Value" property

Value of the tag at the moment of the read operation.

```
object Value { get; }
```

"Quality" property

Quality code of the read operation of the tag.

```
uint32 Quality { get; }
```

"TimeStamp" property

Time stamp of the last successful read operation of the tag.

```
DateTime TimeStamp { get; }
```

"Error" property

Error code of the last read or write operation of the tag.

```
int32 Error { get; }
```

Example

Output properties of the "IProcessValue" object that is returned by the ITag.Read method:

Copy code

```
public void ReadSingleTagSync()
{
    try
    {
        using (ITag myTag = runtime.GetObject<ITag>("Tag1"))
        {
            IProcessValue value = myTag.Read(HmiReadType.Cache); // Reads value from Cache
            System.Console.WriteLine(string.Format("Name: {0} Timestamp: {1} Value: {2}
Quality: {3}", value.Name, value.TimeStamp, value.Value, value.Quality));
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}
```

See also

[ITag \(Page 42\)](#)

[ITagSet \(Page 44\)](#)

[ITagSetQCD \(Page 52\)](#)

8.4.2 ITag

Description

The C# interface "ITag" specifies properties and methods for handling tags of the Runtime system.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

The methods trigger an exception in the case of an error.

Members

The following properties and methods are specified in the interface:

"Name" property

Name of the tag that is read with the "Read" method.

```
string Name { get; set; }
```

"Read" method

Read process value and properties of the tag synchronously from the Runtime system.

```
IProcessValue Read(HmiReadType type = HmiReadType.Cache)
```

type

The enumeration "HmiReadType" specifies the origin of the tag value:

- `HmiReadType.Cache` (default parameter): Reads the tag value from the tag image. If no subscription exists, the tag is subscribed.
- `HmiReadType.Device`: Reads the tag value directly from the AS. The tag image is not used.

"Write" method

Write process value of the tag synchronously in the Runtime system.

```
void Write(  
    object value,  
    HmiWriteType type = HmiWriteType.NoWait)
```

- value
Value of the tag
- type
The enumeration "HmiWriteType" specifies whether the method waits for the write operation to be completed:
 - `HmiWriteType.NoWait` (default parameter): Writes the tag value without waiting. Errors for the write operation are not detected.
 - `HmiWriteType.Wait`: Waits until the tag value is written in the AS. If an error occurs during the write operation, an exception is triggered.

"WriteQCD" method

Write process value with quality code of the tag synchronously in the Runtime system. The tag also has a freely definable time stamp. You can use this to acquire past external measured values, for example.

Note**Reaction to external tags**

For external tags, the method only writes the tag value. The QualityCode and time stamp are set internally by the system.

```
void Write(  
    object value,  
    DateTime timeStamp,  
    uint32 qualityCode,  
    HmiWriteType type = HmiWriteType.NoWait)
```

- value
Value of the tag
- timeStamp
Time stamp of the tag
- qualityCode
Quality code of the tag
- type
The enumeration "HmiWriteType" specifies whether the method waits for the write operation to be completed:
 - HmiWriteType.NoWait (default parameter): Writes the tag value without waiting. Errors for the write operation are not detected.
 - HmiWriteType.Wait: Waits until the tag value is written in the AS. If an error occurs during the write operation, an exception is triggered.

"WriteWithOperatorMessage" method

Write process value of the tag synchronously in the Runtime system and create operator message. In addition to the reason, the operator message contains the old and new value, the user and host names and the unit.

```
void WriteWithOperatorMessage(  
    object value,  
    string reason)
```

- value
Value of the tag
- reason
Reason of the value change for message

Example

Read and write tag synchronously:

Copy code

```
public void ReadSingleTagSync()
{
    try
    {
        using (ITag myTag = runtime.GetObject<ITag>("Tag1"))
        {
            IProcessValue value = myTag.Read(HmiReadType.Cache); // Reads value from Cache
            System.Console.WriteLine(string.Format("Name: {0} Timestamp: {1} Value: {2}
Quality: {3}", value.Name, value.TimeStamp, value.Value, value.Quality));
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occured {0}", ex.Message));
    }
}

public void WriteSingleTagSync()
{
    try
    {
        using (ITag odkTag = runtime.GetObject<ITag>("Tag1"))
        {
            int value = 5;
            odkTag.Write(value, HmiWriteType.NoWait); // Writes value without waiting that
value has been written to PLC
            IProcessValue pvalue = odkTag.Read(HmiReadType.Cache);
        }
    }
    catch (Exception ex)
    {
        System.Console.WriteLine(string.Format("Exception occured {0}", ex.Message));
    }
}
```

See also

[IProcessValue \(Page 40\)](#)

[ITagSet \(Page 44\)](#)

8.4.3 ITagSet

Description

The C# interface "ITagSet" specifies properties, methods and events for optimized access to several tags of the Runtime system.

After initialization of the "ITagSet" object, you can execute read and write access to multiple tags in one call. Simultaneous access demonstrates better performance and lower communication load than single access to multiple tags.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

The methods trigger an exception in the case of an error.

Note

You have the option to enumerate via a TagSet and to access a special element via the name. This is useful if you want to change the value of a tag in the TagSet for a write operation. To write the values in the process image, a "Write" or "WriteAsync" method must first be called:

```
MyTagSet.Add("MyTag1", "MyTag2");
MyTagSet["MyTag1"] = 5; // Set value to 5 for write operation
MyTagSet.Write();
```

Members

The following properties, methods and events are specified in the interface:

"ContextId" property

Identification characteristics of a TagSet. If several TagSets are used to read tags, you can assign the response to the request via ContextId.

Default value -1: The ContextId is not used.

```
int32 ContextId { get; set; }
```

"Count" property

Number of tags of a TagSet list

```
int32 Count { get; }
```

"Add" method

Add tag to a TagSet.

Add tag with or without process value to the TagSet:

```
void Add(ICollection<string> tagNames)
```

```
tagNames
```

List with tag names for TagSet

or

```
void Add(string tagName, object value = null)
```

- `TagName`
Name of the tag for TagSet
- `value`
New value of the tag, default: No value

"Remove" method

Remove individual tag from a TagSet.

```
void Remove(string tagName)
```

tagName

Name of the tag that is removed from TagSet.

"Clear" method

Remove all tags from a TagSet.

```
void Clear()
```

"Read" method

Read process values and properties of all the tags of a TagSet synchronously from the Runtime system.

```
ICollection<IProcessValue> Read(HmiReadType type = HmiReadType.Cache)
```

type

The enumeration "HmiReadType" specifies the origin of the tag value:

- `HmiReadType.Cache` (default): Reads the tag values from the tag image. If no subscription exists, the tag is subscribed.
- `HmiReadType.Device`: Reads the tag values directly from the automation system. The tag image is not used.

"ReadAsync" method

Read process values and properties of all the tags of a TagSet asynchronously from the Runtime system.

```
void ReadAsync(HmiReadType type = HmiReadType.Cache)
```

type

The enumeration "HmiReadType" specifies the origin of the tag value:

- `HmiReadType.Cache` (default): Reads the tag values from the tag image. If no subscription exists, the tag is subscribed.
- `HmiReadType.Device`: Reads the tag values directly from the automation system. The tag image is not used.

"Write" method

Write process values of all tags of a TagSet synchronously in the Runtime system.

```
ICollection<IErrorResult> Write(HmiWriteType type = HmiWriteType.NoWait)
```

type

The enumeration "HmiWriteType" specifies whether the method waits for the write operation to be completed:

- `HmiWriteType.NoWait` (default): Writes the tag values without waiting. Errors for the write operation are not detected.
- `HmiWriteType.Wait`: Waits until the tag values are written in the automation system. The associated errors are written.

"WriteAsync" method

Write process values of all tags of a TagSet asynchronously in the Runtime system.

The method always has the `HmiWriteType.Wait` type and waits until the tag value has been written in the automation system. If an error occurs during the write operation, it is reported via the AsyncHandler.

```
void WriteAsync()
```

"WriteWithOperatorMessage" method

Write process values of all tags of a TagSet synchronously in the Runtime system and create operator messages. In addition to the reason, the operation messages contain the old and new value, the user and host names and the unit.

```
void WriteWithOperatorMessage(string reason)
```

```
reason
```

Reason of the value change for message

"Subscribe" method

Subscribe all tags of a TagSet asynchronously for cyclic monitoring of the process values.

Note**Tags from IO devices with the "Cyclic in operation" acquisition mode**

For a tag with the acquisition mode "Cyclic in operation", the value stored in the process image when Subscribe is called might be outdated. OnAdd therefore only provides the QualityCode "uncertain". Only value changes made after the Subscribe call provide the current value and the QualityCode "good".

```
void Subscribe()
```

"CancelSubscribe" method

Cancel monitoring of all tags of a TagSet.

```
void CancelSubscribe()
```

"OnReadResult" event

After completion of the read operation of the "ReadAsync" method, the event calls an instance of the "OnReadResultHandler" delegate.

Declares the event and the event handler for asynchronous read operations.

```
event OnReadResultHandler OnReadResult
```

"OnWriteResult" event

After completion of the write operation of the "WriteAsync" method, the event calls an instance of the "OnWriteResultHandler" delegate.

Declares the event and the event handler for asynchronous write operations.

```
event OnWriteResultHandler OnWriteResult
```

8.4 Interfaces of the tags

"OnDataChanged" event

After a process value change of a monitored TagSet, the event calls an instance of the "OnDataChangedHandler" delegate.

Declares the event and the event handler for process value changes.

```
event OnDataChangedHandler OnDataChanged
```

"OnReadResultHandler" delegate

Specifies the signature of the event handling method for the "OnReadResult" event of a TagSet.

```
void OnReadResultHandler(  
    ITagSet sender,  
    IList<IProcessValue> values,  
    bool completed)
```

- sender
The read out "TagSet" object
- values
Event data as a list of "IProcessValue" objects of the read tag
- completed
Status of the asynchronous transfer:
 - True: All values of the TagSet are read.
 - False: Not all values of the TagSet are read yet.

"OnWriteResultHandler" delegate

Specifies the signature of the event handling method for the "OnWriteResult" event of a TagSet.

```
void OnWriteResultHandler(  
    ITagSet sender,  
    IList<IErrorResult> values,  
    bool completed)
```

- sender
The written "TagSet" object
- values
Error during write operations of tags as "IErrorResult" object
- completed
Status of the asynchronous transfer:
 - True: All values of the TagSet are written.
 - False: Not all values of the TagSet are written yet.

"OnDataChangedHandler" delegate

Specifies the signature of the event handling method for the "OnDataChanged" event of a TagSet.


```
void OnDataChangedHandler(
    ITagSet sender,
    IList<IProcessValue> values)
```

- sender
The monitored "TagSet" object
- values
Event data as a list of "IProcessValue" objects of the changed process values

Example

Read TagSet synchronously and write with change:

Copy code

```
public void ReadTagSetSync()
{
    try
    {
        using (ITagSet odkTagSet = runtime.GetObject<ITagSet>())
        {
            odkTagSet.Add(new string[] { "Tag1", "Tag2" });
            IList<IProcessValue> values = odkTagSet.Read();
            foreach (var value in values)
                System.Console.WriteLine(string.Format("Name: {0} Timestamp: {1} Value: {2}
Quality: {3}", value.Name, value.TimeStamp, value.Value, value.Quality));
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}

public void WriteTagSetSyncWithChange()
{
    try
    {
        using (ITagSet odkTagSet = runtime.GetObject<ITagSet>())
        {
            odkTagSet.Add(new string[] { "Tag1", "Tag2" });
            // Modify the value of a tag in the tagset and write
            odkTagSet["Tag1"] = 5;
            odkTagSet.Write();
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}
```

Define functions for asynchronous write operations and define the monitoring of TagSets.

8.4 Interfaces of the tags

Monitor events "OnWriteResult" or "OnDataChanged" and call the event handling methods "odkTagSet_OnWriteComplete" or "odkTag_OnDataChanged" on occurrence:

Copy code

```

public void WriteTagSetAsync()
{
    try
    {
        ITagSet odkTagSet = runtime.GetObject<ITagSet>();
        odkTagSet.Add("Tag1", 1);
        odkTagSet.Add("Tag2", 2);
        // Assign callback function
        odkTagSet.OnWriteResult += odkTagSet_OnWriteResult;
        odkTagSet.WriteAsync();
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}

public void odkTagSet_OnWriteResult(ITagSet sender, IList<KeyValuePair<string, int>>
values, bool completed)
{
    System.Console.WriteLine("odkTagSet_OnWriteComplete");
    sender.Dispose();
}

public void SubscribeTagSet()
{
    try
    {
        ITagSet odkTagSet = runtime.GetObject<ITagSet>();
        odkTagSet.Add(new string[] { "Tag1", "Tag2" });
        // Assign callback function
        odkTagSet.OnDataChanged += odkTagSet_OnDataChanged;
        odkTagSet.Subscribe();
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}

public void odkTagSet_OnDataChanged(ITagSet sender, IList<IProcessValue> pItems)
{
    try
    {
        foreach (var value in pItems)
            System.Console.WriteLine(string.Format("Name: {0} Timestamp: {1} Value: {2}
Quality: {3}", value.Name, value.TimeStamp, value.Value, value.Quality));
        // For test purpose: Cancel subscription after first notification
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
    finally
    {
        if (null != sender)
        {
            sender.CancelSubscribe();
        }
    }
}

```

8.4 Interfaces of the tags

Copy code

```
        sender.Dispose();
    }
}
```

See also

[IProcessValue](#) (Page 40)

[ITag](#) (Page 42)

8.4.4 ITagSetQCD

Description

The C# interface "ITagSetQCD" specifies properties, methods and events for optimized writing of several tags of the Runtime system. The tags also have a freely definable time stamp and quality code. You can use this to acquire past external measured values, for example.

Note

Reaction to external tags

For external tags, the method only writes the tag value. The QualityCode and time stamp are set internally by the system.

After initialization of the "ITagSetQCD" object, you can have read access to multiple tags in one call. Simultaneous access demonstrates better performance and lower communication load than single access to multiple tags.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

The methods trigger an exception in the case of an error.

Note

You have the option to enumerate via a TagSet and to access a special element via the name. This is useful if you want to change the value of a tag in the TagSet for a write operation. To write the values in the process image, a "Write" or "WriteAsync" method must first be called:

```
MyTagSet.Add("MyTag1", "MyTag2");

MyTagSet["MyTag1"] = 5; // Set value to 5 for write operation

MyTagSet.Write();
```

Members

The following properties, methods and events are specified in the interface:

"ContextId" property

Identification characteristics of a TagSet. If several TagSets are used to read tags, you can assign the response to the request via ContextId.

Default value -1: The ContextId is not used.

```
int32 ContextId { get; set; }
```

"Count" property

Number of tags of a TagSet list

```
int32 Count { get; }
```

"Add" method

Add user-defined tag with process value, quality code and time stamp to the TagSet.

```
void Add(string tagName, object value, DateTime timeStamp, uint32  
qualityCode)
```

- `TagName`
Name of the tag for TagSet
- `value`
New value of the tag
- `timeStamp`
Time stamp of the tag
- `qualityCode`
Quality code of the tag

"Remove" method

Remove individual tag from a TagSet.

```
void Remove(string tagName)
```

`tagName`

Name of the tag that is removed from TagSet.

"Clear" method

Remove all tags from a TagSet.

```
void Clear()
```

"Write" method

Write process values of all tags of a TagSet synchronously in the Runtime system.

```
ICollection<IErrorResult> Write(HmiWriteType type = HmiWriteType.NoWait)
```

`type`

The enumeration "HmiWriteType" specifies whether the method waits for the write operation to be completed:

- `HmiWriteType.NoWait` (default): Writes the tag values without waiting. Errors for the write operation are not detected.
- `HmiWriteType.Wait`: Waits until the tag values are written in the automation system. The associated errors are written.

"WriteAsync" method

Write process values of all tags of a TagSet asynchronously in the Runtime system.

The method always has the `HmiWriteType.Wait` type and waits until the tag value has been written in the automation system. If an error occurs during the write operation, it is reported via the AsyncHandler.

```
void WriteAsync()
```

"OnWriteComplete" event

After completion of the write operation of the "WriteAsync" method, the event calls an instance of the "OnWriteCompleteTagSetQCDHandler" delegate.

Declares the event and the event handler for asynchronous write operations with quality codes and time stamps.

```
event OnWriteCompleteTagSetQCDHandler OnWriteComplete
```

"OnWriteCompleteTagSetQCDHandler" delegate

Specifies the signature of the event handling method for the "OnWriteComplete" event of a TagSet with quality code and time stamps.

```
void OnWriteCompleteTagSetQCDHandler(
    ITagSetQCD sender,
    IList<IErrorResult> pItems)
```

- sender
Source of the event
- pItems
Error during write operations of tag as "IErrorResult" object

Example

Write TagSet with time stamp and quality code synchronously:

Copy code

```
public void WriteTagSetQCDSync()
{
    try
    {
        using (ITagSetQCD odkTagSet = runtime.GetObject<ITagSetQCD>())
        {
            odkTagSet.Add("Tag1", 1, DateTime.Now, 128);
            odkTagSet.Add("Tag2", 2, DateTime.Now, 128);

            IList<IErrorResult> writeResult = odkTagSet.Write();
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}
```

See also

IProcessValue (Page 40)

ITagSetQCItem (Page 55)

8.4.5 ITagSetQCItem**Description**

The C# interface "ITagSetQCItem" specifies properties for user-defined values of tags in a TagSetQCD. You can use this to acquire past external measured values, for example.

Note**Reaction to external tags**

For external tags, only the tag value is set. The QualityCode and time stamp are set internally by the system.

The objects "ITagSetQCItem" are not used by the "ITagSetQCD" object and can also be edited with the methods of the "ITagSetQCD" interface.

Members

The following properties are specified in the interface:

"Name" property

Name of the tag

```
string Name { get; set; }
```

"Value" property

Tag value

```
object Value { get; set; }
```

"Quality" property

Quality code of the tag

```
int32 Quality { get; set; }
```

"TimeStamp" property

Time stamp of the tag

```
DateTime TimeStamp { get; set; }
```

See also

ITagSetQCD (Page 52)

8.4.6 ILoggedTagValue

Description

The C# interface "ILoggedTagValue" specifies the properties for process values of logging tags of a logging system. The "ILoggedTagValue" interface displays historical process values.

An "ILoggedTagValue" object is a pure data object which maps all properties of the logged process values.

Members

The following properties are specified in the interface:

"Name" property

Name of the logging tag

```
string Name { get; set; }
```

"Value" property

Process value of the logging tag

```
object Value { get; set; }
```

"Quality" property

Quality code of the process value

```
int16 Quality { get; set; }
```

"TimeStamp" property

Time stamp of the process value

```
DateTime TimeStamp { get; set; }
```

"Error" property

Error code of the process value

```
int32 Error { get; }
```

"Flags" property

Context information from the read operation for the process value

```
HmiTagLoggingValueFlags Flags { get; set; }
```

The "HmiTagLoggingValueFlags" enumeration contains the following bit-by-bit-coded values:

- 0: Extra
There are still additional values at the time of the process value.
- 2: Calculated
Process value is calculated.
- 16: Bounding
Process value is a limit value.

- 32: `NoData`
No additional information available
- 64: `FirstStored`
Process value is the first value stored in the logging system.
- 128: `LastStored`
Process value is the last value stored in the logging system.

See also

`ILoggedTag` (Page 57)

`ILoggedTagSet` (Page 58)

8.4.7 `ILoggedTag`

Description

The C# interface "`ILoggedTag`" specifies properties and methods for handling logging tags of a logging system.

The interface inherits the `Dispose()` method of the "`IDisposable`" interface of the .NET framework.

Members

The following properties and methods are specified in the interface:

"Name" property

Name of the logging tag

```
string Name { get; set; }
```

"Read" method

Read logged process values of a logging tag of a time period synchronously from the logging system.

```
IList<ILoggedTagValue> Read(DateTime begin, DateTime end, bool  
boundingValue)
```

- `begin`
Start date of the time period
- `end`
End date of the time period
- `boundingValue`
True, to additionally return high and low limits.

Example

Output process values of the logging tag "Tag1:Tag1Logging1" for a time period:

Copy code

```
public void ReadSingleTag()
{
    var tag = _runtime.GetObject<ILoggedTag>("Tag1:Tag1Logging1");
    var begin = DateTime.UtcNow.AddMinutes(-10);
    var end = DateTime.UtcNow;
    var values = tag.Read(begin, end, true);
    Print(values);
    tag.Dispose();
}
```

See also

[ILoggedTagValue](#) (Page 56)

[ILoggedTagSet](#) (Page 58)

8.4.8 ILoggedTagSet

Description

The C# interface "ILoggedTagSet" specifies properties, methods and events for optimized access to several logging tags of a logging system.

After initialization of the "ILoggedTagSet" object, you can execute read and write access to multiple logging tags in one call. Simultaneous access demonstrates better performance and lower communication load than single access to multiple logging tags.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

Members

The following properties, methods and events are specified in the interface:

"ContextId" property

Identification characteristics of a LoggedTagSet. If several LoggedTagSets are used to read logging tags, you can assign the response to the request via ContextId.

Default value -1: The ContextId is not used.

```
int32 ContextId { get; set; }
```

"Count" property

Number of logging tags of a LoggedTagSet list

```
int32 Count { get; }
```

"Add" method

Add logging tag to a LoggedTagSet.

Add logging tags with or without context to the LoggedTagSet:

```
void Add(ICollection<string> tagNames)
```

tagNames

List with names of logging tags for the LoggedTagSet

or

```
void Add(string tagName)
```

tagName

Name of a logging tag

"Remove" method

Remove individual logging tag from a LoggedTagSet.

```
void Remove(string tagName)
```

tagName

Name of the logging tag that is removed from the LoggedTagSet.

"Clear" method

Remove all logging tags from a LoggedTagSet.

```
void Clear()
```

"Read" method

Read all logging tags of a LoggedTagSet synchronously from the logging system.

```
IList<ILoggedTagValue> Read(DateTime begin, DateTime end, bool  
boundingValue)
```

- begin
Start date of the time period
- end
End date of the time period
- boundingValue
True, to additionally return high and low limits.

"ReadAsync" method

Read all logging tags of a LoggedTagSet asynchronously from the logging system.

```
void ReadAsync(DateTime begin, DateTime end, bool boundingValue)
```

- begin
Start date of the time period
- end
End date of the time period
- boundingValue
True, to additionally return high and low limits.

"Subscribe" method

Subscribe all logging tags of a LoggedTagSet asynchronously for updating the process values following a change. When new process values are logged, they can be processed with the "OnDataChanged" event.

```
void Subscribe()
```

"CancelSubscribe" method

Cancel updating of process values following a change for all logging tags of a LoggedTagSet.

```
void CancelSubscribe()
```

"OnReadComplete" event

After completion of the read operation of the "ReadAsync" method, the event calls an instance of the "OnReadCompleteTagSetHandler" delegate.

Declares the event and the event handler for asynchronous read operations.

```
event OnReadCompleteTagSetHandler OnReadComplete
```

"OnDataChanged" event

After a process value change of a monitored LoggedTagSet, the event calls an instance of the "OnDataChangedTagSetHandler" delegate.

Declares the event and the event handler for process value changes.

```
event OnDataChangedTagSetHandler OnDataChanged
```

"OnReadCompleteTagSetHandler" delegate

Specifies the signature of the event handling method for the "OnReadComplete" event of a LoggedTagSet.

```
void OnReadCompleteTagSetHandler(  
    ILoggedTagSet sender,  
    uint32 errorCode,  
    IList<ILoggedTagValue> values,  
    bool completed)
```

- sender
Source of the event
- errorCode
Error during asynchronous transfer
- values
Event data as a list of "ILoggedTagValue" objects of the read logging tag
- completed
Status of the asynchronous transfer:
 - True: All values of the LoggedTagSet are read.
 - False: Not all values of the LoggedTagSet are read.

"OnDataChangedTagSetHandler" delegate

Specifies the signature of the event handling method for the "OnDataChanged" event of a LoggedTagSet.

```
void OnDataChangedTagSetHandler(
    ILoggedTagSet sender,
    uint32 errorCode,
    IList<ILoggedTagValue> value)
```

- `sender`
Source of the event
- `errorCode`
Error during asynchronous transfer
- `value`
Event data as a list of "ILoggedTagValue" objects with process values of the changed logging tag

Example

Read and output process values of logging tags of a specific time period asynchronously in LoggedTagSet "tagSet":

Copy code

```
public void ReadTagSetAsync()
{
    try
    {
        var begin = DateTime.UtcNow.AddHours(-1);
        var end = DateTime.UtcNow;
        var tagSet = _runtime.GetObject<ILoggedTagSet>();
        tagSet.OnReadComplete += TagSet_OnReadComplete;
        tagSet.Add("Tag1:Tag1Logging1");
        tagSet.Add("Tag2:Tag2Logging1");
        tagSet.ReadAsync(begin, end, true);
    }
    catch (OdkException ex)
    {
        Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}

private void TagSet_OnReadComplete(ILoggedTagSet sender, uint errorCode,
IList<ILoggedTagValue> values, bool completed)
{
    Print(values);
    sender.Dispose();
}
```

8.4 Interfaces of the tags

Output process values of logging tag "Tag1:Tag1Logging1" in case of change:

Copy code

```
public void SubscribeTagSet()
{
    try
    {
        ILoggedTagSet tagSet = _runtime.GetObject<ILoggedTagSet>();
        tagSet.Add("Tag1:Tag1Logging1");
        tagSet.OnDataChanged += tagSet_OnDataChanged;
        tagSet.Subscribe();
    }
    catch (OdkException ex)
    {
        Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}

void tagSet_OnDataChanged(ILoggedTagSet sender, UInt32 errorCode, IList<ILoggedTagValue>
values)
{
    Print(values);
    sender.Dispose();
}
```

See also

[ILoggedTag \(Page 57\)](#)

[ILoggedTagValue \(Page 56\)](#)

8.4.9 ITags

Description

The C# interface "ITags" defines methods with which you can access configured tags.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

Members

"Find" method

Supplies a collection with "ITagAttributes" instances of the specified tag.

```
ICollection<ITagAttributes> Find(ICollection<string> SystemNames,
string Filter = null, UInt32 LanguageID = 0)
```

- `SystemNames`
Collection of `SystemNames` on which the tags were configured.
- (Optional) `Filter`
Filter by name of the tags to restrict the search.
Supports searching with wildcards (*).
- (Optional) `LanguageID`
Language code ID of filter

"FindAsync" method

For asynchronous searching for "ITagAttributes" instances of the specified tags.

```
void FindAsync(IList<string> SystemNames, string Filter = null,
UInt32 LanguageID = 0)
```

- `SystemNames`
Collection of `SystemNames` on which the tags were configured.
- (Optional) `Filter`
Filter by name of the tags to restrict the search.
Supports searching with wildcards (*)
- (Optional) `LanguageID`
Language code ID of filter

"OnTagAttributesRecieved" event

After completion of the "FindAsync" method, the event calls an instance of the "OnTagAttributesRead" delegate.

```
event OnTagAttributesRead OnTagAttributesRecieved;
```

"OnTagAttributesRead" delegate

Specifies the signature of the event handling method for the "OnTagAttributesReceived" event of an "ITagAttributes" instance.

```
public delegate void OnTagAttributesRead(ITags sender,
ICollection<ITagAttributes> tagAttributes, bool completed)
```

- `sender`
Source of the event
- `tagAttributes`
Event data as collection of "ITagAttributes" instances
- `completed`
Status of the asynchronous transfer:
 - True: All tags have been read out.
 - False: Not all tags have been read out yet.

8.4.10 ITagAttributes

Description

The C# interface "ITagAttributes" defines properties of a tag.

Member

"Name" property

The name of the tag. Must be unique throughout the device.

```
string Name { get; }
```

"DisplayName" property

The display name of the tags

```
string DisplayName { get; }
```

"DataType" property

The data type of the tag

```
HmiDataType DataType { get; }
```

"Connection" property

The connection of the tag

The memory location of the tag in the controller is accessed via the connection.

```
string Connection { get; }
```

"AcquisitionCycle" property

The acquisition cycle of tags

If you configure a tag at an object, the acquisition cycle of the tag is displayed at the object.

```
string AcquisitionCycle { get; }
```

"AcquisitionMode" property

The acquisition mode of the tag

```
HmiAcquisitionMode AcquisitionMode { get; }
```

The enumeration "HmiAcquisitionMode" can contain the following values:

- Undefined (0)
- CyclicOnUse (1)
- CyclicContinuous (2)
- OnDemand (3)
- OnChange (4)

"MaxLength" property

The length of the tags.

The length is only available with a string tag. The length is preset for structure tags and cannot be changed.

```
UInt32 MaxLength { get; }
```

"Persistent" property

The persistence of the tags

```
bool Persistent { get; }
```

"InitialValue" property

The start value of the tag

After Runtime starts, the tag retains the start value until an operator or the PLC changes the value.

```
object InitialValue { get; }
```

"InitialMaxValue" property

The start value for the event "On exceeding".

```
object InitialMaxValue { get; }
```

"InitialMinValue" property

The start value for the event "On falling below"

```
object InitialMinValue { get; }
```

"SubstituteValue" property

The substitute value of the tag

If the selected condition occurs, the tag is filled with the substitute value in runtime.

```
object SubstituteValue { get; }
```

"SubstituteValueUsage" property

The condition for using the substitute value of the tag

```
HmiSubstituteValueUsage SubstituteValueUsage { get; }
```

The enumeration "HmiSubstituteValueUsage" can contain the following values:

- None (0)
- InvalidValue (1)
- RangeViolation (2)
- InvalidValueOrRangeViolation (3)

8.4.11 ILoggingTags

Description

The C# interface "ILoggingTags" defines methods with which you can access configured logging tags.

The interface inherits the Dispose() method of the "IDisposable" interface of the .NET framework.

Members

"Find" method

Supplies a collection with "ITagAttributes" instances of the specified logging tag.

```
ICollection<ILoggingTagAttributes> Find(ICollection<string> SystemNames,  
string Filter = null, UInt32 LanguageID = 0)
```

- `SystemNames`
Collection of SystemNames on which the tags were configured.
- (Optional) `Filter`
Filter by name of the tags to restrict the search.
Supports searching with wildcard (*).
Example:
`Tag1 : *` Supplies all logging tags of "Tag1".
- (Optional) `LanguageID`
Language code ID of filter

"FindAsync" method

For asynchronous searching for "ILoggingTagAttributes" instances.

```
void FindAsync(ICollection<string> SystemNames, string Filter = null,  
UInt32 LanguageID = 0)
```

- `SystemNames`
Collection of SystemNames on which the tags were configured.
- (Optional) `Filter`
Filter by name of the tags to restrict the search.
Supports searching with wildcard (*).
Example:
`Tag1. *` Supplies all logging tags of "Tag1".
- (Optional) `LanguageID`
Language code ID of filter

"OnLoggingTagAttributesReadReceived" event

After completion of the "FindAsync" method, the event calls an instance of the "OnLoggingTagAttributesRead" delegate.

```
event OnLoggingTagAttributesRead OnLoggingTagAttributesReadRecieved;
```

"OnLoggingTagAttributesRead" delegate

Specifies the signature of the event handling method for the "OnLoggingTagAttributesRead" event of an "ILoggingTags" instance.

```
public delegate void OnLoggingTagAttributesRead(ILoggingTags sender,
    ICollection<ILoggingTagAttributes> tagAttributes, bool completed);
```

- `sender`
Source of the event
- `tagAttributes`
Event data as collection of "ILoggingTagAttributes" instances
- `completed`
Status of the asynchronous transfer:
 - True: All logging tags are read out.
 - False: Not all logging tags are read out yet.

8.4.12 ILoggingTagAttributes

Description

The C# interface "ILoggingTagAttributes" defines properties of a logging tag.

Member

"Name" property

The name of the tag

```
string Name { get; }
```

"DisplayName" property

The display name of the tags

```
string DisplayName { get; }
```

"DataType" property

The data type of the tag

```
HmiDataType DataType { get; }
```

8.5 Interfaces of the alarms

8.5.1 IAlarmResult

Description

The C# interface "IAlarmResult" specifies properties for accessing properties of active alarms of the Runtime system.

An "IAlarmResult" object is a pure data object which maps all properties of an active alarm.

Members

The following properties are specified in the interface:

"InstanceID" property

InstanceID for an alarm with multiple instances

```
uint32 InstanceID { get; }
```

"SourceID" property

Source at which the alarm was triggered

```
string SourceID { get; }
```

"Name" property

Name of the alarm

```
string Name { get; }
```

"AlarmClassName" property

Name of the alarm class

```
string AlarmClassName { get; }
```

"AlarmClassSymbol" property

Symbol of the alarm class

```
string AlarmClassSymbol { get; }
```

"AlarmParameterValues" property

Parameter values of the alarm

```
ReadOnlyList<object> AlarmParameterValues { get; }
```

"AlarmText1" ... "AlarmText9" properties

Additional texts 1-9 of the alarm

```
string AlarmText1 { get; }
```

...

```
string AlarmText9 { get; }
```

"ChangeReason" property

Trigger event for modification of alarm state

```
uint16 ChangeReason { get; }
```

"Connection" property

Connection by which the alarm was triggered

```
string Connection { get; }
```

"State" property

Current alarm state

```
HmiAlarmState State { get; }
```

The enumeration "HmiAlarmState" can contain the following values:

- Standard (0x00)
- Raised (0x01)
- RaisedCleared (0x02)
- RaisedAcknowledged (0x05)
- RaisedAcknowledgedCleared (0x06)
- RaisedClearedAcknowledged (0x07)
- Removed (0x80)

"StateText" property

Current alarm state as text, for example "Incoming" or "Outgoing"

```
string StateText { get; }
```

"EventText" property

Text that describes the alarm event.

```
string EventText { get; }
```

"InfoText" property

Text that describes an operator instruction for the alarm.

```
string InfoText { get; }
```

"TextColor" property

Text color of the alarm state

```
uint32 TextColor { get; }
```

"BackColor" property

Background color of the alarm state

```
uint32 BackColor { get; }
```

"Flashing" property

Indicates whether the alarm is flashing.

```
bool Flashing { get; }
```

"InvalidFlags" property

Marking of the alarm in case of invalid data

```
byte InvalidFlags { get; }
```

"ModificationTime" property

Time of the last modification to the alarm state

```
DateTime ModificationTime { get; }
```

"RaiseTime" property

Trigger time of the alarm

```
DateTime RaiseTime { get; }
```

"AcknowledgementTime" property

Time of alarm acknowledgment

```
DateTime AcknowledgementTime { get; }
```

"ClearTime" property

Time of alarm reset

```
DateTime ClearTime { get; }
```

"ResetTime" property

Time of alarm reset

```
DateTime ResetTime { get; }
```

"SuppressionState" property

Status of alarm visibility

```
byte SuppressionState { get; }
```

"SystemSeverity" property

Severity of the system error

```
UInt8 SystemSeverity { get; }
```

"Priority" property

Relevance for display and sorting of the alarm

```
UInt8 Priority { get; }
```

"Origin" property

Origin for display and sorting of the alarm

```
string Origin { get; }
```

"Area" property

Origin area for display and sorting of the alarm

```
string Area { get; }
```

"Value" property

Current process value of the alarm

```
object Value { get; }
```

"ValueQuality" property

Quality of the process value of the alarm

```
uint16 ValueQuality { get; }
```

"ValueLimit" property

Limit of the process value of the alarm

```
object ValueLimit { get; }
```

"UserName" property

User name of the operator input alarm

```
string UserName { get; }
```

"UserResponse" property

Expected or required user response to the alarm

```
byte UserResponse { get; }
```

"HostName" property

Name of the host that triggered the alarm.

```
string HostName { get; }
```

"Id" property

ID of the alarm that is also used in the display.

```
UInt32 Id { get; }
```

"AlarmGroupID" property

ID of the alarm group to which the alarm belongs.

```
UInt32 AlarmGroupID { get; }
```

"SourceType" property

Source from which the alarm was generated, e.g. tag-based, controller-based or system-based alarm.

```
HmiAlarmSourceType SourceType { get; }
```

8.5 Interfaces of the alarms

The enumeration "HmiAlarmSourceType" can contain the following values:

- Undefined (0)
- Tag (1)
- Controller (2)
- System (3)
- Alarm (4)

"DeadBand" property

Range of the triggering tag in which no alarms are generated.

```
object DeadBand { get; }
```

"LoopInAlarm" property

Function that navigates from the alarm view to its origin.

```
string LoopInAlarm { get; }
```

"NotificationReason" property

Reason for the notification

```
HmiNotificationReason NotificationReason { get; }
```

The enumeration "HmiNotificationReason" can contain the following values:

- Unknown (0)
- Add (1)
The alarm was added to the filtered result list. The alarm meets the filter criteria that apply to the monitoring.
- Modify (2)
Properties of the alarm were changed, but the alarm is still part of the filtered result list.
- Remove (3)
The alarm was part of the result list, but it no longer meets the filter criteria due to changes to its properties.

Note

Changes to the alarm only lead to notifications again when the alarm meets the filter criteria again. In this case, "NotificationReason" is set to `Add`.

Note**Removing alarm from business logic**

The use case of the client determines whether the client ignores notifications via alarms with the "NotificationReason" `Modify` or `Remove`.

For example:

- State-based monitoring: The client wants to show a list of incoming alarms. All notification reasons are relevant. The client removes an alarm from the list as soon as the notification reason is `Remove`.
 - Event-based monitoring: The client wants to send an email when an alarm comes in. Only the notification reason `Add` is relevant.
-

Example:

An ODK client begins monitoring with the filter criterion "State" = 1. An alarm is triggered. Runtime notifies the ODK client of the "NotificationReason" as follows:

Notification-Reason	Description
Add	<ul style="list-style-type: none">• The "State" property is 1. The alarm has come in.
Modify	<ul style="list-style-type: none">• The "State" property has not changed.• Another property that is not part of the filter criterion has changed, e.g. "Priority".
Remove	The "State" property has changed, e.g. alarm has gone out.

Example

When alarm are incoming, output a selection of properties of the "IAlarmResult" objects:

Copy code

```

public void alarm_OnAlarmHandler(IAAlarmSubscription sender, UInt32 nGlobalError, String
systemName, IList<IAAlarmResult> value, bool completed)
{
    if (0 == nGlobalError)
    {
        try
        {
            AlarmList = new List<IAAlarmResult>();
            foreach (var item in value)
            {
                System.Console.WriteLine(string.Format("InstanceID: {0}", item.InstanceID));
                System.Console.WriteLine(string.Format("AcknowledgementTime: {0}",
item.AcknowledgementTime));
                System.Console.WriteLine(string.Format("AlarmClass: {0}",
item.AlarmClassName));
                System.Console.WriteLine(string.Format("AlarmClassSymbol: {0}",
item.AlarmClassSymbol));
                System.Console.WriteLine(string.Format("Id: {0}", item.Id));
                System.Console.WriteLine(string.Format("AlarmParameterValues: {0}",
item.AlarmParameterValues));
                System.Console.WriteLine(string.Format("AlarmText1: {0}", item.AlarmText1));
                System.Console.WriteLine(string.Format("AlarmText9: {0}", item.AlarmText9));
                System.Console.WriteLine(string.Format("Area: {0}", item.Area));
                System.Console.WriteLine(string.Format("BackColor: {0}", item.BackColor));
                System.Console.WriteLine(string.Format("ChangeReason: {0}",
item.ChangeReason));
                System.Console.WriteLine(string.Format("ClearTime: {0}", item.ClearTime));
                System.Console.WriteLine(string.Format("Connection: {0}", item.Connection));
                System.Console.WriteLine(string.Format("DeadBand: {0}", item.DeadBand));
                System.Console.WriteLine(string.Format("EventText: {0}", item.EventText));
                System.Console.WriteLine(string.Format("InfoText: {0}", item.InfoText));
                System.Console.WriteLine(string.Format("Flashing: {0}", item.Flashing));
                System.Console.WriteLine(string.Format("HostName: {0}", item.HostName));
                System.Console.WriteLine(string.Format("InvalidFlags: {0}",
item.InvalidFlags));
                System.Console.WriteLine(string.Format("LoopInAlarm: {0}",
item.LoopInAlarm));
                System.Console.WriteLine(string.Format("ModificationTime: {0}",
item.ModificationTime));
                System.Console.WriteLine(string.Format("Name: {0}", item.Name));
                System.Console.WriteLine(string.Format("Origin: {0}", item.Origin));
                System.Console.WriteLine(string.Format("Priority: {0}", item.Priority));
                System.Console.WriteLine(string.Format("RaiseTime: {0}", item.RaiseTime));
                System.Console.WriteLine(string.Format("ResetTime: {0}", item.ResetTime));
                System.Console.WriteLine(string.Format("SourceID: {0}", item.SourceID));
                System.Console.WriteLine(string.Format("SourceType: {0}", item.SourceType));
                System.Console.WriteLine(string.Format("State: {0}", item.State));
                System.Console.WriteLine(string.Format("StateText: {0}", item.StateText));
                System.Console.WriteLine(string.Format("SuppressionState: {0}",
item.SuppressionState));
                System.Console.WriteLine(string.Format("SystemSeverity: {0}",
item.SystemSeverity));
                System.Console.WriteLine(string.Format("TextColor: {0}", item.TextColor));
                System.Console.WriteLine(string.Format("User: {0}", item.UserName));
            }
        }
    }
}

```

8.5 Interfaces of the alarms

Copy code

```
        System.Console.WriteLine(string.Format("UserResponse: {0}",
item.UserResponse));
        System.Console.WriteLine(string.Format("Value: {0}", item.Value));
        System.Console.WriteLine(string.Format("ValueLimit: {0}",
item.ValueQuality));

        AlarmList.Add(item);
    }
    ...
}
...
}
```

See also

[IArm \(Page 76\)](#)

[IArmSubscription \(Page 94\)](#)

8.5.2 IAlarm

Description

The C# interface "IArm" specifies properties and methods for handling active alarms of the Runtime system.

The methods trigger an exception in the case of an error.

Members

The following properties and methods are specified in the interface:

"Error" property

Error code of the alarm

```
uint32 Error { get; set; }
```

"Name" property

Name of the active alarm

```
string Name { get; set; }
```

"Acknowledge" method

Acknowledge active alarm or instance of an active alarm synchronously.

```
void Acknowledge(UInt32 instanceId)
```

- `instanceID`
 - Value "0": Acknowledge all instances of an active alarm.
 - Value > "0": Acknowledge instance with this ID.

"Disable" method

Deactivate generation of the alarm in the alarm source synchronously.

```
void Disable()
```

"Enable" method

Activate generation of the alarm in the alarm source synchronously again.

```
void Enable()
```

"Reset" method

Acknowledge the outgoing state of an active alarm or an instance of an active alarm synchronously.

```
void Reset(UInt32 instanceId)
```

- `instanceID`
 - Value "0": Acknowledge the outgoing state of the active alarm.
 - Value > "0": Acknowledge the outgoing state of an instance with this ID.

"Shelve" method

Hide active alarm synchronously.

```
void Shelve()
```

"Unshelve" method

Display hidden alarm synchronously again.

```
void Unshelve()
```

Example

Acknowledge active alarm synchronously:

Copy code

```
public void AcknowledgeAlarms(IList<IAAlarmResult> AlarmList)
{
    if (AlarmList != null)
    {
        if (AlarmList.Count == 1)
        {
            IAAlarm alarmAck = runtime.GetObject<IAAlarm>(AlarmList[0].Name);
            alarmAck.Acknowledge(AlarmList[0].Name);
        }
        else
        {
            //process multiple alarms
        }
    }
}
```

See also

[IAAlarmResult \(Page 68\)](#)

[IAAlarmSet \(Page 78\)](#)

[IAAlarmSubscription \(Page 94\)](#)

8.5.3 IAAlarmSet

Description

The C# interface "IAAlarmSet" specifies properties, methods and events for optimized access to several active alarms of the Runtime system.

After initialization of the "IAAlarmSet" object, you can execute asynchronous operations with multiple alarms in one call, e. g. acknowledgment. Simultaneous access demonstrates better performance and lower communication load than single access to multiple alarms.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

The methods trigger an exception in the case of an error.

Members

The following properties, methods and events are only specified in the interface:

Access operator "this[string]"

Accessing an element of an alarm set via the alarm name.

```
IArm this[string alarmName] { get; set; }
```

alarmName

Name of the alarm that is changed in the AlarmSet.

"Count" property

Number of alarms of an AlarmSet list.

```
int32 Count { get; }
```

"Add" method

Add an active alarm or an instance of the alarm to the AlarmSet.

```
IArm Add(string alarmName, UInt32 instanceId)
```

- alarmName
Name of the alarm that is added to the AlarmSet.
- instanceId
Value = "0": Add all instances of an active alarm.
Value > "0": Add instance with this ID.

"Remove" method

Remove a single alarm or an instance of an alarm from the AlarmSet.

```
void Remove(string alarmName, UInt32 instanceId)
```

- alarmName
Name of the alarm that is removed from the AlarmSet.
- instanceID
value = "0": Remove all instances of an active alarm
Value > "0": Remove instance with this ID.

"Clear" method

Remove all alarms from an AlarmSet.

```
void Clear()
```

"Acknowledge" method

Acknowledge alarms of the AlarmSet asynchronously.

```
int Acknowledge()
```

"Disable" method

Deactivate generation of alarms of the AlarmSet in the alarm source asynchronously.

```
void Disable()
```

"Enable" method

Activate generation of alarms of the AlarmSet in the alarm source asynchronously again.

```
void Enable()
```

"Reset" method

Acknowledge outgoing state of the alarms of the AlarmSet asynchronously.

```
void Reset()
```

"Shelve" method

Hide alarms of the AlarmSet asynchronously.

```
void Shelve()
```

"Unshelve" method

Display hidden alarms of the AlarmSet again.

```
void Unshelve()
```

"OnAcknowledgeHandler" event

When an AlarmSet is acknowledged with the "Acknowledge" and "AcknowledgeInstance" methods, the event calls an instance of the "OnAlarmSetEventHandler" delegate.

Declares the event and the event handler for the asynchronous acknowledgment of the alarms.

```
event OnAlarmSetEventHandler OnAcknowledgeHandler
```

"OnResetHandler" event

Event calls an instance of the "OnAlarmSetEventHandler" delegate using the "Reset" method when acknowledging the outgoing state of the alarms of an AlarmSet.

Declares the event and the event handler for the asynchronous removal of the alarms.

```
event OnAlarmSetEventHandler OnResetHandler
```

"OnDisableHandler" event

When the alarms of an AlarmSet are deactivated with the "Disable" method, the event calls an instance of the "OnAlarmSetEventHandler" delegate.

Declares the event and the event handler for the asynchronous deactivation of the alarms.

```
event OnAlarmSetEventHandler OnDisableHandler
```

"OnEnableHandler" event

When the alarms of an AlarmSet are reactivated with the "Enable" method, the event calls an instance of the "OnAlarmSetEventHandler" delegate.

Declares the event and the event handler for the asynchronous reactivation of the alarms.

```
event OnAlarmSetEventHandler OnEnableHandler
```

"OnShelveHandler" event

When the alarms of an AlarmSet are hidden with the "Shelve" method, the event calls an instance of the "OnAlarmSetEventHandler" delegate.

Declares the event and the event handler for the asynchronous hiding of the alarms.

```
event OnAlarmSetEventHandler OnShelveHandler
```

"OnUnshelveHandler" event

When the alarms of an AlarmSet are displayed with the "Unshelve" method, the event calls an instance of the "OnAlarmSetEventHandler" delegate.

Declares the event and the event handler for the new asynchronous displaying of the alarms.

```
event OnAlarmSetEventHandler OnUnshelveHandler
```

"OnAlarmSetEventHandler" delegate

Specifies the signature of the event handling method for all events of an AlarmSet.

```
void OnAlarmSetEventHandler(  
    IAlarmSet sender,  
    uint32 errorCode,  
    IList<IAlarmSetResult> values,  
    bool completed)
```

- `sender`
Source of the event
- `errorCode`
Error during asynchronous transfer
- `values`
Event data as a list of "IAlarmSetResult" objects of the processed alarms.
- `completed`
Status of the asynchronous transfer:
 - True: All alarms of the AlarmSet are processed.
 - False: Not all alarms of the AlarmSet are processed yet.

Example

Acknowledge active alarms from the "AlarmList" list as an AlarmSet asynchronously:

Copy code

```

public void AcknowledgeAlarms(IList<IAAlarmResult> AlarmList)
{
    if (AlarmList != null)
    {
        if (AlarmList.Count == 1)
        {
            IAAlarm alarmAck = runtime.GetObject<IAAlarm>(AlarmList[0].Name);
            alarmAck.Acknowledge();
        }
        else
        {
            AlarmAckList = new List<IAAlarmResult>();
            IAAlarmSet alarmSet = runtime.GetObject<IAAlarmSet>(); ;
            foreach (var alarmResult in AlarmList)
            {
                IAAlarm pAlarm = null;
                pAlarm = alarmSet.Add(alarmResult.Name);
                AlarmAckList.Add(alarmResult);
            }

            // Assign callback function
            alarmSet.OnAcknowledgeHandler += alarmSet_OnAcknowledgeHandler;
            try
            {
                alarmSet.Acknowledge();
            }
            catch (OdkException ex)
            {
                System.Console.WriteLine(string.Format("OdkException occurred {0}",
ex.Message));
            }
        }
    }
}

public void alarmSet_OnAcknowledgeHandler(IAAlarmSet sender, uint errorCode,
IList<IAAlarmSetResult> values, bool completed)
{
    try
    {
        foreach (var item in values)
        {
            System.Console.WriteLine(string.Format("InstanceId: {0} Name: {1} SystemName:
{2} ",
                item.InstanceId, item.Name, item.SystemName));
        }
    }
    finally
    {
        if (null != sender)
        {
            sender.Dispose();
        }
    }
}

```

8.5 Interfaces of the alarms

Copy code

Remove active alarms from the "AlarmList" list as an AlarmSet asynchronously:

Copy code

```
public void ResetAlarms(IList<IAAlarmResult> AlarmList)
{
    if (AlarmList != null)
    {
        if (AlarmList.Count == 1)
        {
            IAAlarm alarmReset = runtime.GetObject<IAAlarm>(AlarmList[0].Name);
            alarmReset.Reset();
        }
        else
        {
            IAAlarmSet alarmSet = runtime.GetObject<IAAlarmSet>(); ;
            AlarmResetList = new List<IAAlarmResult>();
            foreach (var alarmResult in AlarmList)
            {
                IAAlarm pAlarm = null;
                pAlarm = alarmSet.Add(alarmResult.Name);
                AlarmResetList.Add(alarmResult);
            }
            alarmSet.OnResetHandler += alarmSet_OnReseteHandler;
            try
            {
                alarmSet.Reset();
            }
            catch (OdkException ex)
            {
                System.Console.WriteLine(string.Format("OdkException occurred {0}",
ex.Message));
            }
        }
    }
}

public void alarmSet_OnReseteHandler(IAAlarmSet sender, uint errorCode,
IList<IAAlarmSetResult> values, bool completed)
{
    try
    {
        foreach (var item in values)
        {
            System.Console.WriteLine(string.Format("InstanceId: {0} Name: {1} SystemName:
{2} ",
                item.InstanceId, item.Name, item.SystemName));
        }
    }
    finally
    {
        if (null != sender)
        {
            sender.Dispose();
        }
    }
}
```

See also

[IAlarmSetResult \(Page 86\)](#)

[IAlarm \(Page 76\)](#)

8.5.4 IAlarmSetResult

Description

The C# interface "IAlarmSetResult" specifies properties of alarms in AlarmSets. These properties are returned by the EventHandler for all events of AlarmSets.

Members

The following properties are specified in the interface:

"SystemName" property

System name of the Runtime system of an alarm in the AlarmSet

```
string SystemName { get; }
```

"Name" property

Name of an alarm in the AlarmSet

```
string Name { get; }
```

"Instanceld" property

InstanceId of an alarm in the AlarmSet

```
uint32 InstanceId { get; }
```

"Error" property

Error code of an alarm in the AlarmSet

```
uint32 Error { get; }
```

Example

Read out alarm of an AlarmSet after acknowledgment

Copy code

```
public void alarmSet_OnAcknowledgeHandler(IAlarmSet sender, uint errorCode,
    IList<IAlarmSetResult> values, bool completed)
{
    try
    {
        foreach (var item in values)
        {
            System.Console.WriteLine(string.Format("InstanceId: {0} Name: {1} SystemName:
{2} ",
                item.InstanceId, item.Name, item.SystemName));
        }
    }
    finally
    {
        if (null != sender)
        {
            sender.Dispose();
        }
    }
}
```

See also

[IAlarmSet \(Page 78\)](#)

8.5.5 IAlarmTrigger

Description

The C# interface "IAlarmTrigger" specifies methods for triggering alarms.

Members

"CreateSystemAlarm" method

Generates an alarm of the class SystemAlarmWithoutClearEvent with the state machine alarm without outgoing status with acknowledgment.

8.5 Interfaces of the alarms

```
void CreateSystemAlarm(object alarmText, string area, object  
alarmParameterValue1, ..., object alarmParameterValue7)
```

- **alarmText:**

The alarm text. You have the following options:

- Transferring a text list of the type "ITextList".
The list entries of the text list can directly contain multilingual alarm texts or references to other text lists with multilingual alarm texts.

Note

Only user-defined text lists

This method processes only user-defined text lists.

- Transfer static string with monolingual text.

- **area:**

The area of origin of the alarm

- **alarmParameterValue1 to alarmParameterValue7:**

User-defined comments

The alarm triggers an event with the following event path:

- For multilingual alarm

texts:

```
@ScriptingSystemEvents.SystemAlarmWithoutClearEvent:SystemAlarmWithoutClearEvent
```

- For monolingual alarm

text:

```
@ScriptingSystemEvents.SystemAlarmWithoutClearEventText:SystemAlarmWithoutClearEvent
```

"CreateSystemInformation" method

Generates an alarm of the class SystemInformation with the state machine alarm without outgoing status without acknowledgment.


```
void CreateSystemInformation(object alarmText, string area, object
alarmParameterValue1, ..., object alarmParameterValue7)
```

- **alarmText:**
The alarm text. You have the following options:
 - Transferring a text list of the type "ITextList".
The list entries of the text list can directly contain multilingual alarm texts or references to other text lists with multilingual alarm texts.

Note

Only user-defined text lists

This method processes only user-defined text lists.

- Transfer static string with monolingual text.

- **area:**
The area of origin of the alarm
- **alarmParameterValue1 to alarmParameterValue7:**
User-defined comments

The alarm triggers an event with the following event path:

- For multilingual alarm
text: @ScriptingSystemEvents.SystemInformation:SystemInformation
- For monolingual alarm
text: @ScriptingSystemEvents.SystemInformationText:SystemInformation

"CreateOperatorInputInformation" method

Generates an alarm of the class OperatorInputInformation with the state machine alarm without outgoing status without acknowledgment.

```
void CreateOperatorInputInformation(object alarmText, string area,
object alarmParameterValue1, ..., object alarmParameterValue7)
```

- **alarmText:**
The alarm text. You have the following options:
 - Transferring a text list of the type "ITextList".
The list entries of the text list can directly contain multilingual alarm texts or references to other text lists with multilingual alarm texts.

Note

Only user-defined text lists

This method processes only user-defined text lists.

- Transfer static string with monolingual text.

- **area:**
The area of origin of the alarm
- **alarmParameterValue1 to alarmParameterValue7:**
User-defined comments

8.5 Interfaces of the alarms

The alarm triggers an event with the following event path:

- For multilingual alarm

texts:

```
@ScriptingSystemEvents.OperatorInputInformationText:OperatorInputI  
nformation
```

- For monolingual alarm

text:

```
@ScriptingSystemEvents.OperatorInputInformationText:OperatorInputI  
nformation
```

Example

The following code examples show how to trigger alarms of the class `SystemAlarmWithoutClearEvent`. Alarms of the classes `SystemInformation` and `OperatorInputInformation` are triggered in the same way.

8.5 Interfaces of the alarms

Copying code

```

public void CreateSystemAlarm()
{
    //Create SystemAlarm with monolingual alarm text
    var alarm = runtime.GetObject<IAlarmSubscription>();
    var systemNames = new List<string>();
    systemNames.Add("SYSTEM1");
    alarm.OnAlarmHandler += alarm_OnAlarmHandler;
    alarm.OnPendingAlarmCompleteHandler += Alarm_OnPendingAlarmCompleteHandler;
    alarm.Filter = "AlarmClassName = 'SystemAlarmWithoutClearEvent'";
    alarm.Language = 1033;
    alarm.SystemNames = systemNames;
    alarm.Start();
    var systemAlarm = runtime.GetObject<IAlarmTrigger>();
    systemAlarm.CreateSystemAlarm(alarmText: "Alarm Text", area: "Area",
        alarmParameterValue1: "Param1",
        alarmParameterValue2: "Param2",
        alarmParameterValue3: "Param3",
        alarmParameterValue4: "Param4",
        alarmParameterValue5: "Param5",
        alarmParameterValue6: "Param6",
        alarmParameterValue7: "Param7");
    _event.WaitOne();
    _event.Reset();
    //stop alarm subscription
    alarm.Stop();
    //Dispose alarm object
    alarm.Dispose();
}

public void CreateSystemAlarmWithAlarmTextAsTextList()
{
    //Create SystemAlarm with multilingual alarm text; the translations are directly stored
    in the text list
    var alarm = runtime.GetObject<IAlarmSubscription>();
    var systemNames = new List<string>();
    systemNames.Add("SYSTEM1");
    alarm.OnAlarmHandler += alarm_OnAlarmHandler;
    alarm.OnPendingAlarmCompleteHandler += Alarm_OnPendingAlarmCompleteHandler;
    alarm.Filter = "AlarmClassName = 'SystemAlarmWithoutClearEvent'";
    alarm.Language = 1033;
    alarm.SystemNames = systemNames;
    alarm.Start();
    var systemAlarm = runtime.GetObject<IAlarmTrigger>();
    var texlistforAlarmText = runtime.GetObject<ITextList>();
    // Text list: AlarmTextTemplate
    // Test list entry index: 101
    // Text: "My input msg. input value = @1@d@"
    texlistforAlarmText.Name = "AlarmTextTemplate";
    texlistforAlarmText.TextListEntryIndex = 101;
    systemAlarm.CreateSystemAlarm(alarmText: texlistforAlarmText, area: "Area",
        alarmParameterValue1: "125", // dynamic value for format specifier @1@d@
        alarmParameterValue2: "Param2",
        alarmParameterValue3: "Param3",
        alarmParameterValue4: "Param4",

```

Copying code

```

        alarmParameterValue5: "Param5",
        alarmParameterValue6: "Param6",
        alarmParameterValue7: "Param7");
    _event.WaitOne();
    _event.Reset();
    //stop alarm subscription
    alarm.Stop();
    //Dispose alarm object
    alarm.Dispose();
}

public void CreateSystemAlarmWithTextListAsParameterValue()
{
    //Create SystemAlarm with multilingual alarm text; the text list references other text
    lists with translations
    var alarm = runtime.GetObject<IAlarmSubscription>();
    var systemNames = new List<string>();
    systemNames.Add("SYSTEM1");
    alarm.OnAlarmHandler += alarm_OnAlarmHandler;
    alarm.OnPendingAlarmCompleteHandler += Alarm_OnPendingAlarmCompleteHandler;
    alarm.Filter = "AlarmClassName = 'SystemAlarmWithoutClearEvent'";
    alarm.Language = 1033;
    alarm.SystemNames = systemNames;
    alarm.Start();
    var systemAlarm = runtime.GetObject<IAlarmTrigger>();
    var textList1 = runtime.GetObject<ITextList>("Text_List_1");
    var textList2 = runtime.GetObject<ITextList>("Text_List_2");
    textList1.TextListEntryIndex = 1; //Eng TL @1%t#2T@ Val: @3%s@
    systemAlarm.CreateSystemAlarm(alarmText: textList1, area: "Area",
    alarmParameterValue1: 1,
    alarmParameterValue2: textList2, // text list object
    alarmParameterValue3: "Hello"); // Dynamic value of @3%s@
    _event.WaitOne();
    _event.Reset();
    //stop alarm subscription
    alarm.Stop();
    //Dispose alarm object
    alarm.Dispose();
}

```

See also

[ITextList \(Page 94\)](#)

8.5.6 ITextList

Description

The C# interface "ITextList" is used to transfer multilingual alarm texts for system alarms and operator input alarms. See section IAlarmTrigger (Page 87), CreateSystemInformation method. An ITextList instance is passed to the alarm text. When the operator input alarm is generated, it is replaced by the configured text from the text list.

Members

"Name" property

The name of the text list.

```
string Name { get; set; }
```

"TextListEntryIndex" property

The index of the list entry

```
UInt32 TextListEntryIndex { get; set; }
```

8.5.7 IAlarmSubscription

Description

The C# interface "IAlarmSubscription" specifies methods for monitoring alarms of the Runtime system.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

Members

The following methods and events are specified in the interface:

"SystemNames" property

System names of Runtime systems for the monitoring of active alarms.

```
ICollection<string> SystemNames { get; set; }
```

"Language" property

Country identification of the language of the monitored alarms

```
uint32 Language { get; set; }
```

"Filter" property

SQL-type string for filtering the result set of active alarms

```
string Filter { get; set; }
```

All properties of an alarm can be used in the filter string. The filter string can contain operators. Refer to the section Syntax of the alarm filter (Page 19).

"Start" method

Subscribe systems for monitoring of changes of active alarms.

```
void Start()
```

"Stop" method

Unsubscribe monitoring of active alarms.

Note

Start and Stop in Windows Forms applications

Do not call the "Stop" method for a Windows forms application in the same thread in which you called "Start".

```
void Stop()
```

"OnAlarmHandler" event

Following a change of alarm state on subscribed systems, the event calls an instance of the "OnAlarmHandler" delegate.

Declares the event and the event handler for asynchronous monitoring of alarms.

```
event OnAlarmHandler OnAlarmHandler
```

"OnPendingAlarmCompleteHandler" event

After completion of handling of all active alarms of a system, the event calls an instance of the "OnPendingAlarmCompleteHandler" delegate.

Declares the event and the event handler for confirmation of the asynchronous operations.

```
event OnPendingAlarmCompleteHandler OnPendingAlarmCompleteHandler
```

"OnAlarmHandler" delegate

Specifies the signature of the event handling method for the "OnAlarmHandler" event.

```
void OnAlarmHandler(
    IAlarmSubscription sender,
    uint32 systemError,
    string systemName,
    IList<IAlarmResult> values,
    bool completed)
```

- sender
Reference to the "IAlarmSubscription" object
- systemError
Error code for the asynchronous operation
- systemName
Name of the Runtime system that is subscribed for alarm monitoring by the user.

8.5 Interfaces of the alarms

- `values`
Event data as a list of "IArmResult" objects of the monitored active alarms.
- `completed`
Status of the asynchronous transfer:
 - `True`: All alarms are read out.
 - `False`: Not all alarms are yet read out.

"OnPendingAlarmCompleteHandler" delegate

Specifies the signature of the event handling method for the "OnPendingAlarmCompleteHandler" event.

```
void OnPendingAlarmCompleteHandler(IArmSubscription sender)
```

- `sender`
Source of the event

Example

Define functions for asynchronous monitoring of active alarms. Monitor "OnAlarmHandler" event and when occurring call the event handling method "alarm_OnAlarmHandler":

8.5 Interfaces of the alarms

Copy code

```

public void SubscribeAlarmWithFilterOnOriginAndAlarmclass()
{
    try
    {
        // object filter = "AlarmClass = 'AlarmWithOptionalAcknowledgement' AND Origin =
        'Boiler'";
        IAlarmSubscription alarm = runtime.GetObject<IAlarmSubscription>();
        List<string> systemNames = new List<string>();
        systemNames.Add("SYSTEM1");

        // Assign alarm handler
        alarm.OnAlarmHandler += alarm_OnAlarmHandler;
        alarm.Filter = "AlarmClassName = 'AlarmWithOptionalAcknowledgement' AND Origin =
        'Boiler'";
        alarm.Language = 1033;
        alarm.SystemNames = systemNames;
        alarm.Start();
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occured {0}", ex.Message));
    }
}

public void alarm_OnAlarmHandler(IAlarmSubscription sender, UInt32 nGlobalError, String
systemName, IList<IAlarmResult> value, bool completed)
{
    if (0 == nGlobalError)
    {
        try
        {
            AlarmList = new List<IAlarmResult>();
            foreach (var item in value)
            {
                System.Console.WriteLine(string.Format("InstanceID: {0} SourceID: {1} Name:
                {2} State: {3} EventText: {4} StateText: {5} BackColor: {6} Flashing: {7} Quality: {8}
                ModificationTime: {9}",
                    item.InstanceID, item.SourceID, item.Name, item.State,
                    item.EventText, item.StateText, item.BackColor, item.Flashing,
                    item.Quality, item.ModificationTime));
                AlarmList.Add(item);
            }

            // For test purpose: Cancel subscription after first notification
            AcknowledgeAlarms(AlarmList);
        }
        finally
        {
            if (null != sender)
            {
                sender.Stop();
                sender.Dispose();
            }
        }
    }
}

```

Copy code

```
else
{
    System.Console.WriteLine("AlarmSubscription Failed");
}
```

See also

[IAlarmResult](#) (Page 68)

[IAlarm](#) (Page 76)

8.5.8 ILoggedAlarmResult

Description

The C# interface "ILoggedAlarmResult" specifies methods for accessing properties of logged alarms of a logging system.

Members

The following properties are specified in the interface:

"InstanceID" property

InstanceID for a logged alarm with multiple instances

```
uint32 InstanceID { get; }
```

"Name" property

Name of the logged alarm.

```
string Name { get; }
```

"AlarmClassName" property

Name of the alarm class of the logged alarm.

```
string AlarmClassName { get; }
```

"AlarmClassSymbol" property

Symbol of the alarm class of the logged alarm.

```
string AlarmClassSymbol { get; }
```

"AlarmParameterValues" property

Parameter values of the logged alarm.

```
ReadOnlyList<object> AlarmParameterValues { get; }
```

"AlarmText1" ... "AlarmText9" properties

Additional texts 1-9 of the logged alarm.

```
string AlarmText1 { get; }
```

...

```
string AlarmText9 { get; }
```

"ChangeReason" property

Trigger event of the change of the alarm state of the logged alarm.

```
uint16 ChangeReason { get; }
```

"Connection" property

Connection via which the logged alarm was triggered.

```
string Connection { get; }
```

"State" property

Alarm state of the logged alarm.

```
HmiAlarmState State { get; }
```

The enumeration "HmiAlarmState" can contain the following values:

- Standard (0x00)
- Raised (0x01)
- RaisedCleared (0x02)
- RaisedAcknowledged (0x05)
- RaisedAcknowledgedCleared (0x06)
- RaisedClearedAcknowledged (0x07)
- Removed (0x80)

"StateText" property

Alarm state of the logged alarm as text, e.g. "incoming" or "outgoing".

```
string StateText { get; }
```

"EventText" property

Text of the logged alarm that describes the alarm event.

```
string EventText { get; }
```

"TextColor" property

Text color of the alarm state of the logged alarm.

```
uint32 TextColor { get; }
```

"BackColor" property

Background color of the alarm state of the logged alarm.

```
uint32 BackColor { get; }
```

"Flashing" property

Indicates whether the logged alarm flashes.

```
bool Flashing { get; }
```

"InvalidFlags" property

Marking of the logged alarm in case of invalid data

```
byte InvalidFlags { get; }
```

"ModificationTime" property

Time of the last change of the alarm state of the logged alarm.

```
DateTime ModificationTime { get; }
```

"RaiseTime" property

Trigger time of the logged alarm.

```
DateTime RaiseTime { get; }
```

"AcknowledgementTime" property

Time at which the logged alarm was acknowledged.

```
DateTime AcknowledgementTime { get; }
```

"ClearTime" property

Time at which the logged alarm was cleared.

```
DateTime ClearTime { get; }
```

"ResetTime" property

Time at which the logged alarm was reset.

```
DateTime ResetTime { get; }
```

"SuppressionState" property

Status of the visibility of the logged alarm.

```
byte SuppressionState { get; }
```

"SystemSeverity" property

Severity of the system error

```
byte SystemSeverity { get; }
```

"Priority" property

Relevance for display and sorting of the logged alarm.

```
byte Priority { get; }
```

"Origin" property

Origin for display and sorting of the logged alarm.

```
string Origin { get; }
```

"Area" property

Origin area for display and sorting of the logged alarm.

```
string Area { get; }
```

"Value" property

Process value of the logged alarm.

```
object Value { get; }
```

"AlarmGroupName" property

Name of the group to which the alarm belongs. Blank if the alarm does not belong to a group.

```
string AlarmGroupName { get; }
```

"DeadBand" property

Range of the triggering tag in which no alarms are generated.

```
object DeadBand { get; }
```

"HostName" property

Name of the host that triggered the alarm.

```
string HostName { get; }
```

"InfoText" property

Localizable text for the alarm that contains the associated work instruction.

```
string InfoText { get; }
```

"StateMachine" property

StateMachine model of the alarm. The StateMachine represents the behavior of alarms through arrangement of alarm states and alarm events, e.g. "RaiseClear", "RaiseRequiresAcknowledgment" or "RaiseClearOptionalAcknowledgment".

```
byte StateMachine { get; }
```

"ValueQuality" property

Quality of the process value of the alarm

```
int32 ValueQuality { get; }
```

"ValueLimit" property

Limit of the process value of the logged alarm.

```
object ValueLimit { get; }
```

"SingleAcknowledgement" property

Indicates whether an alarm may be acknowledged only individually or may be acknowledged as a group or multiple selection.

```
bool SingleAcknowledgement { get; }
```

"LoggedAlarmStateObjectID" property

ID of alarm state for referencing within the logging system.

```
string LoggedAlarmStateObjectID { get; }
```

"ID" property

User-defined ID of the alarm that is also used in the display.

```
uint32 ID { get; }
```

"SourceType" property

Source from which the alarm was generated, e.g. tag-based, controller-based or system-based alarm.

```
HmiAlarmSourceType SourceType { get; }
```

The enumeration "HmiAlarmSourceType" can contain the following values:

- Undefined (0)
- Tag (1)
- Controller (2)
- System (3)
- Alarm (4)

"UserName" property

User name of the logged operator input alarm.

```
string UserName { get; }
```

"UserResponse" property

Expected or required user response to the logged alarm.

```
byte UserResponse { get; }
```

"LoopInAlarm" property

Function that navigates from the alarm view to its origin.

```
string LoopInAlarm { get; }
```

"Error" property

Error code of the logged alarm.

```
uint32 Error { get; }
```

See also

[IAlarmLogging](#) (Page 104)

[IAlarmLoggingSubscription](#) (Page 106)

8.5.9 IAlarmLogging

Description

The C# interface "IAlarmLogging" specifies properties and methods for reading out logged alarms of a logging system.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

Members

The following methods and events are specified in the interface:

"Read" method

Read out logged alarms of a time period synchronously from logging system.

```
IList<ILoggedAlarmResult> Read(  
    DateTime begin,  
    DateTime end,  
    string filter,  
    uint32 language,  
    IList<string> systemNames)
```

- `begin`
Start date of the time period
- `end`
End date of the time period
- `filter`
Filter for limiting the read operation with properties of the "ILoggedAlarmResult" object.
- `language`
Country identification of the language of the logged alarm text
- `systemNames`
System names of the Runtime systems of the logged alarms. Default: local system

"ReadAsync" method

Read out logged alarms of a time period asynchronously from logging system.

```
void ReadAsync(  
    DateTime begin,  
    DateTime end,  
    string filter,  
    uint32 language,  
    IList<string> systemNames)
```

- `begin`
Start date of the time period
- `end`
End date of the time period

- `filter`
Filter for limiting the read operation with properties of the "ILoggedAlarmResult" object.
- `language`
Country identification of the language of the logged alarm text
- `systemNames`
System names of the Runtime systems of the logged alarms. Default: local system

"OnReadComplete" event

After readout of the logged alarms, event calls an instance of the "OnReadCompleteAlarmLoggingHandler" delegate.

Declares the event and the event handler for the asynchronous readout of logged alarms.

```
event OnReadCompleteAlarmLoggingHandler OnReadComplete
```

"OnReadCompleteAlarmLoggingHandler" delegate

Specifies the signature of the event handling method for the "OnReadComplete" event.

```
void OnReadCompleteAlarmLoggingHandler(
    IAlarmLogging sender,
    uint32 errorCode,
    IList<ILoggedAlarmResult> values,
    bool completed)
```

- `sender`
Source of the event
- `errorCode`
Error code for the asynchronous operation
- `values`
Event data as a list of "ILoggedAlarmResult" objects of the read-out logged alarms.
- `completed`
Status of the asynchronous transfer:
 - True: All alarms are read out.
 - False: Not all alarms are yet read out.

Example

Reading out and outputting logged alarms asynchronously:

Copy code

```
public void ReadAsync()
{
    try
    {
        var alarm = _runtime.GetObject<IArmLogging>();
        var now = DateTime.UtcNow;
        var begin = now.AddMinutes(-5);
        var end = now.AddMinutes(-2);
        List<string> systemNames = new List<string>();
        systemNames.Add("SYSTEM1");
        alarm.OnReadComplete += Alarm_OnReadComplete;
        alarm.ReadAsync(begin, end, "", 1033, systemNames);
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}

private void Alarm_OnReadComplete(IArmLogging sender, uint errorCode,
    IList<ILoggedAlarmResult> values, bool completed)
{
    PrintValues(values);
    sender.Dispose();
}

private void PrintValues(IList<ILoggedAlarmResult> values)
{
    foreach (var v in values)
    {
        Console.WriteLine(string.Format("Name: {0} Timestamp: {1} Value: {2}", v.Name,
            v.RaiseTime, v.Value));
    }
}
```

See also

[ILoggedAlarmResult \(Page 99\)](#)

[IArmLoggingSubscription \(Page 106\)](#)

8.5.10 IArmLoggingSubscription

Description

The C# interface "IArmLoggingSubscription" specifies properties and methods for monitoring logged alarms of a logging system.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

Members

The following properties, methods and events are specified in the interface:

"SystemNames" property

System names of the Runtime systems of the logged alarms.

```
ICollection<string> SystemNames { get; set; }
```

"Language" property

Country identification of the language of the logged alarms.

```
uint32 Language { get; set; }
```

"Filter" property

SQL-like string for filtering the result set of the logged alarms.

```
string Filter { get; set; }
```

"Start" method

Start monitoring of logged alarms.

```
void Start()
```

"Stop" method

Stop monitoring of all logged alarms.

```
void Stop()
```

"OnDataChanged" event

Following a change of a monitored alarm in the logging systems, the event calls an instance of the "OnDataChangedAlarmLoggingHandler" delegate.

Declares the event and the event handler for the monitoring of logged alarms.

```
event OnDataChangedAlarmLoggingHandler OnDataChanged
```

"OnDataChangedAlarmLoggingHandler" delegate

Specifies the signature of the event handling method for the "OnDataChanged" event.

```
void OnDataChangedAlarmLoggingHandler(  
    IAlarmLoggingSubscription sender,
```

8.5 Interfaces of the alarms

```
uint32 errorCode,  
IList<ILoggedAlarmResult> values)
```

- sender
Source of the event
- errorCode
Error code for the asynchronous operation
- values
Event data as a list of "ILoggedAlarmResult" objects of the monitored logged alarms.

Example

Output logged alarms following a change.

Copy code

```
public void SubscribeAlarm()  
{  
    try  
    {  
        _alarm = _runtime.GetObject<IAlarmLoggingSubscription>();  
        List<string> systemNames = new List<string>();  
        systemNames.Add("SYSTEM1");  
        // Assign alarm handler  
        _alarm.OnDataChanged += Alarm_OnDataChanged;  
        _alarm.Filter = "";  
        _alarm.Language = 1033;  
        _alarm.SystemNames = systemNames;  
        _alarm.Start();  
    }  
    catch (OdkException ex)  
    {  
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));  
    }  
}  
  
private void Alarm_OnDataChanged(IAlarmLoggingSubscription sender, uint errorCode,  
IList<ILoggedAlarmResult> values)  
{  
    PrintValues(values);  
}  
  
private void PrintValues(IList<ILoggedAlarmResult> values)  
{  
    foreach (var v in values)  
    {  
        Console.WriteLine(string.Format("Name: {0} Timestamp: {1} Value: {2}", v.Name,  
v.RaiseTime, v.Value));  
    }  
}
```

Cancel monitoring of logged alarms:

Copy code

```
public void CancelSubscription()
{
    try
    {
        if (_alarm != null)
        {
            _alarm.Stop();
            _alarm.Dispose();
        }
    }
    catch (Exception ex)
    {
        System.Console.WriteLine(ex.Message);
    }
}
```

See also

[IAlarmLogging \(Page 104\)](#)

[ILoggedAlarmResult \(Page 99\)](#)

8.6 Interfaces for connections

8.6.1 IConnectionResult

Description

The C# interface "IConnectionResult" provides access to the details of a connection.

Members

The following properties are specified in the interface:

"Name" property

Name of the connection

```
string Name { get; }
```

"ConnectionState" property

Status of the connection

```
HmiConnectionState ConnectionState { get; }
```

The "HmiConnectionState" enumeration contains the following values:

- Disabled (0)
- Connecting (1)
- Connected (2)
- Disconnecting (3)
- Disconnected (4)
- Reconnecting (5)

"EstablishmentMode" property

Mode in which the connection will be established.

```
HmiConnectionEstablishmentMode EstablishmentMode { get; }
```

The "HmiConnectionEstablishmentMode" enumeration contains the following values:

- None (0)
- AutomaticActive (1)
- AutomaticPassive (2)
- OnDemandActive (3)
- OnDemandPassive (4)

"TimeSynchronizationMode" property

Mode of time synchronization between HMI system and automation system

```
HmiTimeSynchronizationMode TimeSynchronizationMode { get; }
```

The "HmiTimeSynchronizationMode" enumeration contains the following values:

- None (0)
- Slave (1)
- Master (2)

"DisabledAtStartup" property

Indicates whether the connection is disabled at the start of Runtime.

```
bool DisabledAtStartup { get; }
```

- true: Connection is disabled at the connection start.
- false: Connection is active at the connection start.

"Enabled" property

Indicates whether the connection is active.

```
bool Enabled { get; }
```

- true: Connection is active.
- false: Connection is not active.

"ConnectionType" property

Protocol of a communication driver, e.g. "S7 Classic".

```
string ConnectionType { get; }
```

"Error" property

Error code of the connection

```
uint32 Error { get; }
```

Example

Output connection details:

Copy code

```
public void Connection_Read()
{
    var (connection = m_runtime.GetObject<IConnection>("HMI-ConnectionS7Plus");
    if (connection != null)
    {
        var con = connection.Read();
        if (con != null)
        {
            Console.WriteLine("Connection Name is {0} ", con.Name);
            Console.WriteLine("ConnectionState is {0}", con.ConnectionState);
            Console.WriteLine("TimeSynchronizationMode is {0} ", con.TimeSynchronizationMod
e);
            Console.WriteLine("EstablishmentMode is {0} ", con.EstablishmentMode);
            Console.WriteLine("Enabled is {0} ", con.Enabled);
            Console.WriteLine("DisabledAtStartup is {0} ", con.DisabledAtStartup);
            Console.WriteLine("ConnectionType is {0} ", con.ConnectionType);
            Console.WriteLine(" Error is {0} ", con.Error);
        }
    }
}
```

See also

[IConnection \(Page 113\)](#)

[IConnectionSet \(Page 114\)](#)

8.6.2 IConnectionStatusResult**Description**

The C# interface "IConnectionStatusResult" provides access to the status of a connection.

Members

The following properties are specified in the interface:

"Name" property

Name of the connection

```
string Name { get; }
```

"ConnectionState" property

Status of the connection

```
HmiConnectionState ConnectionState { get; }
```

The "HmiConnectionState" enumeration contains the following values:

- Disabled (0)
- Connecting (1)
- Connected (2)
- Disconnecting (3)
- Disconnected (4)
- Reconnecting (5)

"Error" property

Error code of the connection

```
uint32 Error { get; }
```

Example

Output status of a certain connection:

Copy code

```
public void Connection_GetConnectionState()
{
    var connection = m_runtime.GetObject<IConnection>("HMI-ConnectionS7Plus");
    var con = connection.GetConnectionState();
    if (con != null)
    {
        Console.WriteLine("Connection Name is : {0} ", con.Name);
        Console.WriteLine(" Error is : {0} ", con.Error);
        Console.WriteLine("Connection status is: {0}", con.ConnectionStatus);
    }
}
```

See also

[IConnection \(Page 113\)](#)

[IConnectionSet \(Page 114\)](#)

8.6.3 IConnection

Description

The C# interface "IConnection" provides properties and methods for the access to a connection. A connection is a configured, logical assignment of two communication partners.

The interface inherits the `Dispose` method of the "IDisposable" interface of the .NET framework.

Members

The following properties and methods are specified in the interface:

"Name" property

Name of the connection

```
string Name { get; set; }
```

"Read" method

Read connection details synchronously from the Runtime system.

```
IConnectionResult Read()
```

"GetConnectionState" method

Return connection status of a connection.

```
IConnectionStatusResult GetConnectionState()
```

"SetConnectionMode" method

Change connection status of a connection.

```
void SetConnectionMode(ConnectionMode mode)
```

The "ConnectionMode" enumeration contains the following values:

- Disabled (0)
- Enabled (1)

Examples

Disable connection:

Copy code

```
public void Connection_SetConnectionStateDisable()
{
    var connection = m_runtime.GetObject<IConnection>("HMI-ConnectionS7Plus");
    if (connection != null)
    {
        connection.SetConnectionMode(ConnectionMode.Disabled);
    }
}
```

Enable connection:

Copy code

```
public void SetConnectionStateEnable()
{
    using (IConnection connection = runtime.GetObject<IConnection>("HMI-ConnectionS7Plus"))
    {
        if (connection != null)
        {
            connection.SetConnectionMode(ConnectionMode.Enabled);
        }
    }
}
```

See also

[IConnectionSet \(Page 114\)](#)

[IConnectionResult \(Page 109\)](#)

[IConnectionStatusResult \(Page 111\)](#)

8.6.4 IConnectionSet

Description

The C# interface "IConnectionSet" specifies properties, methods and events for optimized access to several connections of the Runtime system.

After initialization of the "IConnectionSet" object, you have read/write access to multiple connections in one call. Simultaneous access takes place with better performance and lower communication load than single access to multiple connections.

The interface inherits the `Dispose` method of the "IDisposable" interface of the .NET framework.

Members

The following properties, methods and events are specified in the interface:

"ContextId" property

ID as additional identification feature of a connection. The ContextId can, for example, be used to recognize identically named connections.

Default value -1: The ContextId is not used.

```
int32 ContextId { get; set; }
```

Access operator "this[string]"

Accessing or changing an element of a ConnectionSet via the connection name.

```
IConnection this[string connectionName] { get; set; }
```

connectionName

Name of the connection

"Count" property

Number of connections of a connection set list

```
int32 Count { get; }
```

"Add" method

Add connections to a connection set.

```
void Add(ICollection<string> connections)
```

connections

List of connections for the connection set

"Remove" method

Remove individual connection from connection set.

```
void Remove(string connection)
```

connection

Name of connection that is removed from the connection set.

"Clear" method

Remove all connections from connection set.

```
void Clear()
```

"Read" method

Read connection details of all connections of the connection set synchronously from the Runtime system.

```
IList<IConnectionResult> Read()
```

"ReadAsync" method

Read connection details of all connections of the connection set asynchronously from the Runtime system.

```
void ReadAsync()
```

"GetConnectionState" method

Read connection status of all connections of the connection set synchronously from the Runtime system.

```
IList<IConnectionStatusResult> GetConnectionState()
```

"Subscribe" method

Subscribe all connections of a connection set asynchronously for change monitoring.

```
void Subscribe()
```

"CancelSubscribe" method

Cancel change monitoring of all connections of a connection set.

```
void CancelSubscribe()
```

"OnConnectionRead" event

After establishment of the connection, event calls an instance of the "OnConnectionHandler" delegate.

Declares the event and the event handler for the establishment of a connection.

```
event OnConnectionHandler OnConnectionRead
```

"OnConnectionStateChange" event

After change of the connection status, event calls an instance of the "OnConnectionStateChangeHandler" delegate.

Declares the event and the event handler for the change of the connection status.

```
event OnConnectionStateChangeHandler OnConnectionStateChange
```

"OnConnectionHandler" delegate

Specifies the signature of the event handling method for the "OnConnectionHandler" event of a connection set.

```
void OnConnectionHandler(  
    IConnectionSet sender,  
    uint32 systemError,  
    IList<IConnectionResult> values)
```

- sender
Source of the event
- systemError
Error code
- values
List of connections

"OnConnectionStateChangeHandler" delegate

Specifies the signature of the event handling method for the "OnConnectionStateChangeHandler" event of a connection set.

```
void OnConnectionStateChangeHandler(  
    IConnectionSet sender,  
    uint32 systemError,  
    IList<IConnectionStatusResult> values)
```

- sender
Source of the event
- systemError
Error code
- values
List of status changes of the connections

Examples

Monitor connection status:

Copy code

```
public void ConnnectionSet_Subscribe()
{
    Console.WriteLine(" .....Connection Set:Subscription start .....");
    using (IConnectionSet subscribe = runtime.GetObject<IConnectionSet>())
    {
        if (subscribe != null)
        {
            ICollection<string> list = new string[] { "HMI-Connection", "HMI-
ConnectionS7Plus" };
            subscribe.Add(list);
            subscribe.OnConnectionStateChanged += Subscribe_OnConnectionStateChanged;
            subscribe.Subscribe();
            Thread.Sleep(500);
        }
    }
}

private void Subscribe_OnConnectionStateChanged(IConnectionSet sender, uint systemError,
IList<IConnectionStatusResult> values)
{
    if (0 == systemError)
    {
        try
        {
            foreach (var item in values)
            {
                Console.WriteLine("Name:{0} ", item.Name);
                Console.WriteLine("State:{0} ", item.ConnectionStatus.ToString());
                Console.WriteLine("Error:{0} ", item.Error);
            }
        }

        catch (OdkException ex)
        {
            System.Console.WriteLine(string.Format("OdkException occured {0}", ex.Message));
        }

        finally
        {
            if (null != sender)
            {
                sender.CancelSubscribe();
                Console.WriteLine("Subscription cancelled");
                sender.Dispose();
            }
        }
    }
    else
    {
        System.Console.WriteLine("Connection set subscription Failed");
    }
}
```

Copy code

Read out connection synchronously:

Copy code

```
public void ConnectionSet_Read()
{
    Console.WriteLine(" .....Connection Set: Read ..... ");
    using (IConnectionSet read = runtime.GetObject<IConnectionSet>())
    {
        if (read != null)
        {
            ICollection<string> list = new string[] { "HMI-Connection", "HMI-
ConnectionS7Plus" };
            read.Add(list);

            IList<IConnectionResult> connectionResult = read.Read();
            foreach (var item in connectionResult)
            {
                System.Console.WriteLine(string.Format("Connection Name is {0} ",
item.Name));
                System.Console.WriteLine(string.Format("ConnectionState is {0}",
item.ConnectionState.ToString()));
                System.Console.WriteLine(string.Format("TimeSynchronizationMode is {0} ",
item.TimeSynchronizationMode.ToString()));
                System.Console.WriteLine(string.Format(" Error is {0} ", item.Error));
                System.Console.WriteLine(string.Format("EstablishMentMode is {0} ",
item.EstablishmentMode));
                System.Console.WriteLine(string.Format("Enabled is {0} ", item.Enabled));
                System.Console.WriteLine(string.Format("DisabledAtStartup is {0} ",
item.DisabledAtStartup));
                System.Console.WriteLine(string.Format("ConnectionType is {0} ",
item.ConnectionType));
            }
        }
    }
}
```

Read out connection asynchronously:

Copy code

```
public void ConnectionSet_ReadAsync()
{
    Console.WriteLine(" .....Connection Set: ReadAsync start .....");
    IConnectionSet readAsync = runtime.GetObject<IConnectionSet>();
    if (readAsync != null)
    {
        ICollection<string> list = new string[] { "HMI-Connection", "HMI-ConnectionS7Plus" };
        readAsync.Add(list);
        readAsync.OnConnectionRead += Read_OnConnectionComplete;
        readAsync.ReadAsync();
        Thread.Sleep(5000);
    }
}

private void Read_OnConnectionComplete(IConnectionSet sender, uint systemError,
IList<IConnectionResult> values)
{
    foreach (var item in values)
    {
        Console.WriteLine("Name:{0} ", item.Name);
        Console.WriteLine("State:{0} ", item.ConnectionState);
        Console.WriteLine("establishmentMode:{0} ", item.EstablishmentMode);
        Console.WriteLine("TimeSynchronizationMode:{0} ", item.TimeSynchronizationMode);
        Console.WriteLine("ConnectionType:{0} ", item.ConnectionType);
        Console.WriteLine("Enabled:{0} ", item.Enabled);
        Console.WriteLine("DisabledAtStartup:{0} ", item.DisabledAtStartup);
        Console.WriteLine("Error:{0} ", item.Error);
    }
}

finally
{
    if (null != sender)
    {
        sender.Dispose();
    }
}
}
```

See also

[IConnection \(Page 113\)](#)

[IConnectionResult \(Page 109\)](#)

[IConnectionStatusResult \(Page 111\)](#)

8.7 Interfaces of the Plant Model

8.7.1 IPlantModel

Description

The C# interface "IPlantModel" specifies methods for access to object instances of the plant model of a Runtime system. The "IPlantModel" object represents the plant model of the graphical Runtime system.

The interface inherits the `Dispose` method of the "IDisposable" interface of the .NET framework.

Formatting a path in the hierarchy

A hierarchy path of object instances consists of several components and has the following syntax:

```
[SystemName].HierarchyName::[PlantObjectID/.../]PlantObjectID
```

The system name can be omitted for referencing a local hierarchy. The dot before the hierarchy name must stay.

Note

Fixed code `HierarchyName`

In the current version, `HierarchyName` has a fixed code which is "hierarchy".

Members

The class implements the following methods:

"GetPlantObject" method

Supplies an object instance of "IPlantObject".

```
IPlantObject GetPlantObject(string plantObject)
```

- `plantObject`
Identifies an `IPlantObject` instance by its name or its path in the hierarchy.

"GetPlantObjectsByType" method

Supplies a list with instances of "IPlantObject" that have a specific type.

```
IList<IPlantObject> GetPlantObjectsByType(string  
plantObjectTypeFilter, string viewFilter = null)
```

- `plantObjectTypeFilter`
Filter for the "IPlantObject" type on which the instances are based.
- Optional: `viewFilter`
Filter for the path in the hierarchy. Only instances from a specific node are returned.

"GetPlantObjectsByExpression" method

Supplies a list with instances of "IPlantObject" instances. The instances are filtered by type and property values.

```
IList<IPlantObject> GetPlantObjectsByExpression(ICollection<string>
propertyNames, string plantObjectTypeFilter, string
expressionFilter, string viewFilter)
```

- `propertyNames`
A list with property names
- `plantObjectTypeFilter`
Filter for the object type on which the instances are based.
- `expressionFilter`
An expression that is a filter for the property values.
- Optional: `viewFilter`
Filter for a hierarchy path.

Example:

```
var plantObjectArr =
PlantModel.GetPlantObjectsByExpression("Temperature", "Motor",
"Temperature>100");
```

"GetPlantObjectsByPropertyNames" method

Supplies a list with "IPlantObject" instances that have specific properties and originate in a specific plant node.

```
IList<IPlantObject>
GetPlantObjectsByPropertyNames(ICollection<string> propertyNames,
string viewFilter = null)
```

- `propertyNames`
A list with property names
If the list contains multiple values, all properties must be available at the object.
- Optional: `viewFilter`
Filter for a hierarchy path.

Example

Example of a hierarchy path:

Hierarchy path	Referenced object instance
<code>System2.TechnologicalHierarchy::P1/S1/L2/LeftPump</code>	References the "LeftPump" object instance in the "TechnologicalHierarchy" of system2.
<code>.TechnologicalHierarchy::P1/S1/L2/LeftPump</code>	References the "LeftPump" object instance in the "TechnologicalHierarchy" of the local system.
<code>U4711</code>	References the "U4711" object instance of the local system.
<code>System2::U4711</code>	References the "U4711" object instance of System2.

Sample code

Copy code

```
public void Odk_GetPlantObjectsByType()
{
    using (IPlantModel myPlantModel = runtime.GetObject<IPlantModel>())
    {
        //gets node for specified Node path
        IList<IPlantObject> plantObject =
myPlantModel.GetPlantObjectsByType("RUNTIME_1::NodeType1",
    ".hierarchy::RootNodeName\\Node1");

        if (plantObject.Count() > 0)
        {
            foreach (IPlantObject item in plantObject)
            {
                System.Console.WriteLine(string.Format("ViewName: {0} Name: {1}",
item.CurrentPlantView, item.Name));
            }
        }
    }
}
```

8.7.2 IPlantObject

Description

The C# interface "IPlantObject" specifies properties and methods for handling object instances of the plant model of a Runtime system.

An object instance in the plant model is based on an object type and its data structure. Each object instance receives its position within the hierarchy by assigning it to a hierarchy node.

The interface inherits the `Dispose` method of the "IDisposable" interface of the .NET framework.

Formatting of a hierarchy path

A hierarchy path of object instances consists of several components and has the following syntax:

```
[SystemName].HierarchyName::[PlantObjectID/.../]PlantObjectID
```

The system name can be omitted for referencing a local hierarchy. The dot before the hierarchy name must stay.

Members

The following properties, methods and events are specified in the interface:

"Name" property

The name for unique identification of the object instance.

```
string Name { get; }
```

"Parent" property

The parent object instance in the hierarchy.

```
IPlantObject Parent { get; }
```

"Children" property

List of child object instances in the hierarchy

```
ReadOnlyList<IPlantObject> Children { get; }
```

"PlantViewPaths" property

The dictionary with string/string pairs that maps the hierarchy names to the hierarchy paths for all hierarchies that contain the "IPlantObject" instance (hierarchy name to hierarchy path).

```
ReadOnlyDictionary<string, string> PlantViewPaths { get; }
```

"CurrentPlantView" property

Path and name of the object instance in the currently selected hierarchy, e.g. "Maintenance".

The "CurrentPlantView" property is used as basis for navigation with the "Parent" or "Children" properties. If the object instance is only contained in one hierarchy, "CurrentPlantView" contains its path. If the object is contained in several views, the hierarchy path must be set via this property before the "Parent" or "Children" property can be used.

```
string CurrentPlantView { get; set; }
```

"GetProperty" method

Supplies a property of the object instance.

```
IPlantObjectProperty GetProperty(string propertyName)
```

propertyName

Name of an object instance property

"GetProperties" method

Supplies a two-dimensional list (name-object pairs) of the data structure of the object instance. The list allows access to the instance properties.

```
IPlantObjectPropertySet GetProperties(ICollection<string>  
propertyName = null)
```

- Optional: propertyName
List with names of one or multiple object instance properties.
Without parameters, all properties of an object instance are returned

"GetActiveAlarms" method

Supplies all active alarms that the object instance contains at the time it is called in the active hierarchy. Unlike with an AlarmSubscription, no status changes or new alarms are signaled that occur after the function call. Users can filter the alarms or specify a SystemName if they only want to receive the active alarms of a specific system.

```
void GetActiveAlarms(UInt32 languageId, bool includeChildren =  
false, string filter = null);
```

- `languageID`
Language code of the language for all alarm texts and the filters. See chapter Locale IDs of the supported languages (Page 21).
- Optional: `includeChildren`
The active alarms of the child instances are returned as well.
- Optional: `filter`
SQL-type string for filtering the alarm texts. The filter can contain operators. See also Syntax of the alarm filter (Page 19).

"CreateAlarmSubscription" method

Supplies a "PlantObjectAlarmSubscription" that can be used to start and stop an alarm subscription.

```
IPlantObjectAlarmSubscription CreateAlarmSubscription();
```

"PlantObjectAlarmHandler" event

The event calls an instance of the "OnPlantObjectAlarmHandler" delegate.

```
event OnPlantObjectAlarmHandler PlantObjectAlarmHandler;
```

"OnPlantObjectAlarmHandler" delegate

Specifies the signature of the event handling method for the "PlantObjectAlarmHandler" event of an "IPlantObject" instance.

```
public delegate void OnPlantObjectAlarmHandler(  
    IPlantObject sender,  
    UInt32 systemError,  
    string systemName,  
    IList<IAAlarmResult> values,  
    bool completed);
```

- `sender`
Source of the event
- `systemError`
Supplies an error code when a global error has occurred. When the error code is set, `values` is irrelevant.
- `systemName`
Name of the Runtime system that is subscribed for alarm monitoring by the user.
- `values`
Event data as a list of "IAAlarmResult" instances of the monitored active alarm.
- `completed`
Status of the asynchronous transfer:
 - `True`: All alarms are read out.
 - `False`: Not all alarms are yet read out.

Example

Copy code

```
public void Odk_PlantObjectGetProperties()
{
    var myPlantModel = _runtime.GetObject<IPlantModel>();
    var strNodeName = ".hierarchy::RootNodeName/Node1";
    var plantObject = myPlantModel.GetPlantObject(strNodeName);
    Console.WriteLine("ViewName: {0} Name: {1}", plantObject.CurrentPlantView, plantObject.Name);
    // get the plant object properties by property names
    var plantObjectProperties = plantObject.GetProperties();
    if (plantObjectProperties != null)
    {
        var nCount = plantObjectProperties.Count;
        var listPropValues = plantObjectProperties.Read();
        Console.WriteLine("Number of Properties {0}", nCount);
        foreach (var item in listPropValues)
        {
            Console.WriteLine("Property Name is {0} ", item.Name);
            Console.WriteLine("Property Value is {0} ", item.Value);
            Console.WriteLine("Property Quality is {0} ", item.Quality);
            Console.WriteLine("Property Error is {0} ", item.Error);
        }
    }
}
```

8.7.3 IPlantObjectProperty

Description

The C# interface "IPlantObjectProperty" specifies the handling of properties of object instances of the plant model of a Runtime system. The properties represent the data structure of an object instance.

The object instance communicates with the automation system through the properties of the data structure. The values of the properties are obtained from linked process tags or internal tags.

You reference an "IPlantObjectProperty" object using the `IPlantObject.GetProperty` or `IPlantObject.GetProperties` method.

The interface inherits the `Dispose` method of the "IDisposable" interface of the .NET framework.

Members

The interface has the following properties and methods:

"Name" property

Name of the property

```
string Name { get; }
```

"Read" method

Reads the value of the "IPlantObjectProperty" instance synchronously and returns it as an "IPlantObjectPropertyValue" object. The value, the quality code and the time stamp of the property are determined when the property is read.

```
IPlantObjectPropertyValue Read()
```

"Write" method

Writes the value synchronously to the "IPlantObjectProperty" instance.

```
void Write(object Value)
```

- Value
New process value of the property

Example

Copy code

```
public void Odk_PlantObjectGetPropertyWrite()
{
    using (IPlantModel myPlantModel = runtime.GetObject<IPlantModel>())
    {
        string strNodeName = ".hierarchy::RootNodeName\\Node1";    //gets node for specified
Node path
        using (IPlantObject plantobject = myPlantModel.GetPlantObject(strNodeName))
        {
            System.Console.WriteLine(string.Format("ViewName: {0} Name: {1}",
plantobject.CurrentPlantView, plantobject.Name));
            if (plantobject != null)
            {
                string strName = "NodeProperty_1";
                using (IPlantObjectProperty plantObjectProperty =
plantobject.GetProperty(strName))
                {
                    if (plantObjectProperty != null)
                    {
                        IPlantObjectPropertyValue pValue = plantObjectProperty.Read();
                        System.Console.WriteLine(string.Format("Property Name: {0}
property value before write
                        operation: {1}", strName, pValue.Value));
                        Object value = 400; // Write Cpm Node Property
                        plantObjectProperty.Write(value);
                        IPlantObjectPropertyValue pValues = plantObjectProperty.Read();
                        System.Console.WriteLine(string.Format("Property Name: {0}
property value after write Operation: {1}", strName, pValues.Value));
                    }
                }
            }
        }
    }
}
```

8.7.4 IPlantObjectPropertyValue

Description

The C# interface "IPlantObjectPropertyValue" specifies the properties of process tags that are connected to an object instance property of the Runtime system.

Members

The following properties are specified in the interface:

"Name" property

Name of the tag

```
string Name { get; }
```

"Value" property

Value of the tag at the moment of the read operation.

```
object Value { get; }
```

"Quality" property

Quality code of the read operation of the tag.

```
Int32 Quality { get; }
```

"TimeStamp" property

Time stamp of the last successful read operation of the tag.

```
DateTime TimeStamp { get; }
```

"Error" method

Error code of the last read or write operation of the tag.

```
UInt32 Error { get; }
```

"OnPlantModelPropertySubscriptionHandler" delegate

Specifies the signature of the event handling method for the

"OnPlantModelPropertySubscriptionHandler" event of an "IPlantObjectPropertySet" instance.

```
public delegate void OnPlantModelPropertySubscriptionHandler(  
    IPlantObjectPropertySet sender,  
    IList<IPlantObjectPropertyValue> values);
```

- sender
Source of the event
- values
Event data as a list of "IPlantObjectPropertyValue" instances of the monitored active alarm.

Example

Copy code

```
public void Odk_PlantObjectGetPropertyRead()
{
    using (var myPlantModel = _runtime.GetObject<IPlantModel>())
    {
        var strNodeName = ".hierarchy::RootNodeName/Node1";
        //gets node for specified Node path
        using (var plantobject = myPlantModel.GetPlantObject(strNodeName))
        {
            Console.WriteLine("ViewName: {0} Name: {1}", plantobject.CurrentPlantView, plantobject.Name);
            var strName = "NodeProperty_1";
            using (var plantObjectProperty = plantobject.GetProperty(strName))
            {
                if (plantObjectProperty != null)
                {
                    // Read Cpm Node Property
                    var plantObjectPropertyValue = plantObjectProperty.Read();
                    if (null != plantObjectPropertyValue)
                    {
                        Console.WriteLine(
                            "Name= {0} TimeStamp {1} QualityCode {2} Error {3} Value {4}",
                            plantObjectPropertyValue.Name, plantObjectPropertyValue.TimeStamp,
                            plantObjectPropertyValue.Quality, plantObjectPropertyValue.Error,
                            plantObjectPropertyValue.Value);
                    }
                }
            }
        }
    }
}
```

8.7.5 IPlantObjectPropertySet

Description

The C# interface "IPlantObjectPropertySet" specifies properties, methods and events for optimized access to several IPlantObjectProperty instances of the Runtime system.

After initialization of the "IPlantObjectPropertySet" object, you have read/write access to multiple IPlantObjectProperty instances in one call. Simultaneous access has better performance and a lower communication load than single access to multiple properties.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

Member

The following properties, methods and events are specified in the interface:

"ContextId" property

"ContextId" can be useful for asynchronous methods. Users can assign "ContextId" if they have to assign the answer from the system to a read/write job.

Default value -1: "ContextId" is not used.

```
Int32 ContextId { get; set; }
```

"[propertyName]" property

Change the process value of a property of the "IPlantObjectPropertySet" instance.

The value is changed by the property only in the local "IPlantObjectPropertySet" instance. To write the values in the process image, a "Write" or "WriteAsync" method must be called.

```
object this[string propertyName] { get; set; }
```

- propertyName
Name of the property that is changed in the PropertySet.

"Count" property

The number of properties of the "IPlantObjectPropertySet" instance.

```
UInt32 Count { get; }
```

"Read" method

Supplies a list with all values of the "IPlantObjectProperty" instances contained in the "IPlantObjectPropertySet" instance. The values are read synchronously.

```
ICollection<IPlantObjectPropertyValue> Read();
```

"ReadAsync" method

Reads the values of all "IPlantObjectProperty" instances of the "IPlantObjectPropertySet" instance asynchronously.

```
void ReadAsync()
```

"Write" method

Writes the values of the "IPlantProperty" instances of the "PlantObjectPropertySet" instance synchronously to the Runtime system. Write operation errors are returned in a list with "IErrorResult" instances.

```
ICollection<IErrorResult> Write()
```

"WriteAsync" method

Writes the values of all "IPlantObjectProperty" instances of the "PlantObjectPropertySet" instance asynchronously to the Runtime system.

```
void WriteAsync()
```

"Subscribe" method

Subscribes all properties of the "IPlantObjectPropertySet" instance asynchronously for change monitoring.

```
void Subscribe()
```

"Add" method

Adds one or more "IPlantObjectProperty" instances to the "IPlantObjectPropertySet" instance.

The method can be called as follows:

- Adding multiple "IPlantObjectProperty" instances without value:

```
void Add(ICollection<string> propertyNames)
```

- name

A collection with the names of the "IPlantObjectProperty" instances.

- Adding a "IPlantObjectProperty" instance with value:

```
void Add(string propertyName, object value);
```

- propertyName

Name of the "IPlantObjectProperty" instance

- value

New process value of the "IPlantObjectProperty" instance

"Remove" method

Removes a property from the "IPlantObjectPropertySet" instance.

```
void Remove(string propertyName)
```

- propertyName

Name of the property that is being removed.

"Clear" method

Removes all properties from the "IPlantObjectPropertySet" instance.

```
void Clear()
```

"OnPropertySetReadComplete" event

After completion of the read operation of the "ReadAsync" method, the event calls an instance of the "OnPropertySetReadCompleteHandler" delegate.

Declares the event and the event handler for asynchronous read operations.

```
event OnPropertySetReadCompleteHandler OnPropertySetReadComplete
```

"OnPropertySetWriteComplete" event

After completion of the write operation of the "WriteAsync" method, the event calls an instance of the "OnPropertySetWriteCompleteHandler" delegate.

Declares the event and the event handler for asynchronous write operations.

```
event OnPropertySetWriteCompleteHandler OnPropertySetWriteComplete
```

"OnPlantModelPropertySubscriptionNotification" event

After the change of a monitored PropertySet, the event calls an instance of the "OnPlantModelPropertySubscriptionHandler" delegate.

Declares the event and the event handler when changing a PropertySet.

```
event OnPlantModelPropertySubscriptionHandler  
OnPlantModelPropertySubscriptionNotification
```

"OnPropertySetReadCompleteHandler" delegate

Specifies the signature of the event handling method for the "OnPropertySetReadComplete" event of an "IPlantObjectPropertySet" instance.

```
void OnPropertySetReadCompleteHandler(  
    IPlantObjectPropertySet sender,  
    UInt32 errorCode,  
    IList<IPlantObjectPropertyValue> values)
```

- sender
Source of the event
- errorCode
Supplies an error code when a global error has occurred.
- values
Event data as a list of "IPlantObjectPropertyValue" instances of the read property.

"OnPropertySetWriteCompleteHandler" delegate

Specifies the signature of the event handling method for the "OnPropertySetWriteComplete" event of an "IPlantObjectPropertySet" instance.

```
void OnPropertySetWriteCompleteHandler(  
    IPlantObjectPropertySet sender,  
    UInt32 errorCode,  
    IList<IPlantObjectPropertyValue> values)
```

- sender
Source of the event
- errorCode
Supplies an error code when a global error has occurred. When the error code is set, values is irrelevant.
- values
Event data as a list of "IPlantObjectPropertyValue" instances of the read property.

"OnPlantModelPropertySubscriptionHandler" delegate

Specifies the signature of the event handling method for the "OnPlantModelPropertySubscriptionNotification" event of an "IPlantObjectPropertySet" instance.

```
void OnPlantModelPropertySubscriptionHandler(  
    IPlantObjectPropertySet sender,  
    IList<IPlantObjectPropertyValue> values)
```

- sender
Source of the event
- values
Event data as a list of the changed "IPlantObjectPropertyValue" instances.

Example

Copy code

```

public void Odk_PlantObjectGetPropertySetReadAsync()
{
    try
    {
        IPlantModel myPlantModel = runtime.GetObject<IPlantModel>();
        string strNodeName = ".hierarchy::RootNodeName\\Node1";    //gets node for specified
Node path
        IPlantObject plantObject = myPlantModel.GetPlantObject(strNodeName);
        System.Console.WriteLine(string.Format("ViewName: {0} Name: {1}",
plantObject.CurrentPlantView,
        plantObject.Name));
        if (plantObject != null)
        {
            ICollection<string> PropNames = null; // get the plant objectproperties by
propepty names
            IPlantObjectPropertySet plantObjectPropertyset =
plantObject.GetPropertySet(PropNames);
            if (plantObjectPropertyset != null)
            {
                plantObjectPropertyset.OnPropertySetReadComplete +=
odkPlantModel_onReadComplete; // Read Plant
                Object properties values asynchronously
                plantObjectPropertyset.ReadAsync();
            }
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}

public void odkPlantModel_onReadComplete(IPlantObjectPropertySet sender, UInt32
SystemError, IList<IPlantObjectPropertyValue> Values)
{
    try
    {
        foreach (var value in Values)
        {
            Console.WriteLine("Name {0}", value.Name);
            Console.WriteLine("TimeStamp {0}", value.TimeStamp); Console.WriteLine("Value
{0}", value.Value);
            Console.WriteLine("Quality {0}", value.Quality); Console.WriteLine("Error {0}",
value.Error);
        }
    }
    finally
    {
        if (null != sender)
        {
            sender.Dispose();
        }
    }
}

```

Copy code

Copy code

```

public void odk_plantModelSubscribe()
{
    try
    {
        IPlantModel myPlantmodel = runtime.GetObject<IPlantModel>();
        string strNodeName = ".hierarchy::RootNodeName\\Node1";    //gets node for specified
Node path
        IPlantObject plantObject = myPlantmodel.GetPlantObject(strNodeName);
        System.Console.WriteLine(string.Format("ViewName: {0} Name: {1}",
plantObject.CurrentPlantView,
        plantObject.Name));
        if (plantObject != null)
        {
            ICollection<string> PropNames = null;
            IPlantObjectPropertySet plantObjectPropertyset =
plantObject.GetProperties(PropNames);
            if (plantObjectPropertyset != null)
            {
                // Assign callback function
                plantObjectPropertyset.OnPlantModelPropertySubscriptionNotification +=
                odkPlantModelPropertySet_OnDataChanged;
                plantObjectPropertyset.Subscribe();
            }
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}

public void odkPlantModelPropertySet_OnDataChanged(IPlantObjectPropertySet sender,
IList<IPlantObjectPropertyValue> Values)
{
    try
    {
        foreach (var value in Values)
        {
            Console.WriteLine("Name {0}", value.Name);
            Console.WriteLine("TimeStamp {0}", value.TimeStamp);
            Console.WriteLine("Value {0}", value.Value);
            Console.WriteLine("Quality {0}", value.Quality); Console.WriteLine("Error {0}",
value.Error);
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
    finally
    {
        if (null != sender)
        {
            sender.Dispose();
        }
    }
}

```

Copy code
}

8.7.6 IPlantObjectAlarmSubscription

Description

The C# interface "IPlantObjectAlarmSubscription" specifies methods for monitoring alarms of "IPlantObject" instances.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

Member

The following properties, methods and events are specified in the interface:

"Filter" property

SQL-type string for filtering the result set of active alarms.

```
string Filter { get; set; }
```

All properties of an alarm can be used in the filter string. The filter string can contain operators. Refer to the section Syntax of the alarm filter (Page 19).

"Language" property

Country identifier of the language of the monitored alarms. See also section Locale IDs of the supported languages (Page 21).

```
UInt32 Language { get; set; }
```

"IncludeChildren" property

If "true" is transferred, the alarm subscription only applies to the alarms of the "IPlantObject" instance and all its children in the hierarchy. If "false" is transferred, it only applies for the alarms of the "IPlantObject" instance.

```
bool IncludeChildren { get; set; }
```

"Start" method

Subscribe systems for monitoring of changes of active alarms.

```
void Start()
```

"Stop" method

Unsubscribe monitoring of active alarms.

```
void Stop()
```

"OnPlantObjectSubscribeAlarmHandler" event

Declares the event for the monitoring of alarms of an "IPlantObject" instance.

The event calls an instance of the "OnPlantObjectSubscribeAlarmHandler" delegate.

```
event OnPlantObjectSubscribeAlarmHandler  
OnPlantObjectSubscribeAlarmHandler;
```

"OnPlantObjectSubscribeAlarmHandler" delegate

Specifies the signature of the event handling method for the "OnPlantObjectSubscribeAlarmHandler" event of an "IPlantObject" instance.

```
public delegate void OnPlantObjectSubscribeAlarmHandler(  
    IPlantObjectAlarmSubscription sender,  
    UInt32 systemError,  
    string systemName,  
    IList<IArmResult> values);
```

- `sender`
Source of the event
- `systemError`
Supplies an error code when a global error has occurred. When the error code is set, `values` is irrelevant.
- `systemName`
Name of the Runtime system that is subscribed for alarm monitoring by the user.
- `values`
Event data as a list of "IArmResult" instances of the monitored active alarm.

Example

Copy code

```

public void Odk_GetAlarmSubscription()
{
    try
    {
        using (IPlantModel myPlantModel = runtime.GetObject<IPlantModel>())
        {
            string strNodeName = ".hierarchy::RootNodeName\\Node1";    //gets node for
specified Node path
            IPlantObject plantobject = myPlantModel.GetPlantObject(strNodeName);
            IPlantObjectAlarmSubscription alarmsub = plantobject.CreateAlarmSubscription();
            //Assign alarm handler
            if (alarmsub != null)
            {
                alarmsub.OnPlantObjectSubscribeAlarmHandler +=
alarm_OnPlantObjectSubscribeAlarmHandler;
                alarmsub.Filter = "";
                alarmsub.Language = 1033;
                alarmsub.IncludeChildren = false;
                alarmsub.Start();
            }
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}

public void alarm_OnPlantObjectSubscribeAlarmHandler(IPlantObjectAlarmSubscription sender,
UInt32 nGlobalError, String systemName, IList<IAlarmResult> value)
{
    try
    {
        foreach (var item in value)
        {
            System.Console.WriteLine(string.Format("Name: {0}", item.Name));
            System.Console.WriteLine(string.Format("InstanceID: {0}", item.InstanceID));
            System.Console.WriteLine(string.Format("AlarmClass: {0}", item.AlarmClassName));
            System.Console.WriteLine(string.Format("AlarmParameterValues: {0}",
item.AlarmParameterValues));
            System.Console.WriteLine(string.Format("AlarmText1: {0}", item.AlarmText1));
            System.Console.WriteLine(string.Format("Area: {0}", item.Area));
        }
    }
    finally
    {
        if (null != sender )
        {
            sender.Stop();
        }
    }
}

```

8.8 Interfaces of the Calendar option

8.8.1 ISHCCalendar

Description

The C# interface "ISHCCalendar" specifies the properties and methods of a calendar. The calendar is integrated via an "IPlantObject" instance.

The interface inherits the "Dispose()" method of the "IDisposable" interface of the .NET framework and the methods of the "ISHCGetObject" interface

Members

"Settings" property

Reference to an "ISHCCalenderSettings" instance which saves the settings of the calendar.

```
ISHCCalendarSettings Settings { get; }
```

"Category" property

Reference to an "ISHCCategoryProvider" instance with which you access categories of the calendar.

```
ISHCCategoryProvider Category { get; }
```

"DayTemplate" property

Reference to an "ISHCDayTemplatesProvider" instance with which you create, read, update and delete day templates for the calendar.

```
ISHCDayTemplatesProvider DayTemplate { get; }
```

"ShiftTemplate" property

Reference to an "ISHCShiftTemplatesProvider" instance with which you create, read, update and delete shift templates for the calendar.

```
ISHCShiftTemplatesProvider ShiftTemplate { get; }
```

"ActionTemplate" property

Reference to an "ISHCActionTemplatesProvider" instance with which you create, read, update and delete action templates for the calendar.

```
ISHCActionTemplatesProvider ActionTemplate { get; }
```

"Day" property

Reference to an "ISHCDayProvider" instance with which you create, read, update and delete days for the calendar.

```
ISHCDayProvider Day { get; }
```

Example

The following example serves as a basis for the other examples for the C# interfaces of the Calendar option.

It shows how you can obtain the "IPlantObject" instance and also an "ISHCCalendar" instance. The "ISHCCalendar" instance referenced via `calendar` is also used in the other examples.

Copy code

```
using Siemens.Runtime.HmiUnified.SHC;
using Siemens.Runtime.HmiUnified;

ISHCCalendar calendar = null;
// Connect to Runtime
IRuntime runtime = Runtime.Connect();
if (runtime != null)
{
    IPlantModel myPlantModel = runtime.GetObject<IPlantModel>();
    IPlantObject po = myPlantModel.GetPlantObject(".hierarchy::Plant/Unit1");
    if (po != null)
    {
        calendar = po.Calendar();
    }
}
```

8.8.2 ISHCCategory

Description

The C# interface "ISHCCategory" specifies the properties and methods of a time category of the time model.

The interface inherits the "Dispose()" method of the "IDisposable" interface of the .NET framework.

Members

"Name" property

The name of the category

```
string Name { get; }
```

"DisplayNames" property

A dictionary from UInt32/string pairs with the display names and their language code IDs.

```
IDictionary<UInt32, string> DisplayNames { get; }
```

"Descriptions" property

A dictionary from UInt32/string pairs with the descriptions of the category and its language code ID.

```
IDictionary<UInt32, string> Descriptions { get; }
```

"Color" property

The color of the category

```
Color Color { get; }
```

"Deleted" property

Saves the information on whether an already used category was deleted in Engineering.

```
bool Deleted { get; }
```

Example

```
static void PrintCategory(ISHCCategory category)
{
    if (null != category)
    {
        Console.WriteLine(" \nName:{0} \nColor:{1} \n Deleted:{2}\n", category.Name,
category.Color, category.Deleted);
    }
    IDictionary<uint, string> displayNames = category.DisplayNames;
    foreach (var item in displayNames)
    {
        Console.WriteLine("Language:{0} DisplayName:{1}", item.Key, item.Value);
    }
    IDictionary<uint, string> description = category.Descriptions;
    foreach (var item in description)
    {
        Console.WriteLine("Language:{0} Description:{1}", item.Key, item.Value);
    }
}
```

See also

Locale IDs of the supported languages (Page 21)

8.8.3 ISHCCategoryProvider

Description

The C# interface "ISHCCategoryProvider" provides you with read access to the "ISHCCategory" instances of an "ISHCCalendar" instance.

Members

"Browse" method

Supplies a collection with the categories of the calendar.

```
ICollection<ISHCCategory> Browse();
```

Example

Copy code

```
ICollection<ISHCCategory> categories = calendar.Category.Browse();  
foreach (var cat in categories)  
{  
    // do something  
}
```

8.8.4 ISHCCalendarSettings

Description

The C# interface "ISHCCalendarSettings" specifies properties and methods for access to the calendar settings of an "ISHCCalendar" instance.

The interface inherits the "Dispose()" method of the "IDisposable" interface of the .NET framework.

Members

"PlantObject" property

Reference to the "IPlantObject" instance to which the calendar belongs.

```
string PlantObject { get; }
```

"FirstDayOfWeek" property

Reference to the "ShcWeekDay" instance which is set as the first day of the week.

```
ShcWeekDay FirstDayOfWeek { get; }
```

"FirstWeekOfYear" property

Reference to the "ShcWeekStart" instance which is set as the first week of the year.

```
ShcWeekStart FirstWeekOfYear { get; }
```

"FiscalYearStartDay" property

The first day of the fiscal year

Default setting: 1

```
Byte FiscalYearStartDay { get; }
```

"FiscalYearStartMonth" property

The first month of the fiscal year

Default setting: 1

```
Byte FiscalYearStartMonth { get; }
```

"DayOffset" property

The offset with which the workday begins, calculated from midnight.

Default setting: 0

Maximum value: 24 hours

```
TimeSpan DayOffset { get; }
```

"Workdays" property

Number of workdays

```
UInt32 Workdays { get; }
```

"TimeZone" property

The Microsoft time zone

```
UInt32 TimeZone { get; }
```

Example

```
static void PrintCalendar(ISHCCalendarSettings calendar)
{
    if (null != calendar)
    {
        string cal = string.Format(" \n Workdays: {0} \n FirstDayOfWeek: {1} \n
FirstWeekOfYear: {2} \n FiscalYearStartDay: {3} \n FiscalYearStartMonth: {4} \n DayOffset:
{5} \n PlantObject: {6} \n \n TimeZone:{7} \n", calendar.Workdays, calendar.FirstDayOfWeek,
calendar.FirstWeekOfYear, calendar.FiscalYearStartDay, calendar.FiscalYearStartMonth,
calendar.DayOffset, calendar.PlantObject, calendar.TimeZone);
        Console.WriteLine(cal);
    }
}
```

8.8.5 ISHCTimeSlice

Description

The C# interface "ISHCTimeSlice" specifies the properties and methods of a time slice.

The interface inherits the "Dispose()" method of the "IDisposable" interface of the .NET framework.

Members

"StartTime" property

Time stamp with the start time of the time slice

```
DateTime StartTime { get; set; }
```

"Duration" property

The duration of the time slice

```
TimeSpan Duration { get; set; }
```

"Category" property

The time category of the time slice

```
string Category { get; set; }
```

8.8.6 ISHCDay

Description

The C# interface "ISHCDay" specifies the properties and methods of a day.

The interface inherits the "Dispose()" method of the "IDisposable" interface of the .NET framework.

Members

"Comments" property

A dictionary from UInt32/string pairs with the comments of the "ISHCDay" instance and their language code IDs.

```
IDictionary<UInt32, string> Comments { get; }
```

"StartTime" property

Time stamp with the start time of the "ISHCDay" instance.

```
DateTime StartTime { get; set; }
```

"IsCustomized" property

Saves information on whether the "ISHCDay" instance was edited by users.

```
bool IsCustomized { get; }
```

"DayTemplate" property

The "ISHCDayTemplate" instance from which the "ISHCDay" instance is derived.

```
string DayTemplate { get; set; }
```

"CreateShift" method

Instantiates an "ISHCShift" instance at the "ISHCDay" instance.


```
ISHCShift CreateShift(  
    ISHCShiftTemplate shcShiftTemplate,  
    TimeSpan startTime);
```

- `shcShiftTemplate`
Reference to the shift template from which the shift is derived
- `startTime`
Time stamp with the start time of the "ISHCShift" instance.

"DeleteShift" method

Deletes a shift of the "ISHCDay" instance.

```
void DeleteShift(  
    ISHCShift shcShift);
```

- `shcShift`
Reference to the shift to be deleted

"GetShifts" method

Supplies a list with all the shifts of the "ISHCDay" instance.

```
ICollection<ISHCShift> GetShifts();
```

"SetComment" method

Adds a new entry to the dictionary of the "Comment" property.

```
void SetComment(  
    UInt32 languageId,  
    string comment);
```

- `languageId`
The language code ID of the comment
- `comment`
A comment

See also

Locale IDs of the supported languages (Page 21)

8.8.7 ISHCDayProvider

Description

The C# interface "ISHCDayProvider" provides you with access to the days of an "ISHCCalendar" instance. With the methods of the provider, you can create, read, update and delete days.

Members

"Browse" method

Supplies a collection with the "ISHCDay" instances of the calendar.

```
ICollection<ISHCDay> Browse(
    DateTime startTime, DateTime end);
```

- **startTime**
Defines the start of the time period whose days are returned.
- **end**
Defines the end of the time period whose days are returned.

Example:

```
static void ReadDayWithShift()
{
    try
    {
        DateTime start = DateTime.Now.StartOfDay();
        DateTime end = DateTime.Now.EndOfDay();
        end = end.AddDays(3);
        ICollection<ISHCDay> days = calendar.Day.Browse(start, end);
        if (days.Count > 0)
        {
            foreach (var day in days)
            {
                // PrintDays(day);
                if (null != day)
                { string strDays = string.Format("\n StartTime :{0} \n IsCustomized :{1} \n
DayTemplate :{2} ", day.StartTime, day.IsCustomized, day.DayTemplate);
                Console.WriteLine(strDays);
                IDictionary<uint, string> Comments = day.Comments;foreach (var item in
Comments)
                {Console.WriteLine("\n Language:{0} DayComment:{1} \n", item.Key,
item.Value); }
            }
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}
```

"Create" method

Adds new "ISHCDay" instances to the calendar.

```
void Create(
    IList<ISHCDay> days);
```

- **days**
List with the new "ISHCDay" instances

Example:

```
static void CreateDayWithShift()
{
    try
    {
        IReadOnlyCollection<ISHCDayTemplate> Daytemplates =
calendar.DayTemplate.Browse(false);
        if (Daytemplates.Count > 0)
        {
            ISHCDayTemplate dayTemplate = Daytemplates.ElementAt(0);
            List<ISHCDay> DayList = new List<ISHCDay>();
            ISHCDay day = calendar.GetObject<ISHCDay>();
            day.DayTemplate = dayTemplate.Name;
            DateTime dtday = DateTime.Now;
            day.StartTime = dtday;
            day.SetComment(1033, "DaywithShift");
            DayList.Add(day);
            calendar.Day.Create(DayList);
            IReadOnlyCollection<ISHCShiftTemplate> ShiftTemplates =
calendar.ShiftTemplate.Browse(false);
            if (ShiftTemplates.Count > 0)
            {
                ISHCShiftTemplate ShiftTemplate = ShiftTemplates.ElementAt(0);
                ISHCShift dayShift = day.CreateShift(ShiftTemplate, new TimeSpan(18, 0, 0));
            }
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}
```

"Update" method

Updates "ISHCDay" instances of the calendar.

```
void Update(
    IList<ISHCDay> days);
```

- days
List of the "ISHCDay" instances to be updated

Example:

```
static void UpdateDayWithShift()
{
    try
    {
        DateTime start = DateTime.Now.StartOfDay();
        DateTime end = DateTime.Now.EndOfDay();
        end = end.AddDays(1);
        IReadOnlyCollection<ISHCDay> days = calendar.Day.Browse(start, end);
        if (days.Count > 0)
        {
            List<ISHCDay> list = new List<ISHCDay>();
            foreach (var day in days)
            {
                IReadOnlyCollection<ISHCShift> shifts = day.GetShifts();
                if (shifts != null)
                {
                    ISHCShift shift = shifts.ElementAt(0);
                    shift.GetTimeSlices().ElementAt(0).Category = "Maintenance";
                }
                list.Add(day);
            }
            calendar.Day.Update(list);
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}
```

"Delete" method

Deletes "ISHCDay" instances of the calendar.

```
void Delete(
    IList<ISHCDay> days);
```

- days
List of "ISHCDay" instances to be deleted

Example:

```
static void DeleteDayWithShift()
{
    try
    {
        DateTime start = DateTime.Now.StartOfDay();
        DateTime end = DateTime.Now.EndOfDay();
        end = end.AddDays(3);
        IReadOnlyCollection<ISHCDay> days = calendar.Day.Browse(start, end);
        if (days.Count > 0)
        {
            List<ISHCDay> list = new List<ISHCDay>();
            foreach (var day in days)
            {
                list.Add(day);
            }
            calendar.Day.Delete(list);
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}
```

8.8.8 ISHCDayTemplate

Description

The C# interface "ISHCDayTemplate" specifies the properties and methods of a day.

The interface inherits the "Dispose()" method of the "IDisposable" interface of the .NET framework.

Members

"Name" property

The name of the "ISHCDayTemplate" instance.

```
string Name { get; set; }
```

"DisplayNames" property

A dictionary from UInt32/string pairs with the display names of the "ISHCDayTemplate" instance and their language code IDs.

```
IDictionary<UInt32, string> DisplayNames { get; }
```

"Descriptions" property

A dictionary from UInt32/string pairs with the descriptions of the "ISHCDayTemplate" instance and their language code IDs.

```
IDictionary<UInt32, string> Descriptions { get; }
```

"Deleted" property

Saves information on whether the day template was deleted by users.

```
bool Deleted { get; }
```

"SetDisplayName" method

Sets the display name of the "ISHCDayTemplate" instance and its language code ID.

```
void SetDisplayName(
    UInt32 languageId,
    string displayName);
```

- languageId
The language code ID of the display name
- displayName
The display name

"SetDescription" method

Sets the description of the "ISHCDayTemplate" instance and its language code ID.

```
void SetDescription(
    UInt32 languageId,
    string description);
```

- languageId
The language code ID
- description
The description

"GetShifts" method

Supplies a collection with the shifts of the "ISHCDayTemplate" instances.

```
ReadOnlyList<ISHCShift> GetShifts();
```

"CreateShift" method

Adds a shift to the "ISHCDayTemplate" instance.

```
ISHCShift CreateShift(
    ISHCShiftTemplate template,
    TimeSpan startTime);
```

- template
Reference to the shift template on which the shift is based.
- startTime
Time stamp with the start time of the shift

"DeleteShift" method

Deletes a shift of the "ISHCDayTemplate" instance.

```
void DeleteShift(  
    ISHCShift shift);
```

- `shift`
Reference to the shift to be deleted

See also

Locale IDs of the supported languages (Page 21)

8.8.9 ISHCDayTemplatesProvider

Description

The C# interface "ISHCDayTemplatesProvider" provides you with access to the day templates of an "ISHCCalendar" instance. With the methods of the provider, you can create, read, update and delete day templates.

Members

"Browse" method

Supplies a collection with the "ISHCDayTemplate" instances of the calendar.

```
ICollection<ISHCDayTemplate> Browse(  
    bool includeDeleted);
```

- `includeDeleted`
Saves information on whether the collection also contains the deleted day templates.

Example:

```
static void ReadDayTemplateWithShift()
{
    try
    {
        IReadOnlyCollection<ISHCDayTemplate> dayTemplate =
calendar.DayTemplate.Browse(false);
        if (dayTemplate.Count > 0)
        {
            foreach (var template in dayTemplate)
            {
                PrintDayTemplates(template);
                ReadShiftsforDayTemplate(template);
            }
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}
```

"Create" method

Adds new "ISHCDayTemplate" instances to the calendar.

```
void Create(
    ICollection<ISHCDayTemplate> dayTemplates);
```

- dayTemplates
Collection with the new "ISHCDayTemplate" instances

Example:

```
static void CreateDayTemplateWithShift()
{
    try
    {
        ISHCDayTemplate Daytemplate = calendar.GetObject<ISHCDayTemplate>();
        if (null != Daytemplate)
        {
            List<ISHCDayTemplate> ListDayTemplate = new List<ISHCDayTemplate>();
            Daytemplate.Name = "DayTemplateName";
            Daytemplate.SetDescription(1033, "DayTemplateDescription");
            Daytemplate.SetDisplayName(1033, "DayTemplateDisplayName");
            ListDayTemplate.Add(Daytemplate);
            calendar.DayTemplate.Create(ListDayTemplate);
            IReadOnlyCollection<ISHCShiftTemplate> ShiftTemplates =
calendar.ShiftTemplate.Browse(false);
            if (ShiftTemplates.Count > 0)
            {
                ISHCShiftTemplate ShiftTemplate = ShiftTemplates.ElementAt(0);
                ISHCShift shift = Daytemplate.CreateShift(ShiftTemplate, new TimeSpan(1, 0,
0));
            }
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}
```

"Update" method

Updates the "ISHCDayTemplate" instances of the calendar.

```
void Update(
    ICollection<ISHCDayTemplate> dayTemplates);
```

- dayTemplates
Collection with the "ISHCDayTemplate" instances to be updated

Example:

```
static void UpdateDayTemplateWithShift()
{
    try
    {
        IReadOnlyCollection<ISHCDayTemplate> dayTemplate =
calendar.DayTemplate.Browse(false);
        if (dayTemplate.Count > 0)
        {
            List<ISHCDayTemplate> list = new List<ISHCDayTemplate>();
            foreach (var dayTemplates in dayTemplate)
            {
                dayTemplates.Name = "UpdatedDayTemplate";
                dayTemplates.SetDisplayName(1033, "UpdatedDayTemplateDisplayName");
                dayTemplates.SetDescription(1033, "UpdatedDayTemplateDescription");
                IReadOnlyCollection<ISHCShift> shifts = dayTemplates.GetShifts();
                if (shifts != null)
                {
                    ISHCShift shift = shifts.ElementAt(0);
                    shift.Duration = new TimeSpan(6, 0, 0);
                }
                list.Add(dayTemplates);
            }
            calendar.DayTemplate.Update(list);
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}
```

"Delete" method

Deletes "ISHCDayTemplate" instances of the calendar.

```
void Delete(
    ICollection<ISHCDayTemplate> dayTemplates);
```

- dayTemplates
Collection with the "ISHCDayTemplate" instances to be deleted

Example:

```
static void DeleteDayTemplateWithShift()
{
    try
    {
        IReadOnlyCollection<ISHCDayTemplate> dayTemplates =
calendar.DayTemplate.Browse(false);
        if (dayTemplate.Count > 0)
        {
            List<ISHCDayTemplate> list = new List<ISHCDayTemplate>();
            foreach (var dayTemplate in dayTemplates)
            {
                list.Add(dayTemplate);
            }
            calendar.DayTemplate.Delete(list);
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}
```

8.8.10 ISHCShiftTemplate

Description

The C# interface "ISHCShiftTemplate" specifies the properties and methods of a shift.

The interface inherits the "Dispose()" method of the "IDisposable" interface of the .NET framework.

Members

"Name" property

The name of the "ISHCShiftTemplate" instance

```
string Name { get; set; }
```

"DisplayNames" property

A dictionary from UInt32/string pairs with the display names of the "ISHCShiftTemplate" instance and their language code IDs.

```
IDictionary<UInt32, string> DisplayNames { get; }
```

"Descriptions" property

A dictionary from UInt32/string pairs with the descriptions of the "ISHCShiftTemplate" instance and their language code IDs.

```
IDictionary<UInt32, string> Descriptions { get; }
```

"Deleted" property

Saves information on whether the shift template was deleted by users.

```
bool Deleted { get; }
```

"Duration" property

The duration of the "ISHCShiftTemplate" instance.

```
TimeSpan Duration { get; set; }
```

"SetDisplayName" method

Sets the display name of the "ISHCShiftTemplate" instance and its language code ID.

```
void SetDisplayName(  
    UInt32 languageId,  
    string displayName);
```

- languageId
The language code ID of the display name
- displayName
The display name

"SetDescription" method

Sets the description of the "ISHCShiftTemplate" instance and its language code ID.

```
void SetDescription(  
    UInt32 languageId,  
    string description);
```

- languageId
The language code ID
- description
The description

"GetTimeSlices" method

Supplies a list with the time slices of the "ISHCShiftTemplate" instance.

```
ReadOnlyList<ISHCTimeSlice> GetTimeSlices();
```

"CreateTimeSlice" method

Adds a time slice to the "ISHCShiftTemplate" instance.

```
void CreateTimeSlice(  
    ISHCTimeSlice slice);
```

- slice
Reference to the new time slice

"DeleteTimeSlice" method

Deletes a time slice of the "ISHCShiftTemplate" instance.

```
void DeleteTimeSlice(  
    ISHCTimeSlice slice);
```

- `slice`
Reference to the time slice to be deleted

See also

Locale IDs of the supported languages (Page 21)

8.8.11 ISHCShiftTemplatesProvider

Description

The C# interface "ISHCShiftTemplatesProvider" provides you with access to the shift templates of an "ISHCCalendar" instance. With the methods of the provider, you can create, read, update and delete shift templates.

Members

"Browse" method

Supplies a collection with the "ISHCShiftTemplate" instances of the calendar.

```
ICollection<ISHCShiftTemplate> Browse(  
    bool includeDeleted);
```

- `includeDeleted`
Saves information on whether the collection also contains the deleted shift templates.

Example:

```
static void ReadShiftTemplatesWithTimeslice()
{
    try
    {
        Console.WriteLine("ReadShiftTemplate With Timeslice");
        IReadOnlyCollection<ISHCShiftTemplate> template =
calendar.ShiftTemplate.Browse(false);
        if (template.Count > 0)
        {
            foreach (var shift in template)
            {
                PrintShiftTemplates(shift);
                ReadTimeslicesforShiftTemplate(shift);
            }
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}",
ex.Message));
    }
}
```

"Create" method

Adds new "ISHCShiftTemplate" instances to the calendar.

```
void Create(
    ICollection<ISHCShiftTemplate> shiftTemplates);
```

- shiftTemplates
Collection with the new "ISHCShiftTemplate" instances of the calendar.

Example:

```
static void CreateShiftTemplateWithTimeSlice()
{
    try
    {
        using (ISHCShiftTemplate pShiftTemplate = calendar.GetObject<ISHCShiftTemplate>())
        {
            pShiftTemplate.Name = "ShiftTemplateName";
            pShiftTemplate.SetDisplayName(1033, "ShiftTemplateDisplayName");
            pShiftTemplate.SetDescription(1033, "ShiftTemplateDescriptions");
            pShiftTemplate.Duration = new TimeSpan(8, 0, 0);
            List<ISHCShiftTemplate> ShiftList = new List<ISHCShiftTemplate>();
            ShiftList.Add(pShiftTemplate);
            calendar.ShiftTemplate.Create(ShiftList);
            IReadOnlyCollection<ISHCCategory> categories = calendar.Category.Browse();
            if (categories.Count > 0)
            {
                ISHCCategory pCat = categories.ElementAt(0);
                ISHCTimeSlice pTimeSlice = calendar.GetObject<ISHCTimeSlice>();
                pTimeSlice.StartTime = DateTime.Now.StartOfDay();
                pTimeSlice.Duration = new TimeSpan(3, 0, 0);
                pTimeSlice.Category = pCat.Name;
                pShiftTemplate.CreateTimeSlice(pTimeSlice);
            }
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}",
ex.Message));
    }
}
```

"Update" method

Updates "ISHCShiftTemplate" instances of the calendar.

```
void Update(
    ICollection<ISHCShiftTemplate> shiftTemplates);
```

- shiftTemplates
Collection with the "ISHCShiftTemplate" instances to be updated

Example:

```

static void UpdateShiftTemplateWithTimeslice()
{
    try
    {
        IReadOnlyCollection<ISHCShiftTemplate> shiftTemplates =
calendar.ShiftTemplate.Browse(false);
        List<ISHCShiftTemplate> list = new List<ISHCShiftTemplate>();
        IReadOnlyCollection<ISHCCategory> categories = calendar.Category.Browse();
        ISHCCategory pCat = categories.ElementAt(1);
        foreach (var shifttemplate in shiftTemplates)
        {
            shifttemplate.Name = "UpdateShiftTemplate";
            shifttemplate.SetDisplayName(1033, "Updated DisplayName");
            shifttemplate.SetDescription(1033, "UpdatedDescription");
            shifttemplate.Duration = new TimeSpan(10, 0, 0);
            list.Add(shifttemplate);
            IReadOnlyCollection<ISHCTimeSlice> Timeslices =
shifttemplate.GetTimeSlices();
            if (Timeslices.Count > 0)
            {
                ISHCTimeSlice timeslice = Timeslices.ElementAt(0);
                timeslice.Duration = new TimeSpan(5, 0, 0);
                timeslice.Category = pCat.Name;
            }
        }
        calendar.ShiftTemplate.Update(list);
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}",
ex.Message));
    }
}

```

"Delete" method

Deletes "ISHCShiftTemplate" instances of the calendar.

```

void Delete(
    ICollection<ISHCShiftTemplate> shiftTemplates);

```

- shiftTemplates
Collection with the "ISHCShiftTemplate" instances of the calendar to be deleted.

Example:

```
static void DeleteShiftTemplateWithTimeslice()
{
    try
    {
        IReadOnlyCollection<ISHCShiftTemplate> shiftTemplates =
calendar.ShiftTemplate.Browse(false);
        if (shiftTemplates.Count > 0)
        {
            List<ISHCShiftTemplate> list = new List<ISHCShiftTemplate>();
            foreach (var shifttemplate in shiftTemplates)
            {
                list.Add(shifttemplate);
            }
            calendar.ShiftTemplate.Delete(list);
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}",
ex.Message));
    }
}
```

8.8.12 ISHCShift

Description

The C# interface "ISHCShift" specifies the properties and methods of a shift.

The interface inherits the "Dispose()" method of the "IDisposable" interface of the .NET framework.

Members

"StartTime" property

Time stamp with the start time of the "ISHCShift" instance.

```
DateTime StartTime { get; set; }
```

"Duration" property

The duration of the "ISHCShift" instance.

```
TimeSpan Duration { get; set; }
```

"ShiftTemplate" property

The shift template of the "ISHCShift" instance.

```
string ShiftTemplate { get; }
```

"IsCustomized" property

Saves information on whether the "ISHCShift" instance was edited by users.

```
bool IsCustomized { get; }
```

"DeltaKind" property

Saves information on how the time slices of the "ISHCShift" instance deviate from the shift template.

```
ShcDeltaType DeltaKind { get; }
```

The enumeration "ShcDeltaType" can contain the following values:

- Added (0)
- Modified (1)
- Deleted (2)

"ShiftId" property

Saves the ShiftID of the "ISHCShift" instance.

```
UInt32 ShiftId { get; set; }
```

"Comments" property

A dictionary from UInt32/string pairs with the descriptions of the "ISHCShift" instance and their language code IDs.

```
IDictionary<UInt32, string> Comments { get; }
```

"GetTimeSlices" method

Supplies a collection with the time slices of the "ISHCShift" instances.

```
ReadOnlyList<ISHCTimeSlice> GetTimeSlices();
```

"CreateTimeSlice" method

Adds a time slice to the "ISHCShift" instance.

```
void CreateTimeSlice(  
    ISHCTimeSlice slice);
```

- slice
Reference to the new time slice

"DeleteTimeSlice" method

Deletes a time slice of the "ISHCShift" instance.

```
void DeleteTimeSlice(  
    ISHCTimeSlice slice);
```

- slice
Reference to the time slice to be deleted

"SetComment" method

Adds a comment with language code ID to the "Comments" property.

```
void SetComment(  
    UInt32 languageId,  
    string comment);
```

- `languageId`
The language code ID of the comment
- `comment`
The comment

"CreateAction" method

Adds an action to the "ISHCShift" instance.

```
ISHCAction CreateAction(  
    ISHCActionTemplate actionTemplate,  
    TimeSpan offset);
```

- `actionTemplate`
The action template of the new action
- `offset`
The offset for the anchor point of the action, in relation to the starting point of the shift.
Positive and negative value allowed.

Example:

```
static void CreateActionUsingShift()  
{  
    try  
    {  
        DateTime start = DateTime.Now.StartOfDay();  
        DateTime end = DateTime.Now.EndOfDay();  
        end = end.AddDays(3);  
        ISHCShift Shift = calendar.Day.Read(start,  
end).ElementAt(0).GetShifts().ElementAt(0);  
        if (Shift != null)  
        {  
            ISHCAction Action =  
Shift.CreateAction(calendar.ActionTemplate.Read(false).ElementAt(0), new TimeSpan(5, 0,  
0));  
        }  
    }  
    catch (OdkException ex)  
    {  
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));  
    }  
}
```

"DeleteAction" method

Deletes an action of the "ISHCShift" instance.

```
void DeleteAction(ISHCAction shcAction);
```

- `shcAction`
Reference to the action to be deleted

"GetActions" method

Supplies a list with the actions of the "ISHCShift" instance.

```
ICollection<ISHCAction> GetActions();
```

Example:

```
static void ReadActionUsingShift()
{
    try
    {
        DateTime start = DateTime.Now.StartOfDay();
        DateTime end = DateTime.Now.EndOfDay();
        end = end.AddDays(3);
        ISHCShift Shift = calendar.Day.Read(start,
end).ElementAt(0).GetShifts().ElementAt(0);
        if (Shift != null)
        {
            ICollection<ISHCAction> action = Shift.GetActions();
            if (action != null)
            {
                ISHCAction actions = action.ElementAt(0);
                string Action = string.Format("\n Offset:{0} \n IsCustomized:
{1} ,actionTemplate:{2}", actions.Offset, actions.IsCustomized, actions.ActionTemplate);
                System.Console.WriteLine(Action);
            }
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}
```

See also

Locale IDs of the supported languages (Page 21)

8.8.13 ISHCAction**Description**

The C# interface "ISHCAction" specifies the properties and methods of an action.

The interface inherits the "Dispose()" method of the "IDisposable" interface of the .NET framework.

Members

"Offset" property

The offset for the anchor point of the "ISHCAction" instance in 100 nanoseconds in relation to the start point of its shift instance. Positive and negative value allowed.

```
TimeSpan Offset { get; set; }
```

"ActionTemplate" property

The action template of the "ISHCAction" instance

```
string ActionTemplate { get; }
```

"IsCustomized" property

Saves information on whether the "ISHCAction" instance was edited by users.

```
bool IsCustomized { get; }
```

"GetElements" method

Supplies a list with the action elements of the "ISHCAction" instance.

```
ICollection<ISHCActionElement> GetElements();
```

8.8.14 ISHCActionElement

Description

The C# interface "ISHCActionElement" specifies the properties and methods of an action element of an "ISHCAction" instance.

The interface inherits the "Dispose()" method of the "IDisposable" interface of the .NET framework.

Members

"ElementType" property

The type of the "ISHCActionElement" instance.

```
ShcActionElementType ElementType { get; }
```

The enumeration "ShcActionElementType" can contain the following values:

- Tag (0)
The action element controls a tag.

"Enabled" property

Saves the information on whether the "ISHCActionElement" instance is activated.

```
bool Enabled { get; set; }
```

"Offset" property

The offset of the "ISHCActionElement" instance in 100 nanoseconds in relation to the anchor point of its action. Positive and negative value allowed.

```
TimeSpan Offset { get; set; }
```

"Value" property

Value of the tag controlled by the "ISHCActionElement" instance

```
object Value { get; set; }
```

"ElementName" property

Name of the tag controlled by the "ISHCActionElement" instance

```
string ElementName { get; set; }
```

8.8.15 ISHCActionTemplate

Description

The C# interface "ISHCActionTemplate" specifies the properties and methods of the action template of an "ISHCAction" instance.

The interface inherits the "Dispose()" method of the "IDisposable" interface of the .NET framework.

Members

"Name" property

The name of the "ISHCActionTemplate" instance.

```
string Name { get; set; }
```

"DisplayNames" property

A dictionary from UInt32/string pairs with the display names of the "ISHCActionTemplate" instance and their language code IDs.

```
IDictionary<UInt32, string> DisplayNames { get; }
```

"Deleted" property

Saves information on whether the action template was deleted by users.

```
bool Deleted { get; }
```

"Descriptions" property

A dictionary from UInt32/string pairs with the descriptions of the "ISHCActionTemplate" instance and their language code IDs.

```
IDictionary<UInt32, string> Descriptions { get; }
```

"SetDisplayName" method

Sets the display name of the "ISHCActionTemplate" instance and its language code ID.

```
void SetDisplayName(  
    UInt32 languageId,  
    string displayName);
```

- languageId
The language code ID of the display name
- displayName
The display name

"SetDescription" property

Sets the description of the "ISHCActionTemplate" instance and its language code ID.

```
void SetDescription(  
    UInt32 languageId,  
    string description);
```

- languageId
The language code
- IDdescription
The description

"CreateElement" property

Adds an "ISHCActionTemplateElement" instance to the "ISHCActionTemplate" instance.

```
void CreateElement(  
    ISHCActionTemplateElement actiontemplateElement);
```

- actiontemplateElement
Reference to the new action template element

"DeleteElement" property

Deletes an "ISHCActionTemplateElement" instance of the "ISHCActionTemplate" instance.

```
void DeleteElement(  
    ISHCActionTemplateElement actiontemplateElement);
```

- actiontemplateElement
Reference to the action template element to be deleted

"GetElements" property

Supplies a list with action template elements of the "ISHCActionTemplate" instances.

```
ICollection<ISHCActionTemplateElement> GetElements();
```

See also

Locale IDs of the supported languages (Page 21)

8.8.16 ISHCActionTemplateElement

Description

The C# interface "ISHCActionTemplateElement" specifies the properties and methods of an action element of an "ISHCActionTemplate" instance.

The interface inherits the "Dispose()" method of the "IDisposable" interface of the .NET framework.

Members

"ElementType" property

The type of the "ISHCActionTemplateElement" instance.

```
ShcActionElementType ElementType { get; }
```

The enumeration "ShcActionElementType" can contain the following values:

- Tag (0)
The action element controls a tag.

"Offset" property

The offset of the "ISHCActionTemplateElement" instance in 100 nanoseconds in relation to the anchor point of its action template. Positive and negative value allowed.

```
TimeSpan Offset { get; set; }
```

"Value" property

Value of the tag controlled by the "ISHCActionTemplateElement" instance.

```
object Value { get; set; }
```

"ElementName" property

Name of the tag controlled by the "ISHCActionTemplateElement" instance.

```
string ElementName { get; set; }
```

8.8.17 ISHCActionTemplatesProvider

Description

The C# interface "ISHCActionTemplatesProvider" provides you with access to the action templates of an "ISHCCalendar" instance. With the methods of the provider, you can create, read, update and delete action templates.

Members

"Browse" method

Supplies a collection with the "ISHCActionTemplate" instances of the calendar.

```
ICollection<ISHCActionTemplate> Browse(  
    bool includeDeleted);
```

- `includeDeleted`
Saves information on whether the collection also contains the deleted action templates.

Example:

```
static void ReadActionTemplate()  
{  
    try  
    {  
        Console.WriteLine("ReadActionTemplate");  
        ICollection<ISHCActionTemplate> actionTemplate =  
calendar.ActionTemplate.Browse(false);  
        foreach (var template in actionTemplate)  
        {  
            PrintActionTemplates(template);  
        }  
    }  
    catch (OdkException ex)  
    {  
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));  
    }  
}
```

"Create" method

Adds new "ISHCActionTemplate" instances to the calendar.

```
void Create(  
    ICollection<ISHCActionTemplate> actionTemplates);
```

- `actionTemplates`
Collection with the new "ISHCActionTemplate" instances

Example:

```
static void CreateActionTemplateWithActionTemplateElement()
{
    try
    {
        ISHCActionTemplate pActionTemplate = calendar.GetObject<ISHCActionTemplate>();
        if (pActionTemplate != null)
        {
            pActionTemplate.Name = "ActionTemplate";
            pActionTemplate.SetDisplayName(1033, "ActionDisplayName");
            pActionTemplate.SetDescription(1033, "ActionDescription");
            List<ISHCActionTemplate> ActionList = new List<ISHCActionTemplate>();
            ActionList.Add(pActionTemplate);
            calendar.ActionTemplate.Create(ActionList);
            ISHCActionTemplateElement pActionTemplateElement =
calendar.GetObject<ISHCActionTemplateElement>();
            if (pActionTemplateElement != null)
            {
                pActionTemplateElement.ElementName = "HMI_RT_1::Unit1.Member_1";
                pActionTemplateElement.Value = false;
                pActionTemplateElement.Offset = new TimeSpan(4, 0, 0);
                pActionTemplate.CreateElement(pActionTemplateElement);
            }
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}
```

"Update" method

Updates "ISHCActionTemplate" instances of the calendar.

```
void Update(
    ICollection<ISHCActionTemplate> actionTemplates);
```

- actionTemplates
Collection with the "ISHCActionTemplate" instances to be updated

Example:

```
static void UpdateActionTemplateWithActionTemplateElement()
{
    IReadOnlyCollection<ISHCActionTemplate> actionTemplates =
calendar.ActionTemplate.Browse(false);
    List<ISHCActionTemplate> list = new List<ISHCActionTemplate>();
    foreach (var actionTemplate in actionTemplates)
    {
        actionTemplate.Name = "UpdatedActionTemplate";
        actionTemplate.SetDisplayName(1033, "UpdatedDisplayName ");
        actionTemplate.SetDescription(1033, "UpdatedDescription");
        IReadOnlyCollection<ISHCActionTemplateElement> action =
actionTemplate.GetElements();
        ISHCActionTemplateElement templateElement = action.ElementAt(0);
        templateElement.Offset = new TimeSpan(6, 0, 0);
        list.Add(actionTemplate);
    }
    calendar.ActionTemplate.Update(list);
}
```

"Delete" method

Deletes "ISHCActionTemplate" instances of the calendar.

```
void Delete(
    ICollection<ISHCActionTemplate> actionTemplates);
```

- actionTemplates
Collection with the "ISHCActionTemplate" instances to be deleted

Example:

```
static void DeleteActionTemplateWithActionTemplateElement()
{
    try
    {
        IReadOnlyCollection<ISHCActionTemplate> actionTemplates =
calendar.ActionTemplate.Browse(false);
        if (actionTemplates.Count > 0)
        {
            List<ISHCActionTemplate> list = new List<ISHCActionTemplate>();
            foreach (var actionTemplate in actionTemplates)
            {
                list.Add(actionTemplate);
            }
            calendar.ActionTemplate.Delete(list);
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}
```

8.9 Interfaces of the contexts

8.9.1 IContextLogging

Description

The C# interface "IContextLogging" defines events and methods for creating and reading "IContextDefinition" instances as well as for starting, stopping, monitoring and reading their "ILoggedContext" instances. You can use "ILoggedContext" instances to filter runtime data, for example, for alarms that fall within the time period of a particular "ILoggedContext" instance.

The interface inherits the Dispose() method of the "IDisposable" interface of the .NET framework.

The methods trigger an exception in case of an error.

Members

"CreateContextDefinitions" method

Creates ContextDefinitions in the database.

```
void CreateContextDefinitions(ICollection<IContextDefinition>  
contextDefinitions)
```

- contextDefinitions:
Collection with "IContextDefinition" instances

"ReadContextDefinitions" method

Reads ContextDefinitions from the database. The instances can be filtered by plant object and HMIContextProviderType .

```
void ReadContextDefinitions(ICollection<string> plantViewPaths =
null, ICollection<HmiContextProviderType> providerTypes = null,
HmiSortingMode sortingMode = HmiSortingMode.Ascending)
```

- (optional) `plantViewPaths`:
Limits the read operation to "IContextDefinition" instances from this collection of plant objects.
- (optional) `providerTypes`:
Limits the read operation to "IContextDefinition" instances with HmiContextProvider types from this collection.
The enumeration "HmiContextProviderType" can contain the following values:
 - NoContext = 0
 - Calendar = 1
For "IContextDefinition" instances generated by the PI Option Calendar.
 - PerformanceInsightMachineState = 2
For "IContextDefinition" instances generated by the PI Option Performance Insight.
 - LineCoordinator = 3
For "IContextDefinition" instances generated by the PI Option Line Coordinator.
 - UserDefined = 5
It is a user-defined "IContextDefinition" instance, created by ODK, for example.
- (optional) `sortingMode`:
The "HmiSortingMode" according to which the read "IContextDefinition" instances are sorted.
The enumeration "HmiSortingMode" can contain the following values:
 - Ascending = 1
Default setting
 - Descending = 2

"ReadContexts" method

Reads "ILoggedContext" instances of a specific time period. The instances can be filtered using a "IContextFilter" instance.

```
void ReadContexts(DateTime start, DateTime end, IContextFilter
ContextFilter=null, HmiSortingMode sortingMode =
HmiSortingMode.Ascending)
```

- `start`:
The start time of the period within which the "ILoggedContext" instances must lie.
- `end`:
The end time of the period within which the "ILoggedContext" instances must lie.

8.9 Interfaces of the contexts

- (optional) `ContextFilter`:
The "IContextFilter" instance whose filter settings are used.
- (optional) `sortingMode`:
The "HmiSortingMode" according to which the read "ILoggedContext" instances are sorted.
The enumeration "HmiSortingMode" can contain the following values:
 - Ascending = 1
Default setting
 - Descending = 2

"StartContext" method

Creates a new "ILoggedContext" instance for a "IContextDefinition" instance.

```
void StartContext(string contextName, HmiContextProviderType  
providerType, object contextValue, DateTime startTime, UInt32  
qualityCode)
```

- `contextName`:
The name of the "IContextDefinition" instance for which the context log entry is created.
- (optional) `providerType`:
The HmiContextProviderType that the "IContextDefinition" instance of the context log entry must have.
The enumeration "HmiContextProviderType" can contain the following values:
 - NoContext = 0
 - Calendar = 1
For "IContextDefinition" instances generated by the PI Option Calendar.
 - PerformanceInsightMachineState = 2
For "IContextDefinition" instances generated by the PI Option Performance Insight.
 - LineCoordinator = 3
For "IContextDefinition" instances generated by the PI Option Line Coordinator.
 - UserDefined = 5
It is a user-defined "IContextDefinition" instance, created by ODK, for example.
- `contextValue`:
The context value of the context log entry
- `startTime`:
The start time of the new context log entry
- `qualityCode`:
The QualityCode of the context value of the context log entry

"StopContext" method

Stops the currently running "ILoggedContext" instance of an "IContextDefinition" instance.

```
void StopContext(string contextName, HmiContextProviderType
providerType, DateTime endtime)
```

- `contextName`:
The name of the "IContextDefinition" instance whose context log entry is stopped.
- (optional) `providerType`:
The `HmiContextProviderType` that the "IContextDefinition" instance of the context log entry must have.
The enumeration "HmiContextProviderType" can contain the following values:
 - `NoContext = 0`
 - `Calendar = 1`
For "IContextDefinition" instances generated by the PI Option Calendar.
 - `PerformanceInsightMachineState = 2`
For "IContextDefinition" instances generated by the PI Option Performance Insight.
 - `LineCoordinator = 3`
For "IContextDefinition" instances generated by the PI Option Line Coordinator.
 - `UserDefined = 5`
It is a user-defined "IContextDefinition" instance, created by ODK, for example.
- `endtime`:
End time of the context log entry

"Add" method

Adds a "IContextDefintion" instance to a vector. The methods `Clear()`, `Subscribe()` and `CancelSubscription()` can be called for the instances of the vector.

```
void Add(string contextName, HmiContextProviderType providerType)
```

- `contextName`:
The name of the "IContextDefintion" instance
- `providerType`:
The `HmiContextProviderType` of the "IContextDefintion" instance
The enumeration "HmiContextProviderType" can contain the following values:
 - `NoContext = 0`
 - `Calendar = 1`
For "IContextDefinition" instances generated by the PI Option Calendar.
 - `PerformanceInsightMachineState = 2`
For "IContextDefinition" instances generated by the PI Option Performance Insight.
 - `LineCoordinator = 3`
For "IContextDefinition" instances generated by the PI Option Line Coordinator.
 - `UserDefined = 5`
It is a user-defined "IContextDefinition" instance, created by ODK, for example.

"Clear" method

Deletes the "IContextDefintion" instances added via `Add()` from the vector.

```
void Clear()
```

"Subscribe" method

Subscribes the "IContextDefinition" instances added to the vector via `Add()` for monitoring.

```
void Subscribe()
```

"CancelSubscription" method

Unsubscribes the "IContextDefinition" instances added to the vector with `Add()` from monitoring.

```
void CancelSubscribe()
```

"OnContextDefinitionCreate" event

The event calls the delegate "OnContextDefinitionCreateDelegate" after the creation of ContextDefinitions.

Declares the event and the event handler for creating "IContextDefinition" instances.

```
event OnContextDefinitionCreateDelegate OnContextDefinitionCreate;
```

"OnContextDefinitionReadReply" event

After reading the ContextDefinitions, the event calls the "OnContextDefinitionReadReplyDelegate" delegate.

Declares the event and the event handler for reading "IContextDefinition" instances.

```
event OnContextDefinitionReadReplyDelegate  
OnContextDefinitionReadReply;
```

"OnLoggedContextReadReply" event

After reading the LoggedContexts, the event calls the "OnContextReadReplyDelegate" delegate.

Declares the event and the event handler for reading "ILoggedContext" instances.

```
event OnContextReadReplyDelegate OnLoggedContextReadReply;
```

"OnContextDataChanged" event

Event calls the delegate "OnContextDataChangedHandler" when starting or stopping a monitored "ILoggedContext" instance.

Declares the event and the event handler for monitoring "ILoggedContext" instances.

```
event OnContextDataChangedHandler OnContextDataChanged;
```

"OnContextDefinitionCreateDelegate" delegate

Specifies the signature of the event handling method for the "OnContextDefinitionCreate" event of the "IContextLogging" interface.

```
public delegate void  
OnContextDefinitionCreateDelegate(IContextLogging sender,  
    UInt32 globalError,  
    string systemName,  
    List<IContextError> errors,
```



```
bool completed)
```

- **sender:**
Source of the event
- **globalError:**
Global ErrorCode when a global error occurs during the call. All other parameters are invalid in this case.
- **systemName:**
Name of the system on which the ContextDefinitions have been created.
- **errors:**
List with the instance-specific errors that were generated when the ContextDefinitions were created.
- **completed:**
Status of the asynchronous transfer:
 - True: All ContextDefinitions have been notified.
 - False: Additional notifications are expected.

"OnContextDefinitionReadReplyDelegate" delegate

Specifies the signature of the event handling method for the "OnContextDefinitionReadReply" event of the "IContextLogging" interface.

```
public delegate void
OnContextDefinitionReadReplyDelegate(IContextLogging sender,
    UInt32 globalError,
    string systemName,
    IList<IContextDefinition> contextDefinitionData,
    bool completed)
```

- **sender:**
Source of the event
- **globalError:**
Global ErrorCode when a global error occurs during the call. All other parameters are invalid in this case.
- **systemName:**
Name of the system on which the ContextDefinitions have been created.
- **contextDefinitionData:**
List of read "IContextDefinition" instances
- **completed:**
Status of the asynchronous transfer:
 - True: All ContextDefinitions have been notified.
 - False: Additional notifications are expected.

"OnContextReadReplyDelegate" delegate

Specifies the signature of the event handling method for the "OnLoggedContextReadReply" event of the "IContextLogging" interface.

```
public delegate void OnContextReadReplyDelegate(IContextLogging
sender,
```

8.9 Interfaces of the contexts

```
UInt32 globalError,  
string systemName,  
IList<ILoggedContext> loggedContexts,  
bool completed)
```

- sender:
Source of the event
- globalError:
Global ErrorCode when a global error occurs during the call. All other parameters are invalid in this case.
- systemName:
Name of the system on which the "ILoggedContext" instances have been created.
- loggedContexts:
List with "ILoggedContext" instances that were started or stopped.
- completed:
Status of the asynchronous transfer:
 - True: All ContextDefinitions have been notified.
 - False: Additional notifications are expected.

"OnContextDataChangedHandler" delegate

Specifies the signature of the event handling method for the "OnContextDataChanged" event of the "IContextLogging" interface.

```
public delegate void OnContextDataChangedHandler(IContextLogging  
sender,  
IList<ILoggedContext> loggedContexts)
```

- sender:
Source of the event
- loggedContexts:
List with "ILoggedContext" instances

Example

Copying code

```
public void CreateContextDefinitions()
{
    Console.WriteLine("CreateContextDefinitions \n");
    var contextLogging = _runtime.GetObject<IContextLogging>();
    List<IContextDefinition> contextDefinitions = new List<IContextDefinition>();
    for (int i = 0; i < 5; i++)
    {
        var contextDefinition = _runtime.GetObject<IContextDefinition>();
        contextDefinition.PlantViewPath = ".hierarchy::Plant/Node1_1";
        contextDefinition.DataType = HmiContextDataType.DInt;
        Dictionary<UInt32, string> displayLanguages = new Dictionary<uint, string>();
        displayLanguages[1033] = "english";
        displayLanguages[1032] = "deutsch";
        contextDefinition.DisplayNames = displayLanguages;
        Random rnd = new Random();
        int num = rnd.Next(1, 1000);
        contextDefinition.Name = "CD_" + num + i.ToString();
        contextDefinitions.Add(contextDefinition);
        contextLogging.OnContextDefinitionCreate +=
ContextLogging_OnContextDefinitionCreate;
        contextLogging.CreateContextDefinitions(contextDefinitions);
        _event.WaitOne();
        _event.Reset();
        contextLogging.Dispose();
    }
}
```

Copying code

```
private void ContextLogging_OnContextDefinitionCreate(IContextLogging sender, UInt32
globalError, string systemName, IList<IContextError> errors, bool completed)
{
    if (globalError != 0)
    {
        Console.WriteLine("System Name:{0} Error:{1}", systemName, globalError);
    }
    else
    {
        if (null != errors)
        {
            foreach (var error in errors)
            {
                Console.WriteLine("ContextName:{0} Error:{1}", error.Name, error.Error);
                Console.WriteLine();
            }
        }
    }
}
```

Copying code

```
public void ReadContextDefinitionsWithFilter()
{
    Console.WriteLine("ReadContextDefinitionsWithFilter \n");
    var contextLogging = _runtime.GetObject<IContextLogging>();
    contextLogging.OnContextDefinitionReadReply +=
OnContextDefinitionReadReplyForContextLogging;
    List<string> plantobjectsfilter = new List<string>();
    plantobjectsfilter.Add(".hierarchy::Plant/Node1_1");
    List<HmiContextProviderType> contextProviderType = new List<HmiContextProviderType>();
    contextProviderType.Add(HmiContextProviderType.UserDefined);
    contextLogging.ReadContextDefinitions(plantViewPaths: plantobjectsfilter,
providerTypes: contextProviderType, sortingMode: HmiSortingMode.Ascending);
    _event.WaitOne();
    _event.Reset();
    contextLogging.Dispose();
}
```

Copying code

```
public void StartContext()
{
    Console.WriteLine("StartContext \n");
    var contextLogging = _runtime.GetObject<IContextLogging>();
    startDt = System.DateTime.Now;
    object value = "Orange Juice";
    uint quality = 192;
    var strContextName = "ContextName_" + strContext_Unique_Num;
    var providerType = HmiContextProviderType.UserDefined;
    contextLogging.StartContext(strContextName, providerType, value, startDt, quality); ;
}
```

Copying code

```
public void StopContext()
{
    Console.WriteLine("StopContext \n");
    var contextLogging = _runtime.GetObject<IContextLogging>();
    endDt = System.DateTime.Now;
    var strContextName = "ContextName_" + strContext_Unique_Num;
    var providerType = HmiContextProviderType.UserDefined;
    contextLogging.StopContext(strContextName, providerType, endDt);
}
```

Copying code

```

void OnContextDefinitionReadReplyForContextLogging(IContextLogging sender, UInt32
globalErrors, string systemName, IList<IContextDefinition> contextDefinitions, bool
completed)
{
    if (globalErrors != 0)
    {
        Console.WriteLine("System Name:{0} Error:{1}", systemName, globalErrors);
    }
    else
    {
        if (null != contextDefinitionData)
        {
            foreach (var cd in contextDefinitionData)
            {
                Console.WriteLine("PlantViewPath:{0} Name:{1} Datatype:{2} Error:{3}",
cd.PlantViewPath, cd.Name, cd.DataType, cd.Error);
                Console.WriteLine();
            }
        }
    }
}

```

Copying code

```

void OnLoggedContextReadReplyForContextLogging(IContextLogging sender, UInt32
globalErrors, string systemName, IList<ILoggedContext> loggedContexts, bool completed)
{
    if (globalErrors != 0)
    {
        Console.WriteLine("System Name:{0} Error:{1}", systemName, globalErrors);
    }
    else
    {
        if (null != loggedContexts)
        {
            foreach (var lc in loggedContexts)
            {
                Console.WriteLine("StartTime:{0} EndTime:{1} Value:{2} Quality:{3} Error:
{4}", lc.StartTime, lc.EndTime, lc.Value, lc.Quality, lc.Error);
                Console.WriteLine();
            }
        }
    }
}

```

8.9 Interfaces of the contexts

Copying code

```
public void Subscribe()
{
    Console.WriteLine("Subscribe \n");
    var contextLogging = _runtime.GetObject<IContextLogging>();
    contextLogging.OnContextDataChanged += OnContextDataChangedForContextLogging;
    CreateContextDefinition();
    var name = "ContextName_" + strContext_Unique_Num;
    var providerType = HmiContextProviderType.UserDefined;
    contextLogging.Add(name, providerType);
    contextLogging.Subscribe();
    StartContext();
    StopContext();
    contextLogging.CancelSubscribe();
    contextLogging.Dispose();
}
```

8.9.2 IContextDefinition

Description

The C# interface "IContextDefinition" specifies properties for the definition of "IContextDefinition" instances. "ILoggingContext" instances can be created using the Start() and Stop() methods based on an "IContextDefinition" instance.

The interface inherits the Dispose() method of the "IDisposable" interface of the .NET framework.

Members

"PlantViewPath" property

Sets the path to the plant object of the "IContextDefinition" instance

```
string PlantViewPath { get; set; }
```

Example: ".hierarchy::Plant/Station"

"ProviderType" property

The source that creates the instance.

Is used together with "Name" to uniquely identify a "IContextDefinition" instance.

```
HmiContextProviderType ProviderType { get; }
```

The enumeration "HmiContextProviderType" can contain the following values:

(optional) providerType:

The HmiContextProviderType that the "IContextDefinition" instance of the context log entry must have.

The enumeration "HmiContextProviderType" can contain the following values:

- NoContext = 0
- Calendar = 1
For "IContextDefinition" instances generated by the PI Option Calendar.
- PerformanceInsightMachineState = 2
For "IContextDefinition" instances generated by the PI Option Performance Insight.
- LineCoordinator = 3
For "IContextDefinition" instances generated by the PI Option Line Coordinator.
- UserDefined = 5
It is a user-defined "IContextDefinition" instance, created by ODK, for example.

"Name" property

The name of the "IContextDefinition" instance

Is used together with "ProviderType" to uniquely identify a "IContextDefinition" instance.

```
string Name { get; set; }
```

"DisplayNames" property

The display name of the "IContextDefinition" instance

```
IDictionary<UInt32, string> DisplayNames { get; set; }
```

"DataType" property

The data type of the "IContextDefinition" instance

```
HmiContextDataType DataType { get; set; }
```

The enumeration "HmiContextDataType" can contain the following values:

- Bool = 0x01
- SInt = 0x02
- Int = 0x03
- DInt = 0x04
- LInt = 0x05
- USInt = 0x06
- UInt = 0x07
- UDInt = 0x08
- ULLInt = 0x09
- Real = 0x0A
- LReal = 0x0B
- LTime = 0x0C
- DateTime = 0x0D
- Byte = 0x11
- Word = 0x12

- DWord = 0x13
- LWord = 0x14
- String = 0x32

"Error" property

The error code of the "IContextDefinition" instance
Is set if the instance is read incorrectly.

```
UInt32 Error { get; }
```

8.9.3 ILoggedContext

Description

The C# interface "ILoggedContext" defines properties of context log entries of an "IContextDefinition" instance.

The context log entries are started and stopped using methods of the "IContextLogging" interface.

Members

"StartTime" property

The start time of the "IContextLogging" instance

```
DateTime StartTime { get; }
```

"EndTime" property

The end time of the "IContextLogging" instance

```
DateTime EndTime { get; }
```

"Error" property

The error code of the "IContextLogging" instance

```
UInt32 Error { get; }
```

"Value" property

The name that the "IContextDefinition" instance of the "IContextLogging" instance has in the user interface.

Must correspond to the data type from "DataType" of the "IContextDefinition" instance.

```
object Value { get; }
```

"Quality" property

The QualityCode of the context value

```
UInt32 Quality { get; }
```


8.9.4 IContextError

Description

The C# interface "IContextError" specifies properties of error results that occur when generating ContextDefinitions in the database.

Members

"Name" property

Name of the "IContextDefintion" instance

```
string Name { get; }
```

"Error" property

The error code

```
UInt32 Error { get; }
```

8.9.5 IContextFilter

Description

The C# interface "IContextFilter" specifies the properties for filtering according to "ILoggedContext" instances.

Members

"Name" property

The name of the "IContextDefinition" instance for whose "ILoggedContext" instances filtering is performed.

```
string Name { set; get; }
```

"ProviderType" property

The HmiContextProviderType of an "IContextDefinition" instance for whose "ILoggedContext" instances the filtering is performed.

```
HmiContextProviderType ProviderType { set; get; }
```

The enumeration "HmiContextProviderType" can contain the following values:

- NoContext = 0
- Calendar = 1
For "IContextDefinition" instances generated by the PI Option Calendar.
- PerformanceInsightMachineState = 2
For "IContextDefinition" instances generated by the PI Option Performance Insight.

8.9 Interfaces of the contexts

- LineCoordinator = 3
For "IContextDefinition" instances generated by the PI Option Line Coordinator.
- UserDefined = 5
It is a user-defined "IContextDefinition" instance, created by ODK, for example.

"Operator" property

The filter operator

```
string Operator { set; get; }
```

The operator is applied to the value. The following operators are allowed:

- For values with data type Int and Real:
 - =
 - !=
 - <
 - >
 - <=
 - >=
- For values with data type String:
 - LIKE
 - =

Strings must always be fully specified.

```
string Operator { set; get; }
```

Examples of operators: ">", "<", ">=", "<=" and "="

"Value" property

To filter by "Value" of the "ILoggedContext" instance

```
object Value { set; get; }
```

Example

Copying code

```
public void ReadContextWithFilter()
{
    Console.WriteLine("ReadContextWithFilter \n");
    var contextLogging = _runtime.GetObject<IContextLogging>();
    contextLogging.OnLoggedContextReadReply += OnLoggedContextReadReplyForContextLogging;
    IContextFilter Filter = _runtime.GetObject<IContextFilter>();
    if (Filter != null)
    {
        Filter.Name = "ContextName_" + strContext_Unique_Num;
        Filter.ProviderType = HmiContextProviderType.UserDefined;
        Filter.Operator = "=";
        Filter.Value = "Orange Juice";
    }
    contextLogging.ReadContexts(startDt, endDt, Filter, HmiSortingMode.Ascending);
    _event.WaitOne();
    _event.Reset();
    contextLogging.Dispose();
}
```


Description of the C++ interfaces

9.1 Error codes of the C++ interfaces

All methods that have defined a `CFRESULT` return `CFSUCCESS` if the method was run through successfully. Otherwise, they return a corresponding error code.

9.2 Interfaces of the Runtime environment

9.2.1 IOdkRt

Description

The C++ interface "IOdkRt" specifies methods for the connection to the Runtime system and the error handling.

Members

The following methods are specified in the interface:

"Connect" method

Connect to a Runtime project.

```
CFRESULT Connect (
    const CFSTR context,
    IRuntime **ppRuntime,
    const CTSTR user = nullptr,
    const CTSTR password = nullptr)
```

- `context`
[in]: Name of the runtime project

Note

The name of the Runtime project is not used in the current version. An empty string must be passed in order to connect to the locally run Runtime project.

- `IRuntime`
[out]: Points to the initialized "IRuntime" object that the ODK object model makes available.

- user
[in]: User name

Note

Can only be used in a future version!

- password
[in]: Password

Note

Can only be used in a future version!

"Close" method

Enable configuration files and plug-ins of the Runtime system.

CFRESULT Close()

"GetErrorHandler" method

Transfers an "IErrorInfo" object for error handling.

CFRESULT GetErrorHandler(IErrorInfo** pErrorInfo)

IErrorInfo

[out]: Points to an "IErrorInfo" object.

Example

Connect to the Runtime system of the active project:

Copy code

```
IRuntimePtr pRuntime;

CFRESULT Connect()
{
    // Connect to running project
    CCfString projectName = L"";
    CFRESULT retVal = Connect(projectName, &pRuntime);

    if(CF_FAILED(retVal))
        PrintErrorInformation(retVal, L"Connect", pRuntime);

    return retVal;
}
```

Error handling when reading out installed options of the Runtime system:

Copy code

```
void GetOptionObject(IRuntimePtr pRuntime)
{
    Siemens::Runtime::HmiUnified::Common::Cpp::IOptionPtr pOdkOption;
    //load option component by name
    CFRESULT errorCode = pRuntime->GetOption(CCFString("MyOptionName"), &pOdkOption);

    ICfUnknownPtr pUnknown;
    //create a instance of the option object "MyOptionObject"
    errorCode = pOdkOption->GetObject(CCFString("MyOptionObject"), &pUnknown);

    if (CF_SUCCEEDED(errorCode))
    {
        IMyOptionObjectPtr pMyOptionObject(pUnknown);
        CCFString strProperty;
        pMyOptionObject->GetMyProperty(&strProperty);
    }
    else
    {
        IErrorInfoPtr pInfo;
        if (CF_SUCCEEDED(GetErrorHandler(&pInfo)))
        {
            SCCfString errorDescription;
            //get error description
            pInfo->GetErrorDescription(errorCode, &errorDescriptionStr);
        }
    }
}
```

See also

[IRuntime \(Page 191\)](#)

[IErrorInfo \(Page 206\)](#)

9.2.2 IRuntime

Description

The C++ interface "IRuntime" specifies methods for information and the addressing of the Runtime system.

Members

The following methods are specified in the interface:

"GetObject" method

Create new instance of an object of the Runtime system. Possible object types are defined in the configuration file OdkObjectModel.xml.

```
CFRESULT GetObject(const CFSTR value, ICfUnknown **ppObject)
```

- value
[in]: Name of the object type, for example "Tag" for tags
- ppObject
[out]: Points to the initialized object of the runtime system.

"GetProduct" method

Return an "IProduct" object that allows access to the version information and installed options of the Runtime system.

```
CFRESULT GetProduct(IProduct **ppProduct)
```

ppProduct

[in/out]: Points to an "IProduct" object that contains the product information of the runtime system.

"GetOption" method

Referencing installed option of the Runtime system.

```
CFRESULT GetOption(  
    const CFSTR optionName,  
    IOption **ppOption)
```

- optionName
[in]: Name of the installed option
- ppProduct
[out]: Points to an installed option of the Runtime system as "IOption" object.

"GetUserName" method

Return the name of the logged-on user.

```
CFRESULT GetUserName(CFSTR* name)
```

name

[out]: Displays the user name.

Example

Initialize an object of the "Tag" type of the Runtime system:

Copy code

```
CFRESULT ReadSingleTagSync(IRuntimePtr pRuntime, CCfString tag)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCfString(L"Tag"), &pUnk);

    if(pUnk != nullptr && CF_SUCCEEDED(errCode))
    {
        ITagPtr pTag(pUnk);
        pTag->SetTagName(tag);

        ... //further tag processing
    }

    return errCode;
}
```

Output technical product version of the Runtime system:

Copy code

```
void GetVersionInfo(IRuntimePtr pRuntime)
{
    IProductPtr pProduct;
    CFRESULT errCode = pRuntime->GetProduct(&pProduct);

    if(pProduct != NULL && CF_SUCCEEDED(errCode))
    {
        uint16_t uintMajor, uintMinor, uintUpdate, uintServicePack;
        IVersionInfoPtr pVersion;
        pProduct->GetVersion(&pVersion);

        pVersion->GetMajor(&uintMajor);
        pVersion->GetMinor(&uintMinor);
        pVersion->GetServicePack(&uintServicePack);
        pVersion->GetUpdate(&uintUpdate);

        wcout << L"WinCC Unified version: " << uintMajor << L"-" << uintMinor << L"-" <<
        uintServicePack << L"-" << uintUpdate << endl;
    }
}
```

Use installed options:

Copy code

```
void GetOptionObject(IRuntimePtr pRuntime)
{
    Siemens::Runtime::HmiUnified::Common::Cpp::IOptionPtr pOdkOption;
    CFRESULT errCode = pRuntime->GetOption(CCfString("MyOptionName"), &pOdkOption);

    ICfUnknownPtr pUnk;
    errCode = pOdkOption->GetObject("MyOptionObject2", &pUnk);

    if (CF_SUCCEEDED(errCode))
    {
        IMyOptionObjectPtr pMyOptionObject(pUnknown);
        CCfString strProperty;
        pMyOptionObject->GetMyProperty(&strProperty);
    }
    else
    {
        IErrorInfoPtr pInfo;
        if (CF_SUCCEEDED(GetErrorHandler(&pInfo)))
        {
            CCfString errorDescriptionStr;
            //get error description
            pInfo->GetErrorDescription(errCode, &errorDescriptionStr);
        }
    }

    //using extension methods for CPM node
    ICpmPtr pCpm;
    errCode = pRuntime->GetObject(CCfString("Cpm"), (ICfUnknown**) &pCpm);

    ICpmNodePtr pCpmNode;
    CCfString strNode(".hierarchy::PlantView\\Unit1");
    errCode = pCpm->GetNode(strNode, &pCpmNode);

    //using specific option interface
    IMyOptionPtr pMyOption(pOdkOption);

    IMyCpmNodeFormulaPtr pFormula;
    pMyOption->GetObject(pCpmNode, CCfString("Formula"), (ICfUnknown**) &pFormula);
    pFormula->SetName(CCfString("Quality"));
    int32_t result;
    pFormula->Calc(&result);
}
```

See also

[IOdkRt \(Page 189\)](#)

[IProduct \(Page 195\)](#)

[IOption \(Page 196\)](#)

9.2.3 IProduct

Description

The C++ interface "IProduct" specifies methods for handling product information of the Runtime system.

Members

The following methods are specified in the interface:

"GetOptions" method

Return installed options of the Runtime system as array of "IOption" objects.

```
CFRESULT GetOptions(IOptionEnumerator **ppEnumerator)
```

```
ppEnumerator
```

[out]: Points to the installed options as "IOptionEnumerator" object.

"GetVersion" method

Return version structure of the installed Runtime system as "IVersionInfo" object.

```
CFRESULT GetVersion(IVersionInfo** versionInfo)
```

```
versionInfo
```

[out]: Points to a structure with version information of the installed Runtime system.

Example

Output technical product version of the Runtime system:

Copy code

```
void GetVersionInfo(IRuntimePtr pRuntime)
{
    IProductPtr pProduct;
    CFRESULT errCode = pRuntime->GetProduct(&pProduct);

    if(pProduct != NULL && CF_SUCCEEDED(errCode))
    {
        uint16_t uintMajor, uintMinor, uintUpdate, uintServicePack;
        IVersionInfoPtr pVersion;
        pProduct->GetVersion(&pVersion);

        pVersion->GetMajor(&uintMajor);
        pVersion->GetMinor(&uintMinor);
        pVersion->GetServicePack(&uintServicePack);
        pVersion->GetUpdate(&uintUpdate);

        wcout << L"WinCC Unified version: " << uintMajor << L"-" << uintMinor << L"-" <<
        uintServicePack << L"-" << uintUpdate << endl;
    }
}
```

Output name of all installed options:

Copy code

```
void GetVersionInfo(IRuntimePtr pRuntime)
{
    IProductPtr pProduct;
    CFRESULT errCode = pRuntime->GetProduct(&pProduct);
    if (pProduct != NULL && CF_SUCCEEDED(errCode))
    {
        IOptionEnumeratorPtr pItems;
        errCode = pProduct->GetOptions(&pItems);
        if (CF_SUCCEEDED(errCode))
        {
            while (pItems->MoveNext() == CF_SUCCESS)
            {
                IOptionPtr pValue;
                pItems->Current(&pValue);
                CCfString module;
                pValue->GetName(&module);
                wcout << L"Option name: " << module << endl;
            }
        }
        else
        {
            wcout << L"No option installed." << endl;
            PrintErrorInformation(errCode, L"GetOptions", pRuntime);
        }
    }
    else
    {
        PrintErrorInformation(errCode, L"GetProduct", pRuntime);
    }
}
```

See also

[IOdkRt \(Page 189\)](#)
[IRuntime \(Page 191\)](#)
[IOption \(Page 196\)](#)
[IOptionEnumerator \(Page 198\)](#)
[IVersionInfo \(Page 200\)](#)

9.2.4 IOption

Description

The C++ interface "IOption" specifies properties and methods for handling installed product options of the Runtime system.

Members

The following methods are specified in the interface:

"GetName" method

Return name of an installed option of the Runtime system.

```
CFRESULT GetName(CFSTR *pValue)
```

pValue

[out]: Points to the name of an installed option of the runtime system.

"GetObject" method

Referencing installed option of the Runtime system.

```
CFRESULT GetObject(  
    const CFSTR Value,  
    ICfUnknown** ppObject)
```

- Value
[in]: Name of the installed option of the Runtime system
- ppObject
[out]: Points to the installed option of the Runtime system as an "ICfUnknown" object.

"GetVersion" method

Reference version structure of an installed option of the Runtime system as "IVersionInfo" object.

```
CFRESULT GetVersion(IVersionInfo** versionInfo)
```

versionInfo

[out]: Points to a structure with version information of an installed option of the Runtime system.

Example

Read out installed option:

Copy code

```
void GetOptionObject(IRuntimePtr pRuntime)
{
    Siemens::Runtime::HmiUnified::Common::Cpp::IOptionPtr pOdkOption;
    CFRESULT errCode = pRuntime->GetOption(CcFString("MyOptionName"), &pOdkOption);

    ICfUnknownPtr pUnk;
    errCode = pOdkOption->GetObject("MyOptionObject2", &pUnk);

    if (CF_SUCCEEDED(errCode))
    {
        IMyOptionObjectPtr pMyOptionObject(pUnknown);
        CcFString strProperty;
        pMyOptionObject->GetMyProperty(&strProperty);
    }
    else
    {
        IErrorInfoPtr pInfo;
        if (CF_SUCCEEDED(GetErrorHandler(&pInfo)))
        {
            CcFString errorDescriptionStr;
            //get error description
            pInfo->GetErrorDescription(errCode, &errorDescriptionStr);
        }
    }
}
```

See also

[IProduct \(Page 195\)](#)

[IOptionEnumerator \(Page 198\)](#)

[IVersionInfo \(Page 200\)](#)

9.2.5 IOptionEnumerator

Description

The "IOptionEnumerator" interface is a C++ interface that specifies methods for handling the enumeration of installed product options of the Runtime system.

All the methods return CF_SUCCESS in case of successful execution.

Members

The following methods are specified in the interface:

"Current" method

Output the current element of the enumeration of a list.

```
CFRESULT Current(IOption **ppItem)
```

ppItem

[out]: Points to the current "IOption" object as an element of the list.

"MoveNext" method

Go to the next element of the enumeration of a list.

```
CFRESULT MoveNext()
```

"Reset" method

Reset the current position in the enumeration of a list.

```
CFRESULT Reset()
```

The "MoveNext" method subsequently moves to the first element of the list.

"Count" method

Output the size of the enumeration or the number of elements of a list.

```
CFRESULT Count(uint32_t *value)
```

value

[out]: Points to the value for the number of the elements of the list.

Example

Access the installed options "IOption" of the runtime system:

Copy code

```
void GetVersionInfo(IRuntimePtr pRuntime)
{
    IProductPtr pProduct;
    CFRESULT errCode = pRuntime->GetProduct(&pProduct);
    if(pProduct != NULL && CF_SUCCEEDED(errCode))
    {
        IOptionEnumeratorPtr pItems;
        if(CF_SUCCEEDED(pProduct->GetOptions(&pItems)))
        {
            while(pItems->MoveNext() == CF_SUCCESS)
            {
                IOptionPtr pValue;
                pItems->Current(&pValue);
                ...
            }
        }
        else
        {
            wcout << L"No option installed." <<endl;
        }
    }
}
```

See also

IOption (Page 196)

9.2.6 IVersionInfo

Description

The C++ interface "IVersionInfo" specifies methods for reading out version information of the runtime system.

Members

The following methods are specified in the interface:

"GetMajor" method

Return main version of an installed option of the Runtime system.

```
CFRESULT GetMajor(uint16_t *pValue)
```

pValue

[out]: Points to the main version of an installed option of the Runtime system.

"GetMinor" method

Return minor version of an installed option of the Runtime system.

```
CFRESULT GetMinor(uint16_t *pValue)
```

pValue

[out]: Points to the minor version of an installed option of the Runtime system.

"GetServicePack" method

Return service pack of an installed option of the Runtime system.

```
CFRESULT GetServicePack(uint16_t *pValue)
```

pValue

[out]: Points to the service pack of an installed option of the Runtime system.

"GetUpdate" method

Return update version of an installed option of the Runtime system.

```
CFRESULT GetUpdate(uint16_t *pValue)
```

pValue

[out]: Points to the update version of an installed option of the Runtime system.

Example

Output technical product version of the Runtime system:

Copy code

```
void GetVersionInfo(IRuntimePtr pRuntime)
{
    IProductPtr pProduct;
    CFRESULT errCode = pRuntime->GetProduct(&pProduct);

    if(pProduct != NULL && CF_SUCCEEDED(errCode))
    {
        uint16_t uintMajor, uintMinor, uintUpdate, uintServicePack;
        IVersionInfoPtr pVersion;
        pProduct->GetVersion(&pVersion);

        pVersion->GetMajor(&uintMajor);
        pVersion->GetMinor(&uintMinor);
        pVersion->GetServicePack(&uintServicePack);
        pVersion->GetUpdate(&uintUpdate);

        wcout << L"WinCC Unified version: " << uintMajor << L"-" << uintMinor << L"-" <<
        uintServicePack << L"-" << uintUpdate << endl;
    }
}
```

See also

IProduct (Page 195)

IOption (Page 196)

9.2.7 IErrorResult

Description

The "IErrorResult" interface is a C++ interface that specifies methods for reading out error details.

Members

The following methods are specified in the interface:

"GetError" method

Read out error code of an error message.

```
CFRESULT GetError(CFRESULT *value)
```

value

[out]: Points to an error code.

"GetName" method

Read out name of the associated object of the data source.

```
CFRESULT GetError(CFSTR *value)
```

value

[out]: Points to an object name.

Example

Read out details of "IErrorResult" error messages:

Copy code

```
IErrorResultEnumerator* WriteTagSetSync(IRuntimePtr pRuntime, std::vector<TagTuple_T> tags)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCcString(L"TagSet"), &pUnk);
    IErrorResultEnumerator* pEnumerator = nullptr;
    if (pUnk != nullptr && CF_SUCCEEDED(errCode))
    {
        ITagSetPtr pTagSet(pUnk);

        // add tags to tag set
        for (int i = 0; i < tags.size(); i++)
        {
            pTagSet->AddWithValue(CCcString(tags[i]._tagName), tags[i]._tagValue);
        }

        errCode = pTagSet->Write(&pEnumerator);
        if (CF_FAILED(errCode))
        {
            std::wcout << L"Write operation failed." << std::endl;
            PrintErrorInformation(errCode, L"Write", pRuntime);
        }

        if (pEnumerator != nullptr)
        {
            while (pEnumerator->MoveNext() == CF_SUCCESS)
            {
                IErrorResult* pValue;
                CFRESULT errorCode = pEnumerator->Current(&pValue);
                if (pValue != nullptr && CF_SUCCEEDED(errorCode))
                {
                    pValue->GetError(&errorCode);
                    CCcString str;
                    pValue->GetName(&str);

                    if (CF_FAILED(errorCode))
                    {
                        std::wcout << L"Write Tag failed, Tag name: " << str << L", ErrorCode: " << errorCode << std::endl;
                        PrintErrorInformation(errorCode, L"Write Tag", pRuntime);
                    }
                }
            }
        }
        else
        {
            std::wcout << L"Error, couldn't create ODK object." << std::endl;
            PrintErrorInformation(errCode, L"GetObject", pRuntime);
        }
        return pEnumerator;
    }
}
```

See also

[IErrorResultEnumerator](#) (Page 204)

9.2.8 IErrorResultEnumerator

Description

The "IErrorResultEnumerator" interface is a C++ interface that specifies methods for handling the enumeration of error messages of the Runtime system.

All the methods return CF_SUCCESS following successful execution.

Members

The following methods are specified in the interface:

"Current" method

Output the current element of the enumeration of a list.

```
CFRESULT Current(IErrorResult **ppItem)
```

ppItem

[out]: Points to the current "IErrorResult" object as an element of the list.

"MoveNext" method

Go to the next element of the enumeration of a list.

```
CFRESULT MoveNext()
```

"Reset" method

Reset the current position in the enumeration of a list.

```
CFRESULT Reset()
```

The "MoveNext" method moves afterwards to the first element of the list.

"Count" method

Output the size of the enumeration or the number of elements of a list.

```
CFRESULT Count(uint32_t *value)
```

value

[out]: Points to the value for the number of elements of the list.

Example

Access the "IErrorResult" error messages when writing a TagSet:

Copy code

```
IErrorResultEnumerator* WriteTagSetSync(IRuntimePtr pRuntime, std::vector<TagTuple_T> tags)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCcString(L"TagSet"), &pUnk);
    IErrorResultEnumerator* pEnumerator = nullptr;
    if (pUnk != nullptr && CF_SUCCEEDED(errCode))
    {
        ITagSetPtr pTagSet(pUnk);

        // add tags to tag set
        for (int i = 0; i < tags.size(); i++)
        {
            pTagSet->AddWithValue(CCcString(tags[i]._tagName), tags[i]._tagValue);
        }

        errCode = pTagSet->Write(&pEnumerator);
        if (CF_FAILED(errCode))
        {
            std::wcout << L"Write operation failed." << std::endl;
            PrintErrorInformation(errCode, L"Write", pRuntime);
        }

        if (pEnumerator != nullptr)
        {
            while (pEnumerator->MoveNext() == CF_SUCCESS)
            {
                IErrorResult* pValue;
                CFRESULT errorCode = pEnumerator->Current(&pValue);
                if (pValue != nullptr && CF_SUCCEEDED(errorCode))
                {
                    pValue->GetError(&errorCode);
                    CCcString str;
                    pValue->GetName(&str);

                    if (CF_FAILED(errorCode))
                    {
                        std::wcout << L"Write Tag failed, Tag name: " << str << L", ErrorCode: " << errorCode << std::endl;
                        PrintErrorInformation(errorCode, L"Write Tag", pRuntime);
                    }
                }
            }
        }
        else
        {
            std::wcout << L"Error, couldn't create ODK object." << std::endl;
            PrintErrorInformation(errCode, L"GetObject", pRuntime);
        }
        return pEnumerator;
    }
}
```

See also

[IErrorResult](#) (Page 201)

[ITagSet](#) (Page 221)

[ITagSetQCD](#) (Page 229)

9.2.9 IErrorInfo

Description

The "IErrorInfo" interface is a C++ interface that specifies methods for handling error codes.

Members

The following methods are specified in the interface:

"GetErrorDescription" method

Output an error description for the error code.

You have to use the "GetErrorHandler" method to instantiate an "IErrorInfo" object beforehand.

```
CFRESULT GetErrorDescription(  
    uint32_t errorCode,  
    CFSTR *errorDescription)
```

- `errorCode`
[in]: Error code that is handed over by the ODK client.
- `errorDescription`
[out]: Points to the error description of the error code.

Example

Output object name and description of an error:

Copy code

```
void PrintErrorInformation(CFRESULT errorCode, CCfSmartString objectName, IRuntimePtr
pRuntime)
{
    if (pRuntime != nullptr)
    {
        IErrorInfoPtr pInfo;
        CFRESULT result = pRuntime->GetObject(CCfString(L"ErrorHandler"),
(ICfUnknown**) &pInfo);

        if (CF_FAILED(result))
        {
            std::wcout << "Error occurred: Can not create 'ErrorHandler' object " <<
std::endl;
            return;
        }

        CCfString resStr;
        result = pInfo->GetErrorDescription(errorCode, &resStr);

        if (CF_SUCCEEDED(result))
        {
            CCfSmartString errorDescription(resStr);
            std::wcout << "Error occurred: '" << errorDescription.Get() << "', ObjectName =
" << objectName.Get() << std::endl;
        }
        else
        {
            std::wcout << "Error occurred: 'GetErrorDescription' failed, Error number: " <<
result << std::endl;
        }
    }
    else
    {
        std::wcout << "IRuntimePtr is NULL " << std::endl;
    }
}
```

See also

[IOdkRt \(Page 189\)](#)

9.3 Interfaces of the tags

9.3.1 IProcessValue

Description

The C++ interface "IProcessValue" specifies properties and methods for values of process tags of the Runtime system. The "IProcessValue" interface represents values from the result of a read operation or monitoring.

Members

The following methods are specified in the interface:

"GetTagName" method

Return the name of the tag.

```
CFRESULT GetTagName (CFSTR *value)
```

value

[out]: Points to the name of the tag belonging to the process value.

"GetValue" method

Return value of the tag at the moment of the read operation.

```
CFRESULT GetValue (CFVARIANT *value)
```

value

[out]: Points to the process value of the tag.

"GetQuality" method

Return quality code of the read operation of the tag.

```
CFRESULT GetQuality (int32_t *value)
```

value

[out]: Points to the quality code of the process tag.

"GetTimeStamp" method

Return the time stamp of the last successful read operation of the tag.

```
CFRESULT GetTimeStamp (CFDATETIME64 *value)
```

value

[out]: Points to the time stamp of the read operation of the process tag.

"GetError" method

Return error code of the last read or write operation of the tag.

```
CFRESULT GetError (int32_t *value)
```


value

[out]: Points to the error code of the process tag.

Example

Read out a process tag and output the properties of the "IProcessValue" object:

Copy code

```
CFRESULT ReadSingleTagSync(IRuntimePtr pRuntime, CCfString tag)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCfString(L"Tag"), &pUnk);

    if (pUnk != nullptr && CF_SUCCEEDED(errCode))
    {
        ITagPtr pTag(pUnk);
        pTag->SetTagName(tag);

        IProcessValuePtr pValue;

        // Read value of tag
        errCode = pTag->Read(&pValue);

        if (pValue != nullptr && CF_SUCCEEDED(errCode))
        {
            CCfString timeStamp;
            CFDATE64 cfTimeStamp;
            pValue->GetTimeStamp(&cfTimeStamp);
            CCfDateTime64 time(cfTimeStamp);
            timeStamp = time.GetDateTimeString(false);

            CCfString strName;
            pValue->GetTagName(&strName);

            CCfVariant varValue;
            pValue->GetValue(&varValue);

            std::wcout << strName.ToUTF8().c_str() << L" " << timeStamp << L" " << L" Value:
" << (double)(varValue) << std::endl;
        }
        else
        {
            std::wcout << L"Read operation failed." << std::endl;
        }
    }
    else
    {
        std::wcout << L"Error, couldn't create ODK object." << std::endl;
    }
    return errCode;
}
```

See also

IProcessValueEnumerator (Page 210)

ITag (Page 212)

ITagSet (Page 221)

ITagSetQCD (Page 229)

9.3.2 IProcessValueEnumerator

Description

The "IProcessValueEnumerator" interface is a C++ interface that specifies methods for handling the enumeration of process values of the Runtime system. The enumeration is, for example, used when reading out process values of a TagSet.

All the methods return CF_SUCCESS in case of successful execution.

Members

The following methods are specified in the interface:

"Current" method

Output the current element of the enumeration of a list.

```
CFRESULT Current(IProcessValue **ppItem)
```

ppItem

[out]: Points to the current "IProcessValue" object as an element of the list.

"MoveNext" method

Go to the next element of the enumeration of a list.

```
CFRESULT MoveNext()
```

"Reset" method

Reset the current position in the enumeration of a list.

```
CFRESULT Reset()
```

The "MoveNext" method subsequently moves to the first element of the list.

"Count" method

Output the size of the enumeration or the number of elements of a list.

```
CFRESULT Count(uint32_t *value)
```

value

[out]: Points to the value for the number of the elements of the list.

Example

Output process values of TagSets:

Copy code

```

...
IProcessValueEnumeratorPtr pItem;
errCode = pItem->Read(&pItem);
if(pItem != nullptr && CF_SUCCEEDED(errCode))
{
    std::wcout << "Read finished " << std::endl;
    // Iterate over the process value objects
    while(CF_SUCCEEDED(pItem->MoveNext()))
    {
        IProcessValuePtr pValue;
        errCode = pItem->Current(&pValue);    // get current process value
        if(pValue != nullptr && CF_SUCCEEDED(errCode))
        {
            CCfString timeStamp;
            CFDATE64 cfTimeStamp;
            pValue->GetTimeStamp(&cfTimeStamp);
            CCfDate64 time(cfTimeStamp);
            timeStamp = time.GetDateTimeString(false);
            CCfString strName;

            pValue->GetTagName(&strName);
            CCfVariant varValue;
            pValue->GetValue(&varValue);
            int32_t quality;
            pValue->GetQuality(&quality);
            int32_t error = 0;
            pValue->GetError(&error);
            std::wcout << L" " << strName.ToUTF8().c_str() << L" " <<
timeStamp.ToUTF8().c_str() << L" " << L" Value: " << static_cast<double>(varValue) << L"
Quality: " << quality << L" Error: " << (uint32_t)error << std::endl;
        }
    }
}
else
{
    PrintErrorInformation(errCode, L"Read", pRuntime);
}
...

```

See also

IProcessValue (Page 208)

9.3.3 ITag

Description

The C++ interface "ITag" specifies methods for handling tags of the Runtime system.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

The following methods are specified in the interface:

"SetTagName" method

Set name of the tag.

```
CFRESULT SetTagName(const CFSTR tagName)
```

tagName

[in]: Name of the tag

"Write" method

Write process value of the tag synchronously in the Runtime system.

```
CFRESULT Write(  
    const CFVARIANT value,  
    CFENUM type = HmiWriteType::NoWait)
```

- value
[in]: Tag value
- type
[in/optional]: The enumeration "HmiWriteType" specifies whether the method waits for the write operation to be completed:
 - HmiWriteType::NoWait (default): Writes the tag value without waiting. Errors for the write operation are not detected.
 - HmiWriteType::Wait: Waits until the tag value is written in the AS. The associated errors are written.

"WriteQCD" method

Write process value with quality code of the tag synchronously in the Runtime system. The tag also has a freely definable time stamp. You can use this to acquire past external measured values, for example.

Note

Reaction to external tags

For external tags, the method only writes the tag value. The QualityCode and time stamp are set internally by the system.

```
CFRESULT WriteQCD(
    const CFVARIANT value,
    const CFDATEETIME64 timeStamp,
    const int16_t qualityCode,
    CFENUM type = HmiWriteType::NoWait)
```

- value
[in]: Tag value
- timeStamp
[in]: Time stamp of the tag. Also in the past.
- qualityCode
[in]: Quality code of the tag
- type
[in/optional]: The enumeration "HmiWriteType" specifies whether the method waits for the write operation to be completed:
 - HmiWriteType::NoWait (default): Writes the tag value without waiting. Errors for the write operation are not detected.
 - HmiWriteType::Wait: Waits until the tag value is written in the AS. The associated errors are written.

"WriteWithOperatorMessage" method

Write process value of the tag synchronously in the Runtime system and create operator message. In addition to the reason, the operator message contains the old and new value, the user and host names and the unit.

```
CFRESULT WriteWithOperatorMessage(
    const CFVARIANT value,
    const CFSTR reason)
```

- value
[in]: Value of the tag
- reason
[in]: Reason of the value change for message

"Read" method

Read process value and properties of the tag synchronously from the Runtime system.

```
CFRESULT Read(
    IProcessValue **ppValue,
    CFENUM type = HmiReadType::Cache)
```

- ppValue
[out]: Points to the properties and the value of the tag as an "IProcessValue" object.
- type
[in/optional]: The enumeration "HmiReadType" specifies the origin of the tag value:
 - HmiReadType::Cache (default): Reads the tag value from the tag image. If no registration exists, the tag is registered.
 - HmiReadType::Device: Reads the tag value directly from the AS. The tag image is not used.

Example

Write tags synchronously:

Copy code

```
CFRESULT WriteSingleTagSync(IRuntimePtr pRuntime, CCfString tag, CCfVariant& value)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCfString(L"Tag"), &pUnk);
    if (pUnk != nullptr && CF_SUCCEEDED(errCode))
    {
        ITagPtr pTag(pUnk);
        pTag->SetTagName(tag);

        // Write value of tag
        errCode = pTag->Write(value);
        if(CF_FAILED(errCode))
        {
            std::wcout << L"Write operation failed." << std::endl;
            PrintErrorInformation(errCode, L"Write", pRuntime);
        }
    }
    else
    {
        std::wcout << L"Error, couldn't create ODK object." << std::endl;
        PrintErrorInformation(errCode, L"GetObject", pRuntime);
    }
    return errCode;
}
```

Write tag with time stamp and quality code synchronously:

Copy code

```
void WriteSingleTagQCDSync(IRuntimePtr pRuntime, CCfString tag, CCfVariant& value)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCfString(L"Tag"), &pUnk);

    if(pUnk != nullptr && CF_SUCCEEDED(errCode))
    {
        ITagPtr pTag(pUnk);
        pTag->SetTagName(tag);

        // Write value of tag
        errCode = pTag->WriteQCD(value, CCfDateTime64::Now(), 128);
        if(CF_FAILED(errCode))
        {
            std::wcout << L"Write operation failed." << std::endl;
            PrintErrorInformation(errCode, L"Write", pRuntime);
        }
    }
    else
    {
        std::wcout << L"Error, couldn't create ODK object." << std::endl;
        PrintErrorInformation(errCode, L"GetObject", pRuntime);
    }
}
```

Read tags synchronously:

Copy code

```
CFRESULT ReadSingleTagSync(IRuntimePtr pRuntime, CCfString tag)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCfString(L"Tag"), &pUnk);

    if(pUnk != nullptr && CF_SUCCEEDED(errCode))
    {
        ITagPtr pTag(pUnk);
        pTag->SetTagName(tag);

        IProcessValuePtr pValue;

        // Read value of tag
        errCode = pTag->Read(&pValue);

        if(pValue != nullptr && CF_SUCCEEDED(errCode))
        {
            ...
        }
        else
        {
            std::wcout << L"Read operation failed." << std::endl;
        }
    }
    else
    {
        std::wcout << L"Error, couldn't create ODK object." << std::endl;
    }
    return errCode;
}
```

See also

[IProcessValue](#) (Page 208)

[ITagCallback](#) (Page 216)

9.3.4 ITagCallback

Description

The "ITagCallback" interface and the "COdkTagSourceCBBBase" and "COdkTagSetCB" classes define methods for implementing asynchronous read and write operations with tags. The methods are used by the C++-interface "ITagSet".

Members of the interface

The following methods are specified in the "ITagCallback" interface:

"OnReadComplete" method

Callback method is called on completion of asynchronous read operations.

The "OnReadComplete" callback method is called when the "ITagSet.ReadAsync" method is used.

```
CFRESULT OnReadComplete(
    IProcessValueEnumerator *pEnumerator,
    uint32_t errorCode,
    int32_t contextId,
    CFBOOL completed)
```

- `pEnumerator`
[out]: Points to an "IProcessValueEnumerator" object that contains the enumeration of the read process values.
- `systemError`
[out]: Error code for the asynchronous operation
- `contextId`
[out]: ContextID as additional identification feature of the tag.
- `completed`
[out]: Status of the asynchronous transfer:
 - True: All alarms are read out.
 - False: Not all alarms are yet read out.

"OnWriteComplete" method

Callback method is called on completion of asynchronous write operations.

The "OnWriteComplete" callback method is called when the "ITagSet.WriteAsync" method is used.

```
CFRESULT OnWriteComplete(
    IErrorResultEnumerator *pEnumerator,
    uint32_t errorCode,
    int32_t contextId,
    CFBOOL completed)
```

- `pEnumerator`
[out]: Points to an "IErrorResultEnumerator" object that contains the enumeration with errors for the write operations of the tag.
- `systemError`
[out]: Error code for the asynchronous operation
- `contextId`
[out]: ContextID as additional identification feature of the tag.
- `completed`
[out]: Status of the asynchronous transfer:
 - True: All alarms are read out.
 - False: Not all alarms are yet read out.

"OnDataChanged" method

Callback method is called when a monitored tag value is changed.

The callback method is called after the process value change of a monitored TagSet ("ITagSet.Subscribe" method).

```
CFRESULT OnDataChanged(
    IProcessValueEnumerator *pEnumerator,
    uint32_t errorCode,
    int32_t contextId)
```

- pEnumerator
[out]: Points to an "IProcessValueEnumerator" object that contains the enumeration of the read process values.
- systemError
[out]: Error code for the asynchronous operation
- contextId
[out]: ContextID as additional identification feature of the tag.

Members of the classes

The following methods are implemented in the "COdkTagSourceCBBBase" and "COdkTagSetCB" classes:

"SetEvent" method

Signals an event.

```
CFBOOL SetEvent()
```

"ResetEvent" method

Resets the signaling of an event.

```
CFBOOL ResetEvent()
```

"WaitForcompletion" method

Waits for the signaling of an event.

```
uint32_t WaitForcompletion(uint32_t dwMilliseconds)
```

```
dwMilliseconds
```

[in]: Time interval in milliseconds for which an event is waited for.

"GetValues" method

Return process values of the asynchronous read operation.

```
std::vector<IProcessValue*> GetValues()
```

Example

In the following section tags in a TagSet are read asynchronously. To this purpose the "ReadTagSetAsync" function uses a "COdkTagSetCB" object that implements the "ITagCallback" interface and that uses the "COdkTagSourceCBBase" class. The service life of the "COdkTagSetCB" object is determined via reference counting.

9.3 Interfaces of the tags

Copy code

```

void ReadTagSetAsync(IRuntimePtr pRuntime, std::vector<CCfString> tags)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCcString(L"TagSet"), &pUnk);

    if(pUnk != nullptr && CF_SUCCEEDED(errCode))
    {
        ITagSetPtr pTagSet(pUnk);

        // add tags to tag set
        for(int i = 0; i < tags.size(); i++)
        {
            pTagSet->Add(tags[i]);
        }

        COdkTagSetCB* pTagSetCB = new COdkTagSetCB();

        if(pTagSetCB != nullptr)
        {
            pTagSetCB->AddRef();

            // Read the tag set asynchronously, result comes via callback
            if(CF_SUCCEEDED(pTagSet->ReadAsync(pTagSetCB))
            {
                if (CF_SUCCESS == pTagSetCB-
>WaitForCompletion(std::numeric_limits<uint32_t>::max()))
                {
                    vector<IProcessValuePtr> pValues = pTagSetCB->GetValues();

                    std::wcout << L"Read finished " << std::endl;

                    // display tag values
                    for(int i = 0; i < pValues.size(); i++)
                    {
                        IProcessValue* pValue = pValues[i];
                        CCfString timeStamp;
                        CFDATE_TIME64 cfTimeStamp;
                        pValue->GetTimeStamp(&cfTimeStamp);
                        CCfDateTime64 time(cfTimeStamp);
                        timeStamp = time.GetDateTimeString(false);
                        CCfString strName;
                        pValue->GetTagName(&strName);
                        CCfVariant varValue;
                        pValue->GetValue(&varValue);

                        std::wcout << strName << L" " << timeStamp << L" " << L" Value: " <<
(double)(varValue) << std::endl;
                    }
                }
            }
            else
            {
                std::wcout << L"Error, couldn't create callback interface." << std::endl;
                PrintErrorInformation(errCode, L"WaitForCompletion", pRuntime);
            }
        }
    }
}

```

Copy code

```

        else
        {
            std::wcout << L"ReadAsync request failed." << std::endl;
            PrintErrorInformation(errCode, L"ReadAsync", pRuntime);
        }
    }
    else
    {
        std::wcout << L"General error" << std::endl;
        PrintErrorInformation(errCode, L"CodkTagSetCB", pRuntime);
    }
}
else
{
    std::wcout << L"Error, couldn't create ODK object." << std::endl;
    PrintErrorInformation(errCode, L"GetObject", pRuntime);
}
}

```

See also

[ITag \(Page 212\)](#)

[ITagSet \(Page 221\)](#)

[ITagSetQCD \(Page 229\)](#)

9.3.5 ITagSet**Description**

The C++ interface "ITagSet" specifies properties and methods for an optimized access to several tags of the Runtime system.

After initialization of the "ITagSet" object, you can execute read and write access to multiple tags in one call. Simultaneous access demonstrates better performance and lower communication load than single access to multiple tags.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

The following methods are specified in the interface:

"SetContextId" method

Set ID as an additional identification feature of the tag. The ContextId can, for example, be used to recognize identically named tags from different monitoring functions. Default value -1: The ContextId is not used.

```
CFRESULT SetContextId(const int32_t value)
```

value

[in]: ContextId of the tag:

"GetContextId" method

Set ID as an additional identification feature of the tag. The ContextId can, for example, be used to recognize identically named tags from different monitoring functions. Default value -1: The ContextId is not used.

```
CFRESULT GetContextId(int32_t *value)
```

value

[out]: Points to the ContextId of the tag.

"Remove" method

Remove individual tag from a TagSet.

```
CFRESULT Remove(const CFSTR tagName)
```

tagName

[in]: Name of the tag that is removed from TagSet.

"Add" method

Add tag to a TagSet.

```
CFRESULT Add(const CFSTR tagName)
```

tagName

[in]: Name of the tag for TagSet

"AddWithValue" method

Add tag with process value to the TagSet.

```
CFRESULT AddWithValue(const CFSTR tagName, const CFVARIANT value)
```

- tagName
[in]: Name of the tag
- value
[in]: New value of the tag

"GetValue" method

Read process value of a tag of a TagSet.

To fill the local TagSet with process values, a "Read", "ReadAsync" or "AddWithValue" method must be called beforehand.

The values of the "IProcessValue" object are not available until after execution of the methods "Read", "ReadAsync" or "AddWithValue".

```
CFRESULT GetValue(const CFSTR tagName, CFVARIANT *pValue)
```

- tagName
[in]: Name of the tag from the TagSet
- pValue
[out]: Points to the process value of the tag.

"SetValue" method

Change the process value of a tag of a TagSet.

The "SetValue" method changes only the values of the local TagSet. In order to write the changed values into the automation system you must additionally execute the "Write" or "WriteAsync" method.

```
CFRESULT SetValue(const CFSTR tagName, const CFVARIANT value)
```

- `tagName`
[in]: Name of the tag from the TagSet
- `value`
[in]: New value of the tag

"Write" method

Write process values of all tags of a TagSet synchronously in the Runtime system.

```
CFRESULT Write(HmiUnified::Rt::IErrorResultEnumerator  
**ppEnumerator, CFENUM type = HmiWriteType::NoWait)
```

- `ppEnumerator`
[out]: Points to an "IErrorResultEnumerator" object that contains the enumeration with errors for the write operations.
- `type`
[in/optional]: The enumeration "HmiWriteType" specifies whether the method waits for the write operation to be completed:
 - `NoWait` (default): Writes the tag values without waiting. Errors for the write operation are not detected.
 - `Wait`: Waits until the tag values are written in the automation system. The associated errors are written.

"WriteWithOperatorMessage" method

Write process values of all tags of a TagSet synchronously in the Runtime system and create operator messages. In addition to the reason, the operation messages contain the old and new value, the user and host names and the unit.

```
CFRESULT WriteWithOperatorMessage(const CFSTR reason)
```

```
reason  
[in]: Reason of the value change for message
```

"WriteAsync" method

Write process values of all tags of a TagSet asynchronously in the Runtime system.

The method always has the `HmiWriteType::Wait` type and waits until the tag value has been written in the automation system. Associated errors are written.

```
CFRESULT WriteAsync(  
    ITagCallback* pTagSetCb)
```

- `pTagSetCb`
[in]: Points to the "ITagCallback" object that implements the callback interface.

"Read" method

Read process values and properties of all the tags of a TagSet synchronously from the Runtime system.

```
CFRESULT Read(
    IProcessValueEnumerator **ppEnumerator,
    CFENUM type = HmiReadType::Cache)
```

- ppEnumerator
[in/out]: Points to the properties and process values of the tags as an "IProcessValueEnumerator" object.
- type
[in/optional]: The enumeration "HmiReadType" specifies the origin of the tag value:
 - Cache (default): Reads the tag values from the tag image. If no subscription exists, the tag is subscribed.
 - Device: Reads the tag values directly from the automation system. The tag image is not used.

"ReadAsync" method

Read process values and properties of all the tags of a TagSet asynchronously from the Runtime system.

```
CFRESULT ReadAsync(
    ITagCallback *pTagSetCb,
    CFENUM type = HmiReadType::Cache)
```

- pTagSetCb
[in]: Points to the "ITagCallback" object that implements the callback interface.
- type
[in/optional]: The enumeration "HmiReadType" specifies the origin of the tag value:
 - Cache (default): Reads the tag value from the tag image. If no subscription exists, the tag is subscribed.
 - Device: Reads the tag value directly from the AS. The tag image is not used.

"Subscribe" method

Subscribe all tags of a TagSet asynchronously for cyclic monitoring of the process values.

Note**Tags from IO devices with the "Cyclic in operation" acquisition mode**

For a tag with the acquisition mode "Cyclic in operation", the value stored in the process image when Subscribe is called might be outdated. OnAdd therefore only provides the QualityCode "uncertain". Only value changes made after the Subscribe call provide the current value and the QualityCode "good".

```
CFRESULT Subscribe(ITagCallback *pTagSetCb)
```

pTagCb

[in]: Points to the "ITagCallback" object that implements the callback interface.

"CancelSubscribe" method

Cancel monitoring of all tags of a TagSet.

```
CFRESULT CancelSubscribe()
```

"GetCount" method

Return the number of tags of a TagSet list.

```
CFRESULT GetCount(int32_t *value)
```

value

[out]: Points to the value for the number of tags of the TagSet list.

"Clear" method

Remove all tags from a TagSet.

```
CFRESULT Clear()
```

Example

Write TagSet asynchronously:

Copy code

```

struct TagTuple_T
{
    CCfSmartString _tagName;
    CCfVariant _tagValue;
};

void WriteTagSetAsync(IRuntimePtr pRuntime, std::vector<TagTuple_T> tags)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCfString(L"TagSet"), &pUnk);
    vector<IErrorResultPtr> pErrors;
    if(pUnk != nullptr && CF_SUCCEEDED(errCode))
    {
        ITagSetPtr pTagSet(pUnk);
        // add tags to tag set
        for(int i = 0; i < tags.size(); i++)
        {
            pTagSet->AddWithValue(CCfString(tags[i]._tagName), tags[i]._tagValue);
        }
        CODkTagSetCB* pTagSetCB = new CODkTagSetCB();
        if(pTagSetCB != nullptr)
        {
            pTagSetCB->AddRef();
            // Write value of tag asynchronously
            pTagSet->WriteAsync(pTagSetCB);
            errCode = pTagSetCB->WaitForCompletion(std::numeric_limits<uint32_t>::max());
            if (CF_SUCCESS == errCode)
            {
                pErrors = pTagSetCB->GetErrors();

                for (int i = 0; i < pErrors.size(); i++)
                {
                    IErrorResult* pError = pErrors[i];
                    CFRESULT tagError;
                    pError->GetError(&tagError);
                    CCfString strName;
                    pError->GetName(&strName);
                    if (CF_FAILED(tagError))
                    {
                        PrintErrorInformation(tagError, L"Tag Write", pRuntime);
                    }
                }
            } else
            {
                PrintErrorInformation(errCode, L"WaitForcompletion", pRuntime);
            }
        }
        else
        {
            PrintErrorInformation(errCode, L"CODkTagSetCB", pRuntime);
        }
    }
    else
    {

```

9.3 Interfaces of the tags

Copy code

```

        PrintErrorInformation(errCode, L"GetObject", pRuntime);
    }
}

```

Start monitoring for tags of a TagSet:

Copy code

```

void SubscribeTagSet(IRuntimePtr pRuntime, std::vector<CCfString> tags)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCfString(L"TagSet"), &pUnk);

    if (pUnk != nullptr && CF_SUCCEEDED(errCode))
    {
        ITagSetPtr pTagSet(pUnk);

        // add tags to tag set
        for (int i = 0; i < tags.size(); i++)
        {
            pTagSet->Add(tags[i]);
        }

        COdkTagSetCB* pTagSetCB = new COdkTagSetCB();

        if (pTagSetCB != nullptr && CF_SUCCEEDED(errCode))
        {
            pTagSetCB->AddRef();

            // subscribe tags
            errCode = pTagSet->Subscribe(pTagSetCB);
            if (CF_FAILED(errCode))
            {
                std::wcout << L"Error, couldn't create callback interface." << std::endl;
                PrintErrorInformation(errCode, L"Subscribe", pRuntime);
            }
        }
        else
        {
            std::wcout << L"Error, couldn't create callback interface." << std::endl;
            PrintErrorInformation(errCode, L"COdkTagSetCB", pRuntime);
        }
    }
    else
    {
        std::wcout << L"Error, couldn't create ODK object." << std::endl;
        PrintErrorInformation(errCode, L"GetObject", pRuntime);
    }
}

```

See also

IProcessValue (Page 208)

ITagCallback (Page 216)

IErrorResultEnumerator (Page 204)

9.3.6 ITagSetQCD**Description**

The C++ interface "ITagSetQCD" specifies methods for optimized writing of several tags of the Runtime system. The tags also have a freely definable time stamp and quality code. You can use this to acquire past external measured values, for example.

Note**Reaction to external tags**

For external tags, the method only writes the tag value. The QualityCode and time stamp are set internally by the system.

After initialization of the "ITagSetQCD" object, you can have read access to multiple tags in one call. Simultaneous access demonstrates better performance and lower communication load than single access to multiple tags.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

The following methods are specified in the interface:

"SetContextId" method

Set ID as an additional identification feature of the tag. The ContextId can, for example, be used to recognize identically named tags from different monitoring functions. Default value -1: The ContextId is not used.

```
CFRESULT SetContextId(const int32_t value)
```

value

[in]: ContextId of the tag:

"GetContextId" method

Set ID as an additional identification feature of the tag. The ContextId can, for example, be used to recognize identically named tags from different monitoring functions. Default value -1: The ContextId is not used.

```
CFRESULT GetContextId(int32_t *value)
```

value

[out]: Points to the ContextId of the tag.

"Remove" method

Remove individual tag from a TagSet.

```
CFRESULT Remove(const CFSTR tagName)
```

tagName

[in]: Name of the tag that is removed from TagSet.

"Add" method

Add tag with process value, quality code and time stamp to the TagSet.

```
CFRESULT Add(
    const CFSTR tagName,
    const CFVARIANT value,
    const CFDATE64 timeStamp,
    const int16_t qualityCode)
```

- tagName
[in]: Name of the tag for TagSet
- value
[in]: New process value of the tag
- timeStamp
[in]: Time stamp of the process value. Also in the past.
- qualityCode
[in]: Quality code for process value

"Write" method

Write process values of all tags of a TagSet synchronously in the Runtime system.

```
CFRESULT Write(HmiUnified::Rt::IErrorResultEnumerator
**ppEnumerator, CFENUM type = HmiWriteType::NoWait)
```

- ppEnumerator
[out]: Points to an "IErrorResultEnumerator" object that contains the enumeration with errors for the write operations.
- type
[in/optional]: The enumeration "HmiWriteType" specifies whether the method waits for the write operation to be completed:
 - HmiWriteType::NoWait (default): Writes the tag values without waiting. Errors for the write operation are not detected.
 - HmiWriteType::Wait: Waits until the tag values are written in the automation system. The associated errors are written.

"WriteAsync" method

Write process values of all tags of a TagSet asynchronously in the Runtime system.

The method always has the HmiWriteType::Wait type and waits until the tag value has been written in the automation system. Associated errors are written.

```
CFRESULT WriteAsync(  
    ITagCallback* pTagSetCb)
```

- pTagSetCb
[in]: Points to the "ITagCallback" object that implements the callback interface.

"GetCount" method

Return the number of tags of a TagSet list.

```
CFRESULT GetCount(int32_t *value)
```

value

[out]: Points to the value for the number of tags of the TagSet list.

"GetItem" method

Return tag of the TagSet for changing or reading out process value, QualityCode and time stamp.

```
CFRESULT GetItem(  
    const CFSTR name,  
    ITagSetQCItem **pTagSetQCItem)
```

- name
[in]: Name of the tag in the TagSet
- pTagSetQCItem
[out]: Points to tag as "TagSetQCItem" object

"Clear" method

Remove all tags from a TagSet.

```
CFRESULT Clear()
```

Example

Write TagSet with time stamp and quality code synchronously:

Copy code

```

struct TagTuple_T
{
    CCfSmartString _tagName;
    CCfVariant _tagValue;
};

void WriteTagSetQCDSync(IRuntimePtr pRuntime, std::vector<TagTuple_T> tags)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCfString(L"TagSetQCD"), &pUnk);
    if(pUnk != nullptr && CF_SUCCEEDED(errCode))
    {
        ITagSetQCDDPtr pTagSetQCD(pUnk);

        // add tags to tag set
        for(int i = 0; i < tags.size(); i++)
        {
            pTagSetQCD->Add(CCfString(tags[i]._tagName), tags[i]._tagValue,
CCfDateTime64::Now(), 128);
        }

        IErrorResultEnumerator* pEnumerator;
        errCode = pTagSetQCD->Write(&pEnumerator);
        if(CF_FAILED(errCode))
        {
            std::wcout << L"Write operation failed." << std::endl;
            PrintErrorInformation(errCode, L"Write", pRuntime);
        }

        if (pEnumerator != nullptr)
        {
            while (pEnumerator->MoveNext() == CF_SUCCESS)
            {
                IErrorResult* pValue;
                CFRESULT errorCode = pEnumerator->Current(&pValue);
                if (pValue != nullptr && CF_SUCCEEDED(errorCode))
                {
                    pValue->GetError(&errorCode);
                    CCfString str;
                    pValue->GetName(&str);
                    if (CF_FAILED(errorCode))
                    {
                        std::wcout << L"Write Tag failed, Tag name: " << str << L", ErrorCode:
" << errorCode << std::endl;
                        PrintErrorInformation(errorCode, L"Write Tag", pRuntime);
                    }
                }
            }
        }
        else
        {
            std::wcout << L"Error, couldn't create ODK object." << std::endl;
            PrintErrorInformation(errCode, L"GetObject", pRuntime);
        }
    }
}

```

9.3 Interfaces of the tags

Copy code

```
}
```

See also

[IProcessValue](#) (Page 208)

[ITagCallback](#) (Page 216)

[ITagSetQCItem](#) (Page 234)

[IErrorResultEnumerator](#) (Page 204)

9.3.7 ITagSetQCItem

Description

The C++ interface "ITagSetQCItem" specifies methods for adapting tags of the Runtime system. You can read in tags into a TagSetQCD and then change all names, values, time stamps and QualityCodes of the tags.

Note

Reaction to external tags

For external tags, only the tag value is set. The QualityCode and time stamp are set internally by the system.

All the methods return CF_SUCCESS following successful execution.

Members

The following methods are specified in the interface:

"GetName" method

Return the name of the tag.

```
CFRESULT GetName(CFSTR *name)
```

name

[out]: Points to the name of the tag.

"SetName" method

Change name of the tag.

```
CFRESULT SetName(const CFSTR name)
```

name

[in]: New name of the tag.

"GetValue" method

Return value of the tag at the moment of the read operation.

```
CFRESULT GetValue(CFVARIANT *value)
```

value

[out]: Points to the process value of the tag.

"SetValue" method

Change value of the tag.

```
CFRESULT SetValue(const CFVARIANT value)
```

value

[in]: New process value of the tag.

"GetTimeStamp" method

Return time stamp of the tag.

```
CFRESULT GetTimeStamp(CFDATE64 *timeStamp)
```

timeStamp

[out]: Points to the time stamp of the tag.

"SetTimeStamp" method

Change time stamp of the tag.

```
CFRESULT SetTimeStamp(const CFDATE64 timeStamp)
```

timeStamp

[in]: New time stamp of the tag

"GetQuality" method

Return quality code of the read operation of the tag.

```
CFRESULT GetQuality(int32_t *qualityCode)
```

qualityCode

[out]: Points to the quality code of the tag.

"SetQuality" method

Change quality code of the tag.

```
CFRESULT SetQuality(const int32_t qualityCode)
```

qualityCode

[in]: New quality code of the tag

See also

[ITagSetQCD \(Page 229\)](#)

9.3.8 ILoggedTagValue

Description

The C++ interface "ILoggedTagValue" specifies the properties for process values of logging tags of a logging system. The "ILoggedTagValue" interface displays historical process values.

Members

The following methods are specified in the interface:

"GetTagName" method

Return name of the logging tag.

```
CFRESULT GetTagName (CFSTR *value)
```

value

[out]: Points to the name of the process value belonging to the logging tag.

"GetValue" method

Return process value of the logging tag.

```
CFRESULT GetValue (CFVARIANT *value)
```

value

[out]: Points to the process value of the logging tag.

"GetQuality" method

Return quality code of the process value.

```
CFRESULT GetQuality (int16_t *value)
```

value

[out]: Points to the quality code of the process value.

"GetTimeStamp" method

Return time stamp of the process value.

```
CFRESULT GetTimeStamp (CFDATETIME64 *value)
```

value

[out]: Points to the time stamp of the process value.

"GetError" method

Return error code of the process value.

```
CFRESULT GetError (uint32_t *value)
```

value

[out]: Points to the error code of the process value.

"GetFlags" method

Return context information from the read operation for the process value.

```
CFRESULT GetFlags(HmiTagLoggingValueFlags *value)
```

```
value
```

[out]: Points to the context information.

The "HmiTagLoggingValueFlags" enumeration contains the following bit-by-bit-coded values:

- 0: Extra
There are still additional values at the time of the process value.
- 2: Calculated
Process value is calculated.
- 16: Bounding
Process value is a limit value.
- 32: NoData
No additional information available
- 64: FirstStored
Process value is the first value stored in the logging system.
- 128: LastStored
Process value is the last value stored in the logging system.

Example

Output process values of a logging tag:

Copy code

```
void PrintValues(ILoggedTagValueEnumeratorPtr pItems)
{
    // Iterate over the process value objects
    while (CF_SUCCEEDED(pItems->MoveNext()))
    {
        ILoggedTagValuePtr pValue;
        CFRESULT errCode = pItems->Current(&pValue); // get current process value
        if (pValue != nullptr && CF_SUCCEEDED(errCode))
        {
            CCfString timeStamp;
            CFDATE_TIME64 cfTimeStamp;
            pValue->GetTimeStamp(&cfTimeStamp);
            CCfDateTime64 time(cfTimeStamp);
            timeStamp = time.GetDateTimeString(false);
            CCfString strName;
            pValue->GetTagName(&strName);
            CCfVariant varValue;
            pValue->GetValue(&varValue);

            std::wcout << strName << L" " << timeStamp << L" " << L" Value: " << (double)
(varValue) << std::endl;
        }
    }
}
```

See also

ILoggedTagValueEnumerator (Page 238)

ILoggedTag (Page 244)

9.3.9 ILoggedTagValueEnumerator

Description

The "ILoggedTagValueEnumerator" interface is a C++ interface that specifies methods for handling the enumeration of process values of a logging tag of the Runtime system. The methods are used by the C++-interfaces "ILoggedTag" and "ILoggedTagSet".

All the methods return CF_SUCCESS in case of successful execution.

Members

The following methods are specified in the interface:

"Current" method

Output the current element of the enumeration of a list.

```
CFRESULT Current(ILoggedTagValue **ppItem)
```

ppItem

[out]: Points to the current "ILoggedTagValue" object as an element of the list.

"MoveNext" method

Go to the next element of the enumeration of a list.

```
CFRESULT MoveNext()
```

"Reset" method

Reset the current position in the enumeration of a list.

```
CFRESULT Reset()
```

The "MoveNext" method subsequently moves to the first element of the list.

"Count" method

Output the size of the enumeration or the number of elements of a list.

```
CFRESULT Count(uint32_t *value)
```

value

[out]: Points to the value for the number of the elements of the list.

Example

Output process values of a logging tag:

Copy code

```

void PrintValues(ILoggedTagValueEnumeratorPtr pItems)
{
    // Iterate over the process value objects
    while (CF_SUCCEEDED(pItems->MoveNext()))
    {
        ILoggedTagValuePtr pValue;
        CFRESULT errCode = pItems->Current(&pValue); // get current process value
        if (pValue != nullptr && CF_SUCCEEDED(errCode))
        {
            ...
        }
    }
}

```

See also

ILoggedTagValue (Page 236)

ILoggedTag (Page 244)

ILoggedTagSet (Page 247)

ILoggedTagCallback / ILoggedTagSetCallback (Page 239)

9.3.10 ILoggedTagCallback / ILoggedTagSetCallback

Description

The interfaces "ILoggedTagCallback" and "ILoggedTagSetCallback" and the classes "COdkTagSourceCBBBase" and "COdkTagSetLoggingCB" define methods for implementing asynchronous read and write operations with logging tags. The methods are used by the C++-interfaces "ILoggedTag" and "ILoggedTagSet".

All the methods return CF_SUCCESS following successful execution.

Members of "ILoggedTagCallback"

The following methods are specified in the interface:

"OnReadComplete" method

Callback method is called on completion of asynchronous read operations.

CFRESULT OnReadComplete(ILoggedTagValueEnumerator *pEnumerator)

pEnumerator

[out]: Points to an "ILoggedTagValueEnumerator" object that contains the enumeration of the process values.

"OnDeleteComplete" method

Callback method is called on completion of asynchronous delete operations.

```
CFRESULT OnDeleteComplete (ILoggedTagValueEnumerator *pEnumerator)
```

pEnumerator

[out]: Points to an "ILoggedTagValueEnumerator" object that contains the enumeration of the process values.

"OnWriteComplete" method

Callback method is called on completion of asynchronous write operations. Can only be applied to individual logging tags in the "ILoggedTagCallback" interface.

```
CFRESULT OnWriteComplete (ILoggedTagValueEnumerator *pEnumerator)
```

pEnumerator

[out]: Points to an "ILoggedTagValueEnumerator" object that contains the enumeration of the process values.

"OnDataChanged" method

Callback method is called when a monitored logging tag is changed.

```
CFRESULT OnDataChanged (ILoggedTagValueEnumerator *pEnumerator)
```

pEnumerator

[out]: Points to an "ILoggedTagValueEnumerator" object that contains the enumeration of the process values.

Members of "ILoggedTagSetCallback"

The following methods are specified in the interface:

"OnReadComplete" method

Callback method is called on completion of asynchronous read operations.

The "OnReadComplete" callback method is called when the "ReadAsync" method is used.

```
CFRESULT OnReadComplete (ILoggedTagValueEnumerator *pEnumerator,
                          uint32_t errorCode,
                          int32_t contextId)
```

- pEnumerator
[out]: Points to an "ILoggedTagValueEnumerator" object that contains the enumeration of the process values.
- errorCode
[out]: Error code for the asynchronous operation
- contextId
[out]: ContextID as additional identification feature of the logging tag.

"OnDataChanged" method

Callback method is called when a monitored logging tag is changed.

The callback method is called after the process value change of a monitored logging tag or a LoggedTagSet.


```
CFRESULT OnDataChanged(ILoggedTagValueEnumerator *pEnumerator,
                      uint32_t errorCode,
                      int32_t contextId)
```

- `pEnumerator`
[out]: Points to an "ILoggedTagValueEnumerator" object that contains the enumeration of the process values.
- `errorCode`
[out]: Error code for the asynchronous operation
- `contextId`
[out]: ContextID as additional identification feature of the logging tag.

Members of the classes

The following methods are implemented in the "COdkTagSourceCBBBase" and "COdkTagSetLoggingCB" classes:

"SetEvent" method

Signals an event.

```
CFBOOL SetEvent()
```

"ResetEvent" method

Resets the signaling of an event.

```
CFBOOL ResetEvent()
```

"WaitForcompletion" method

Waits for the signaling of an event.

```
uint32_t WaitForcompletion(uint32_t dwMilliseconds)
```

```
dwMilliseconds
```

[in]: Time interval in milliseconds for which an event is waited for.

"GetValues" method

Return process values of the asynchronous read operation.

```
std::vector<ILoggedTagValuePtr> GetValues()
```

Example

Output LoggedTagSet asynchronously:

Copy code

```

void LoggingReadTagSetAsync(IRuntimePtr pRuntime, std::vector<CCfString> tags)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCfString(L"LoggedTagSet"), &pUnk);

    if (pUnk != nullptr && CF_SUCCEEDED(errCode))
    {
        ILoggedTagSetPtr pTagSet(pUnk);

        // add tags to tag set
        for (int i = 0; i < tags.size(); i++)
        {
            pTagSet->Add(tags[i]);
        }

        CCfDateTime64 begin, end;
        begin = CCfDateTime64::Now(true);
        end = begin;
        begin.SubtractTimeSpan(Get1Minute() * 30);

        ILoggedTagValueEnumeratorPtr pItems;

        COdkTagSetLoggingCB* pTagSetCB = new COdkTagSetLoggingCB();

        if (pTagSetCB != nullptr)
        {
            pTagSetCB->AddRef();

            // Read the tag set asynchronously, result comes via callback
            if (CF_SUCCEEDED(pTagSet->ReadAsync(pTagSetCB, begin, end, true)))
            {
                if (CF_SUCCESS == pTagSetCB-
>WaitForCompletion(std::numeric_limits<uint32_t>::max()))
                {
                    vector<ILoggedTagValuePtr> pValues = pTagSetCB->GetValues();

                    std::wcout << L"Read finished " << std::endl;

                    // display tag values
                    for (int i = 0; i < pValues.size(); i++)
                    {
                        ILoggedTagValue* pValue = pValues[i];
                        CCfString timeStamp;
                        CFDATE64 cfTimeStamp;
                        pValue->GetTimeStamp(&cfTimeStamp);
                        CCfDateTime64 time(cfTimeStamp);
                        timeStamp = time.GetDateTimeString(false);
                        CCfString strName;
                        pValue->GetTagName(&strName);
                        CCfVariant varValue;
                        pValue->GetValue(&varValue);

                        std::wcout << strName << L" " << timeStamp << L" " << L" Value: " <<
(double)(varValue) << std::endl;
                    }
                }
            }
        }
    }
}

```

Copy code

```

    }
    else
    {
        std::wcout << L"Error, couldn't create callback interface." << std::endl;
        PrintErrorInformation(errCode, L"WaitForcompletion", pRuntime);
    }
}
else
{
    std::wcout << L"ReadAsync request failed." << std::endl;
    PrintErrorInformation(errCode, L"ReadAsync", pRuntime);
}
}
else
{
    std::wcout << L"General error" << std::endl;
    PrintErrorInformation(errCode, L"CODkTagSetCB", pRuntime);
}
}
else
{
    std::wcout << L"Error, couldn't create ODK object." << std::endl;
    PrintErrorInformation(errCode, L"GetObject", pRuntime);
}
}
}

```

See also[ILoggedTag \(Page 244\)](#)[ILoggedTagSet \(Page 247\)](#)[ILoggedTagValueEnumerator \(Page 238\)](#)**9.3.11 ILoggedTag****Description**

The C++ interface "ILoggedTag" specifies methods for handling logging tags of a logging system.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

The following methods are specified in the interface:

"SetTagName" method

Set name of the logging tag.

```
CFRESULT SetTagName(const CFSTR tagName)
```

```
tagName
```

[in]: Name of the logging tag

"Read" method

Read logged process values of a logging tag of a time period synchronously from the logging system.

```
CFRESULT Read(  
    const CFTIME64 begin,  
    const CFTIME64 end,  
    ILoggedTagValueEnumerator **ppEnumerator,  
    CFBOOL boundingValue)
```

- begin
[in]: Start date of the time period
- end
[in]: End date of the time period
- ppEnumerator
[in/out]: Points to the enumeration of process values of the logging tag as "ILoggedTagValueEnumerator" object.
- boundingValue
[in]: True, in order to additionally return high and low limits.

Example

Read out process values of a logging tag synchronously from a logging system:

Copy code

```
void LoggingReadSingleTagSync(IRuntimePtr pRuntime, CCfString tag)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCfString(L"LoggedTag"), &pUnk);

    if (pUnk != nullptr && CF_SUCCEEDED(errCode))
    {
        ILoggedTagPtr pTag(pUnk);
        pTag->SetTagName(tag);

        CCfDateTime64 begin, end;
        begin = CCfDateTime64::Now(true);
        end = begin;
        begin.SubtractTimeSpan(Get1Minute() * 3);

        ILoggedTagValueEnumeratorPtr pItems;
        // Read value of tag
        errCode = pTag->Read(begin, end, &pItems, true);

        if (pItems != nullptr && CF_SUCCEEDED(errCode))
        {
            std::wcout << "Read finished " << std::endl;
            PrintValues(pItems);
        }
        else
        {
            std::wcout << L"Read operation failed." << std::endl;
        }
    }
    else
    {
        std::wcout << L"Error, couldn't create ODK object." << std::endl;
    }
}
```

See also

[ILoggedTagValue \(Page 236\)](#)

[ILoggedTagSet \(Page 247\)](#)

[ILoggedTagCallback / ILoggedTagSetCallback \(Page 239\)](#)

[ILoggedTagValueEnumerator \(Page 238\)](#)

9.3.12 ILoggedTagSet

Description

The C++ interface "ILoggedTagSet" specifies methods for optimized access to several logging tags of a logging system.

After initialization of the "ILoggedTagSet" object, you can execute read and write access to multiple logging tags in one call. Simultaneous access demonstrates better performance and lower communication load than single access to multiple logging tags.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

The following methods are specified in the interface:

"SetContextId" method

ID as additional identification feature of the logging tag. The ContextId can, for example, be used to recognize identically named logging tags from different monitoring functions. Default value -1: The ContextId is not used.

```
CFRESULT SetContextId(const int32_t value)
```

value

[in]: ContextId of the logging tag

"GetContextId" method

ID as additional identification feature of the logging tag. The ContextId can, for example, be used to recognize identically named logging tags from different monitoring functions. Default value -1: The ContextId is not used.

```
CFRESULT GetContextId(int32_t *value)
```

value

[out]: Points to the ContextId of the logging tag.

"Remove" method

Remove individual logging tag from a LoggedTagSet.

```
CFRESULT Remove(const CFSTR tagName)
```

tagName

[in]: Name of the logging tag that is removed from the LoggedTagSet.

"Add" method

Add logging tag to a LoggedTagSet.

```
CFRESULT Add(const CFSTR tagName)
```

tagName

[in]: Name of the logging tag for the LoggedTagSet

"Subscribe" method

Subscribe all logging tags of a LoggedTagSet asynchronously for updating the process values following a change. When new process values are logged, they can be processed with the "OnDataChanged" event.

```
CFRESULT Subscribe (ILoggedTagSetCallback *pTagSetLoggedCb)
```

```
pTagSetLoggedCb
```

[in]: Points to the "ILoggedTagSetCallback" object that implements the callback interface.

"CancelSubscribe" method

Cancel updating of process values following a change for all logging tags of a LoggedTagSet.

```
CFRESULT CancelSubscribe ()
```

"Read" method

Retrieve all logging tags of a LoggedTagSet for a period of time synchronously from the logging system.

```
CFRESULT Read (
    const CFTIME64 begin,
    const CFTIME64 end,
    ILoggedTagValueEnumerator **ppEnumerator,
    CFBOOL boundingValue)
```

- begin
[in]: Start date of the time period
- end
[in]: End date of the time period
- ppEnumerator
[in/out]: Points to the enumeration of process values of the logging tags of the LoggedTagSet as "ILoggedTagValueEnumerator" object.
- boundingValue
[in/optional]: True, in order to additionally return high and low limits.

"ReadAsync" method

Retrieve all logging tags of a LoggedTagSet for a period of time synchronously from the logging system.

```
CFRESULT ReadAsync (
    ILoggedTagSetCallback *pTagLoggedCb,
    const CFTIME64 begin,
    const CFTIME64 end,
    CFBOOL boundingValue)
```

- pTagLoggedCb
[in]: Points to the "ILoggedTagSetCallback" object that implements the callback interface.
- begin
[in]: Start date of the time period

- `end`
[in]: End date of the time period
- `boundingValue`
[in/optional]: True, in order to additionally return high and low limits.

"GetCount" method

Return the number of logging tags of a LoggedTagSet list.

```
CFRESULT GetCount(int32_t *value)
```

`value`

[out]: Points to the value for the number of logging tags of the LoggedTagSet list.

"Clear" method

Remove all logging tags from a LoggedTagSet.

```
CFRESULT Clear()
```

Example

Subscribe logging tags of a LoggedTagSet for change monitoring:

Copy code

```
void LoggingSubscribeTagSet(IRuntimePtr pRuntime, std::vector<CCfString> tags)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCfString(L"LoggedTagSet"), &pUnk);

    if (pUnk != nullptr && CF_SUCCEEDED(errCode))
    {
        ILoggedTagSetPtr pTagSet(pUnk);
        // add tags to tag set
        for (int i = 0; i < tags.size(); i++)
        {
            pTagSet->Add(tags[i]);
        }

        CODkTagSetLoggingCB* pTagSetCB = new CODkTagSetLoggingCB();

        if (pTagSetCB != nullptr && CF_SUCCEEDED(errCode))
        {
            pTagSetCB->AddRef();

            // subscribe tags
            errCode = pTagSet->Subscribe(pTagSetCB);
            if (CF_FAILED(errCode))
            {
                std::wcout << L"Error, couldn't create callback interface." << std::endl;
                PrintErrorInformation(errCode, L"Subscribe", pRuntime);
            }
        }
    }
    else
    {
        std::wcout << L"Error, couldn't create ODK object." << std::endl;
        PrintErrorInformation(errCode, L"GetObject", pRuntime);
    }
}
```

See also

[ILoggedTag \(Page 244\)](#)

[ILoggedTagCallback / ILoggedTagSetCallback \(Page 239\)](#)

[ILoggedTagValueEnumerator \(Page 238\)](#)

9.3.13 ITags

Description

The C++ interface "ITags" defines methods with which you can access configured tags.

The interface inherits from the "ICfDispatch" interface.

All the methods return `CF_SUCCESS` following successful execution. In the case of an error, the methods return the corresponding error code.

Members

"Find" method

Supplies an enumerator for access to the "ITagAttributes" instances of the specified tag.

```
CFRESULT Find(
    CFVARIANT systemIDs,
    int32_t language,
    CFSTR filter,
    ITagAttributesEnumerator** ppITagAttributesEnumerator)
```

```
Find(CFVARIANT systemIDs, uint32_t language, CFSTR filter,
    ITagAttributesEnumerator** ppITagAttributesEnumerator)
```

- `systemIDs`
[in]: Collection of SystemNames on which the tags were configured.
- `language`
[in]: Language code ID of filter
- `filter`
[in]: Filter by name of the tags to restrict the search.
Supports searching with wildcard (*)
- `ppITagAttributesEnumerator`
[out]: Enumerator which supplies access to the "ITagAttributes" instances.

"FindAsync" method

For asynchronous searching for "ITagAttributes" instances of the specified tags.

```
FindAsync(
    CFVARIANT systemIDs
    uint32_t language
    CFSTR filter
    ITagAttributesCallback* pCallback)
```

- `systemIDs`
[in]: Collection of SystemNames on which the tags were configured.
- `language`
[in]: Language code ID of filter

9.3 Interfaces of the tags

- `filter`
[in]: Filter by name of the tags to restrict the search.
Supports searching with wildcard (*)
- `filterpCallback`:
[in]: Callback pointer to an "ITagAttributesCallback" instance

9.3.14 ITagAttributes

Description

The C++ interface "ITagAttributes" defines methods for access to the attributes of a tag.

All the methods return `CF_SUCCESS` following successful execution. In the case of an error, the methods return the corresponding error code.

Members

"GetName" method

Supplies the name of the tags. Must be unique throughout the device.

```
GetName (  
    CFSTR* name)
```

- `name`
[out]: The name

"GetDisplayName" method

Supplies the display name of the tags.

```
GetDisplayName (  
    CFSTR* name)
```

- `name`
[out]: The display name

"GetDataType" method

Supplies the data type of the tags.

```
GetDataType (  
    CFENUM* datatype)
```

- `datatype`
[out]: The data type of the tag

"GetConnection" method

Supplies the connection of the tag.

The memory location of the tag in the controller is accessed via the connection.

```
GetConnection(
    CFSTR* connection)
```

- `connection`
[out]: The connection

"GetAcquisitionCycle" method

Specifies the tag acquisition cycle.

If you configure a tag at an object, the acquisition cycle of the tag is displayed at the object.

```
GetAcquisitionCycle(
    CFSTR* acquisitionCycle)
```

- `acquisitionCycle`
[out]: The acquisition cycle

"GetAcquisitionMode" method

Supplies the acquisition mode of the tag.

```
GetAcquisitionMode(
    CFENUM* acquisitionMode)
```

- `acquisitionMode`
[out]: Value of the enumeration "HmiAcquisitionMode".
The enumeration "HmiAcquisitionMode" can contain the following values:
 - Undefined (0)
 - CyclicOnUse (1)
 - CyclicContinuous (2)
 - OnDemand (3)
 - OnChange (4)

"GetMaxLength" method

Supplies the length of the tags.

The length is only available with a string tag. The length is preset for structure tags and cannot be changed.

```
GetMaxLength(
    uint32_t* maxLength)
```

- `maxLength`
[out]: Maximum length

"GetPersistent" method

Supplies the persistence of the tags.

```
GetPersistent(
    CFBOOL* persistent)
```

- `persistent`
[out]: The persistence

9.3 Interfaces of the tags

"GetInitialValue" method

Supplies the start value of the tag.

After Runtime starts, the tag retains the start value until an operator or the PLC changes the value.

```
GetInitialValue(  
    CFVARIANT * initialValue)
```

- initialValue
[out]: The start value

"GetInitialMaxValue" method

Supplies the start value for the event "On exceeding".

```
InitialMaxValue(  
    CFVARIANT * initialValue)
```

- initialValue
[out]: The start value

"GetInitialMinValue" method

Supplies the start value for the event "On falling below".

```
InitialMinValue(  
    CFVARIANT * initialValue)
```

- initialValue
[out]: The start value

"GetSubstituteValue" method

Supplies the substitute value of the tags.

If the selected condition occurs, the tag is filled with the substitute value in runtime.

```
GetSubstituteValue(  
    CFVARIANT * substituteValue)
```

- substituteValue
[out]: The substitute value

"GetSubstituteValueUsage" method

Supplies the condition for the use of the substitute value of the tag.

```
GetSubstituteValueUsage(  
    CFVARIANT * substituteValueUsage)
```

- substituteValueUsage
[out]: The condition

9.3.15 ITagAttributesEnumerator

Description

The "ITagAttributesEnumerator" interface is a C++ interface that specifies methods for handling the enumeration of tag attributes. The enumerator enables access to individual attributes from a set of tag attributes.

The interface inherits from the "ICfUnknown" interface.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

"MoveNext" method

Go to the next element of the enumerator.

```
CFRESULT MoveNext()
```

"Current" method

Output the current element of the enumerator

```
CFRESULT Current(  
    ITagAttributes** ppItem)
```

- ppItem
[out]: The current "ITagAttributes" instance

"Reset" method

Reset the current position in the enumerator. The "MoveNext" method moves afterwards to the first element.

```
CFRESULT Reset()
```

"Count" method

Output the size of the enumerator or the number of its elements.

```
CFRESULT Count(  
    uint32_t* value)
```

- value
[out]: Number of attributes

9.3.16 ITagAttributesCallback

Description

The C++ interface "ITagAttributesCallback" defines the callback method "OnTagAttributesRead". The method is used for implementing asynchronous read operations of tag attributes. The method is used by the C++ interface "ITags".

The interface implements the methods of the "ICfUnknown" interface.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

"OnTagAttributesRead" method

Callback method, is called on completion of an asynchronous search for "ITagAttributes" instances.

```
OnTagAttributesRead(  
    ITagAttributesEnumerator* pEnumerator,  
    CFBOOL bIsCompleted)
```

- ITagAttributesEnumerator
[in/out]: Reference to an "ITagAttributesEnumerator" instance which provides access to the tag attributes.
- bIsCompleted
[out]: Supplies information on whether the read operation was successfully completed.

9.3.17 ILoggingTags

Description

The C++ interface "ILoggingTags" defines methods with which you can access configured logging tags.

The interface inherits from the "ICfDispatch" interface.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

"Find" method

Supplies an enumerator for access to the "ITagAttributes" instances of the specified logging tag.

```
CFRESULT Find(  
    CFVARIANT systemIDs,  
    uint32_t language,  
    CFSTR filter,
```



```

        ILoggingTagAttributesEnumerator**
ppILoggingTagAttributesEnumerator)

```

- `systemIDs`
[in]: Collection of SystemNames on which the logging tags were configured.
- `language`
[in]: Language code ID of filter
- `filter`
[in]: Filter by name of the logging tags to restrict the search.
Supports searching with wildcard (*).
Example:
`Tag1:*` Supplies all logging tags of "Tag1".
- `ppILoggingTagAttributesEnumerator`
[out]: Enumerator which supplies access to the "ILoggingTagAttributes" instances.

"FindAsync" method

For asynchronous searching for "ILoggingTagAttributes" instances.

```

CFRESULT FindAsync(CFVARIANT systemIDs,
                  uint32_t language,
                  CFSTR filter,
                  ILoggingTagAttributesCallback* pCallback)

```

- `systemIDs`
[in]: Collection of SystemNames on which the logging tags were configured.
- `language`
[in]: Language code ID of filter
- `filter`
[in]: Filter by name of the logging tags to restrict the search.
Supports searching with wildcard (*).
- `pCallback`
[in]: Callback pointer to an "ILoggingTagAttributesCallback" instance

9.3.18 ILoggingTagAttributes

Description

The C++ interface "ITagAttributes" defines methods for access to the attributes of a logging tag.

All the methods return `CF_SUCCESS` following successful execution. In the case of an error, the methods return the corresponding error code.

Members

"GetName" method

Supplies the name of the logging tag.

```
GetName (
    CFSTR* name)
```

- name
[out]: The name

"GetDisplayName" method

Supplies the display name of the logging tags.

```
GetDisplayName (
    CFSTR* name)
```

- name
[out]: The display name

"GetDataType" method

Supplies the data type of the logging tags.

```
GetDataType (
    CFENUM* datatype)
```

- datatype
[out]: The data type

9.3.19 ILoggingTagAttributesEnumerator

Description

The "ILoggingTagAttributesEnumerator" interface is a C++ interface that specifies methods for handling the enumeration of logging tag attributes. The enumerator enables access to individual attributes from a set of tag attributes.

The interface inherits from the "ICfUnknown" interface.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members**"MoveNext" method**

Go to the next element of the enumerator.

```
CFRESULT MoveNext ()
```

"Current" method

Output the current element of the enumerator

```
CFRESULT Current (
    ITagAttributes** ppItem)
```

- ppItem
[out]: The current "ITagAttributes" instance

"Reset" method

Reset the current position in the enumerator. The "MoveNext" method moves afterwards to the first element.

```
CFRESULT Reset()
```

"Count" method

Output the size of the enumerator or the number of its elements.

```
CFRESULT Count(
    uint32_t* value)
```

- value
[out]: Number of attributes

9.3.20 ILoggingTagAttributesCallback

Description

The C++ interface "ILoggingTagAttributesCallback" defines the callback method "OnTagAttributesRead". The method is used for implementing asynchronous read operations of logging tag attributes. The method is used by the C++ interface "ILoggingTags".

The interface implements the methods of the "ICfUnknown" interface.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

"OnTagAttributesRead" method

Callback method, is called on completion of an asynchronous search for "ITagAttributes" instances.

```
OnTagAttributesRead(
    ILoggingTagAttributesEnumerator* pEnumerator,
    CFBOOL bIsCompleted)
```

- ILoggingTagAttributesEnumerator
[in]: Reference to an "ILoggingTagAttributesEnumerator" instance which provides access to the tag attributes.
- bIsCompleted
Supplies information on whether the result of the read operation is complete or whether other events will come.

9.4 Interfaces of the alarms

9.4.1 IAlarmResult

Description

The C++ interface "IAlarmResult" specifies methods for accessing properties of active alarms of the Runtime system.

An "IAlarmResult" object is a pure data object that maps all properties of an active alarm.

Members

The following methods are specified in the interface:

"GetInstanceID" method

Return InstanceID for an alarm with multiple instances.

```
CFRESULT GetInstanceID(uint32_t *value)
```

value

[out]: Points to the InstanceID of the alarm.

"GetSourceID" method

Return source at which the alarm was triggered.

```
CFRESULT GetSourceID(CFSTR *value)
```

value

[out]: Points to the source of the alarm.

"GetName" method

Return name of the alarm.

```
CFRESULT GetName(CFSTR *value)
```

value

[out]: Shows the name of the alarm.

"GetAlarmClassName" method

Return name of the alarm class.

```
CFRESULT GetAlarmClassName(CFSTR *value)
```

value

[out]: Points to the symbol of the associated alarm class.

"GetAlarmClassSymbol" method

Return symbol of the alarm class

```
CFRESULT GetAlarmClassSymbol(CFSTR *value)
```

value

[out]: Shows the name of the associated alarm class.

"GetState" method

Return current alarm state.

```
CFRESULT GetState(int32_t *value)
```

value

[out]: Shows the current alarm status.

"GetStateText" method

Return current alarm state as text, for example "Incoming" or "Outgoing".

```
CFRESULT GetStateText(CFSTR *value)
```

value

[out]: Shows the current alarm status as text.

"GetEventText" method

Return text that describes the alarm event.

```
CFRESULT GetEventText(CFSTR *value)
```

value

[out]: Points to the text that describes the alarm event.

"GetInfoText" method

Return text that describes the alarm event.

```
CFRESULT GetInfoText(CFSTR *value)
```

value

[out]: Points to the text that describes an operator instruction for the alarm.

"GetAlarmText1GetAlarmText9" method

Return additional texts 1-9 of the alarm.

```
CFRESULT GetAlarmText1(CFSTR *value)
```

...

```
CFRESULT GetAlarmText9(CFSTR *value)
```

value

[out]: Points to the additional text of the alarm.

"GetTextColor" method

Return text color of the alarm state.

```
CFRESULT GetTextColor(uint32_t *value)
```

value

[out]: Points to the text color of the alarm state.

"GetBackColor" method

Return background color of the alarm state.

9.4 Interfaces of the alarms

```
CFRESULT GetBackColor(uint32_t *value)
```

value

[out]: Points to the background color of the alarm state.

"GetFlashing" method

Return flashing background color of the alarm state.

```
CFRESULT GetFlashing(CFBOOL *value)
```

value

[out]: Points to the flashing background color of the alarm state.

"GetModificationTime" method

Return time of the last modification to the alarm state.

```
CFRESULT GetModificationTime(CFDATE64 *value)
```

value

[out]: Points to the time of the last change of the alarm state.

"GetChangeReason" method

Return trigger event for change of the alarm state.

```
CFRESULT GetChangeReason(uint16_t *value)
```

value

[out]: Points to the trigger event for the change of the alarm state.

"GetRaiseTime" method

Return trigger time of the alarm.

```
CFRESULT GetRaiseTime(CFDATE64 *value)
```

value

[out]: Shows the triggering time of the alarm.

"GetAcknowledgementTime" method

Return time of alarm acknowledgment.

```
CFRESULT GetAcknowledgementTime(CFDATE64 *value)
```

value

[out]: Points to the time of alarm acknowledgment.

"GetClearTime" method

Return time of alarm reset

```
CFRESULT GetClearTime(CFDATE64 *value)
```

value

[out]: Points to the time of alarm reset.

"GetResetTime" method

Return time of alarm reset.

```
CFRESULT GetResetTime(CFDATE64 *value)
```

value

[out]: Points out the time of the alarm reset.

"GetSuppressionState" method

Return status of alarm visibility.

```
CFRESULT GetSuppressionState(uint8_t *value)
```

value

[out]: Points to the status of the alarm visibility.

"GetPriority" method

Return relevance for display and sorting of the alarm.

```
CFRESULT GetPriority(uint8_t *value)
```

value

[out]: Points to the relevance of the alarm.

"GetOrigin" method

Return origin for display and sorting of the alarm

```
CFRESULT GetOrigin(CFSTR *value)
```

value

[out]: Points to the origin of the alarm.

"GetArea" method

Return origin area for display and sorting of the alarm.

```
CFRESULT GetArea(CFSTR *value)
```

value

[out]: Points to the origin area of the alarm.

"GetValue" method

Return current process value of the alarm.

```
CFRESULT GetValue(CFVARIANT *value)
```

value

[out]: Points to the current process value of the alarm.

"GetValueQuality" method

Return quality of the process value of the alarm.

```
CFRESULT GetValueQuality(uint16_t *value)
```

value

[out]: Points to the quality of the process value of the alarm.

"GetValueLimit" method

Return limit of the process value of the alarm.

```
CFRESULT GetValueLimit(CFVARIANT *value)
```

value

[out]: Points to the limit of the process value of the alarm.

"GetUserName" method

Return user name of the operator input alarm.

```
CFRESULT GetUserName(CFSTR *value)
```

value

[out]: Points to the user name of the operator input alarm.

"GetLoopInAlarm" method

Return function that navigates from the alarm view to its origin.

```
CFRESULT GetLoopInAlarm(CFSTR *value)
```

value

[out]: Points to the function name that navigates to the origin of the alarm.

"GetAlarmParameterValues" method

Return parameter values of the alarm.

```
CFRESULT GetAlarmParameterValues(CFVARIANT *value)
```

value

[out]: Points to the parameter values of the alarm.

"GetInvalidFlags" method

Return marking of the alarm with invalid data.

```
CFRESULT GetInvalidFlags(uint8_t *value)
```

value

[out]: Points to the invalid data of the alarm.

"GetConnection" method

Return connection by which the alarm was triggered.

```
CFRESULT GetConnection CFSTR *value)
```

value

[out]: Points to the connection of the alarm.

"GetSystemSeverity" method

Return severity of the system error.

```
CFRESULT GetSystemSeverity(uint8_t *value)
```

value

[out]: Points to the severity of the system error.

"GetUserResponse" method

Return expected or required user response to the alarm.

```
CFRESULT GetUserResponse(uint8_t *value)
```


value

[out]: Points to the expected or required user response to the alarm.

"GetSourceType" method

Return source from which the alarm was generated, e.g. tag-based, controller-based or system-based alarm.

```
CFRESULT GetSourceType(uint16_t *value)
```

value

[out]: Points to the type of source.

"GetDeadBand" method

Return range of the triggering tag in which no alarms are generated.

```
CFRESULT GetDeadBand(CFVARIANT *value)
```

value

[out]: Points to the non-triggering range.

"GetId" method

Return ID of the alarm that is also used in the display.

```
CFRESULT GetId(uint32_t *value)
```

value

[out]: Points to the ID of the alarm.

"GetAlarmGroupID" method

Return the ID of the alarm group to which the alarm belongs.

```
CFRESULT GetAlarmGroupID(uint32_t* groupId)
```

groupId

[out]: The ID of the alarm group

"GetHostName" method

Return name of the host that triggered the alarm.

```
CFRESULT GetHostName(CFSTR *value)
```

value

[out]: Points to the host name.

"GetNotificationReason" method

Return the reason for the notification.

```
CFRESULT GetNotificationReason(CFENUM* value)
```

value:

[out]: Points to the enumeration of the notification reason. The notification reason can have the following values:

- `Unknown` (0)
- `Add` (1)
The alarm was added to the filtered result list. The alarm meets the filter criteria that apply to the monitoring.
- `Modify` (2)
Properties of the alarm were changed, but the alarm is still part of the filtered result list.
- `Remove` (3)
The alarm was part of the result list, but it no longer meets the filter criteria due to changes to its properties.

Note

Changes to the alarm only lead to notifications again when the alarm meets the filter criteria again. In this case, "NotificationReason" is set to `Add`.

Note

Removing alarm from business logic

The use case of the client determines whether the client ignores notifications via alarms with the "NotificationReason" `Modify` or `Remove`.

For example:

- **State-based monitoring:** The client wants to show a list of incoming alarms. All notification reasons are relevant. The client removes an alarm from the list as soon as the notification reason is `Remove`.
 - **Event-based monitoring:** The client wants to send an email when an alarm comes in. Only the notification reason `Add` is relevant.
-

Example:

An ODK client begins monitoring with the filter criterion "State" = 1. An alarm is triggered. Runtime notifies the ODK client of the "NotificationReason" as follows:

Notification-Reason	Description
Add	<ul style="list-style-type: none"> • The "State" property is 1. The alarm has come in.
Modify	<ul style="list-style-type: none"> • The "State" property has not changed. • Another property that is not part of the filter criterion has changed, e.g. "Priority".
Remove	The "State" property has changed, e.g. alarm has gone out.

Example

When alarm are incoming, output a selection of properties of the "IAlarmResult" objects:

9.4 Interfaces of the alarms

Copy code

```
// Callback for alarm notifications
CFRESULT CFCALLTYPE CAlarmValue::OnAlarm(IAlarmResultEnumerator* pItems, uint32_t
systemError, CFSTR systemName, CFBOOL completed)
{
    CFRESULT hr = CF_FALSE;
    CFBOOL bSet = false;
    uint32_t nsize;
    pItems->Count(&nsize);

    if (nsize > 0 && CF_SUCCEEDED(systemError)) {
        m_AlarmValue.clear();
        int index = 0;

        while (CF_SUCCEEDED(pItems->MoveNext()))
        {
            IAlarmResultPtr ppValues;
            if (CF_SUCCEEDED(pItems->Current(&ppValues)))
            {
                AlarmAttributes AlarmValue;
                AlarmValue.m_nInstanceID;
                ppValues->GetInstanceID(&AlarmValue.m_nInstanceID);
                AlarmValue.m_strSourceID;
                CCfString strId;
                ppValues->GetSourceID(&strId);
                AlarmValue.m_strSourceID = CCfSmartString(strId);
                CCfString strName;
                ppValues->GetName(&strName);
                AlarmValue.m_strName = CCfSmartString(strName);
                CCfString strClassName;
                ppValues->GetAlarmClassName(&strClassName);
                AlarmValue.m_strAlarmClassName = CCfSmartString(strClassName);
                ppValues->GetState(&AlarmValue.m_nState);
                CCfString strEvent;
                ppValues->GetEventText(&strEvent);
                AlarmValue.m_strEventText = CCfSmartString(strEvent);
                CCfString strText;
                ppValues->GetStateText(&strText);
                AlarmValue.m_strStateText = CCfSmartString(strText);
                ppValues->GetBackColor(&AlarmValue.m_nBackColor);
                ppValues->GetTextColor(&AlarmValue.m_nTextColor);
                ppValues->GetFlashing(&AlarmValue.m_bFlashing);

                ...

                AlarmValue.m_nAlarmsSize = nsize;

                AlarmValue.m_strSystemName = systemName;
                std::cout << "System name = " << AlarmValue.m_strSystemName.ToUTF8().c_str()
<< std::endl;

                m_AlarmValue.push_back(AlarmValue);
                std::cout << "Alarm name = " << strName.ToUTF8().c_str() << std::endl;
            }
            index++;
        }
    }
}
```

Copy code

```
        this->SetEvent();  
        return CF_SUCCESS;  
    }  
    return hr;  
}
```

See also

[IAlarmResultEnumerator](#) (Page 269)

[IAlarm](#) (Page 270)

[IAlarmSubscription](#) (Page 295)

9.4.2 IAlarmResultEnumerator

Description

The "IAlarmResultEnumerator" interface is a C++ interface that specifies methods for handling the enumeration of active alarms of the Runtime system. Through the enumeration you access individual alarms from the quantity of all active alarm of the runtime system.

All the methods return CF_SUCCESS following successful execution.

Members

The following methods are specified in the interface:

"Current" method

Output the current element of the enumeration of a list.

```
CFRESULT Current(IAlarmResult **ppItem)
```

ppItem

[out]: Points to the current "IAlarmResult" object as an element of the list.

"MoveNext" method

Go to the next element of the enumeration of a list.

```
CFRESULT MoveNext()
```

"Reset" method

Reset the current position in the enumeration of a list.

```
CFRESULT Reset()
```

The "MoveNext" method moves afterwards to the first element of the list.

"Count" method

Output the size of the enumeration or the number of elements of a list.

```
CFRESULT Count(uint32_t *value)
```

value

[out]: Points to the value for the number of elements of the list.

Example

When alarms are incoming, the "IAlarmResult" objects enumerate:

Copy code

```
// Callback for alarm notifications
CFRESULT CFCALLTYPE CAlarmValue::OnAlarm(IAlarmResultEnumerator* pItems, uint32_t
systemError, CFSTR systemName, CFBOOL completed)
{
    CFRESULT hr = CF_FALSE;
    CFBOOL bSet = false;
    uint32_t nsize;
    pItems->Count(&nsize);

    if (nsize > 0 && CF_SUCCEEDED(systemError)) {
        m_AlarmValue.clear();
        int index = 0;

        while (CF_SUCCEEDED(pItems->MoveNext()))
        {
            IAlarmResultPtr ppValues;
            if (CF_SUCCEEDED(pItems->Current(&ppValues)))
            {
                //AlarmResult processing...
            }
            index++;
        }
        this->SetEvent();
        hr = CF_SUCCESS;
    }
    return hr;
}
```

See also

[IAlarmResult \(Page 260\)](#)

9.4.3 IAlarm**Description**

The C++ interface "IAlarm" specifies properties and methods for handling active alarms of the Runtime system.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

The following methods are specified in the interface:

"SetSourceID" method

Set the ID of the source at which the active alarm was triggered.

```
CFRESULT SetSourceID(CFVARIANT sourceID)
```

sourceID

[in]: ID of the source of the active alarm

"SetName" method

Set the name of the active alarm.

```
CFRESULT SetName(CFSTR name)
```

name

[in]: Name of the active alarm

"GetName" method

Read out name of the active alarm.

```
CFRESULT GetName(CFSTR *name)
```

name

[out]: Points to a name of the active alarm.

"SetErrorCode" method

Set error code of the alarm.

```
CFRESULT SetErrorCode(uint32_t errorCode)
```

errorCode

[in]: Error code

"GetError" method

Read out error code of the alarm.

```
CFRESULT GetError(uint32_t *error)
```

error

[out]: Error code

"Disable" method

Deactivate generation of the alarm in the alarm source synchronously.

```
CFRESULT Disable()
```

"Enable" method

Activate generation of the alarm in the alarm source synchronously again.

CFRESULT Enable()

"Shelve" method

Hide active alarm synchronously.

CFRESULT Shelve()

"Unshelve" method

Display hidden alarm synchronously again.

CFRESULT Unshelve()

"Acknowledge" method

Acknowledge active alarm or instance of an active alarm synchronously.

CFRESULT Acknowledge(uint32_t instanceId)

- instanceId
Value "0": Acknowledge all instances of an active alarm.
Value > "0": Acknowledge instance with this ID.

"Reset" method

Acknowledge the outgoing state of an active alarm or an instance of an active alarm synchronously.

CFRESULT Reset(uint32_t instanceId)

- instanceId
Value "0": Acknowledge the outgoing state of the active alarm.
Value > "0": Acknowledge the outgoing state of an instance with this ID.

Example

Acknowledge list of active alarm synchronously:

Copy code

```
vector<AlarmAttributes> g_vecAlarmList;

void AcknowledgeAlarmSync(IRuntimePtr pRuntime)
{
    SubscribeAlarm(pRuntime);
    ICfUnknownPtr pUnk;
    if (CF_SUCCEEDED(pRuntime->GetObject(CCfString(L"Alarm"), &pUnk)))
    {
        IAlarmPtr pAlarm(pUnk);
        if (g_vecAlarmList.size() > 0)
        {
            vector<AlarmAttributes>::iterator it = g_vecAlarmList.begin();

            // iterate through list of notified alarms and acknowledge each alarm
            while (it != g_vecAlarmList.end())
            {
                CCfVariant vtAlarmName = it->m_strName;
                pAlarm->SetName(vtAlarmName);
                CFRESULT errCode = pAlarm->Acknowledge();
                if (CF_FAILED(errCode))
                {
                    std::wcout << L"Sync Acknowledge failed" << endl;
                    PrintErrorInformation(errCode, L"Acknowledge", pRuntime);
                }
                it++;
            }
            g_vecAlarmList.clear();
        }
    }
}
```

See also

IAlarmResult (Page 260)

9.4.4 IAlarmCallback

Description

The C++ interface "IAlarmCallback" defines methods for implementing asynchronous operations for monitoring active alarms. The methods are used by the "IAlarmSubscription" interface.

Members

The following methods are specified in the "IAlarmCallback" interface:

"OnAlarm" method

Callback method is called when active alarms on monitored systems change the alarm state.

The "OnAlarm" callback method is called when the "IAlarmSubscription.Start" method is used.

```
CFRESULT OnAlarm(  
    IAlarmResultEnumerator* pItems,  
    uint32_t systemError,  
    CFSTR systemName,  
    CFBOOL completed)
```

- `pItems`
[in]: Points to an "IAlarmResultEnumerator" object that contains the enumeration of the changed active alarms.
- `systemError`
[in]: Error code for the asynchronous operation
- `systemName`
[in]: System of the associated Runtime system.
- `completed`
[in]: True, if no more data from the callback can be expected.

"OnPendingAlarmsComplete" method

Callback method is called to display the completion of the handling of all the active alarms of a system.

The "OnPendingAlarmsComplete" callback method is called when the "IAlarmSubscription.Start" method is used.

```
CFRESULT OnPendingAlarmsComplete()
```

Example

In the following section monitored active alarms are written asynchronously into the "g_vecAlarmList" map at a change. To this purpose the "SubscribeAlarm" function uses a "CArmValue" object that implements the "IAlarmCallback" interface and that uses the "COdkTagSourceCBBase" class. The service life of the "CArmValue" object is determined via reference counting.

9.4 Interfaces of the alarms

Copy code

```

const uint32_t g_nMaxWaitTime = 6000;
vector<AlarmAttributes> g_vecAlarmList;

void SubscribeAlarm(IRuntimePtr pRuntime)
{
    ICfUnknownPtr pUnk;
    if (CF_SUCCEEDED(pRuntime->GetObject(CCFString(L"AlarmSubscription"), &pUnk))
    {
        IAlarmSubscriptionPtr pAlarm(pUnk);
        CAlarmValue* pAlarmValue = new CAlarmValue();
        pAlarmValue->AddRef();
        IAlarmCallback *pCB = pAlarmValue;

        CCfDafeArrayBound bounds(1UL, 0);

        CCfSafeArray daAttribute;
        CCfSafeArray daSystemID(CF_VT_SREF, 1, &bounds);
        CCfVariant daDataSource = 0;
        CCfVariant vSystemIDs = 0;

        CCfSmartString daFilter = L"";

        CCfSREF id(L"SYSTEM1");
        int32_t index = 0;
        daSystemID.PutElement(&index, &id);

        CCfVariant daLanguage = 1033;
        CFRESULT errCode;

        daSystemID.Detach(&vSystemIDs);

        CCfSmartString strFilter = L"";
        // Start Subscription
        pAlarm->SetFilter(strFilter.AllocCFSTR());
        pAlarm->SetLanguage(1033);
        pAlarm->SetSystemNames(vSystemIDs);

        errCode = pAlarm->Start(pCB);
        if (errCode == CF_SUCCESS)
        {
            std::wcout<< "Subscription Success"<<endl;
        }

        // Wait for alarm notifications
        if (pAlarmValue->WaitForcompletion(g_nMaxWaitTime) == CF_SUCCESS)
        {
            errCode = pAlarm->Stop();
            if (CF_FAILED(errCode))
            {
                std::wcout << "CancelSubscribe failed" << endl;
                PrintErrorInformation(errCode, L"CancelSubscribe", pRuntime);
            }

            // Get current alarms from callback
            pAlarmValue->GetAlarmAttributes(g_vecAlarmList);
        }
    }
}

```

Copy code

```

    }
}

```

See also

[IAlarmSubscription](#) (Page 295)

9.4.5 IAlarmSourceCommandCallback

Description

The C++ interface "IAlarmSourceCommandCallback" defines methods for implementing asynchronous operations with active alarms. The methods are used by the "IAlarmSet" interface.

Members

The following methods are specified in the interface:

"OnAcknowledge" method

Callback method is called when an active alarm was acknowledged.

The "OnAcknowledge" callback method is called when the "IAlarmSet.Acknowledge" method is used.

```

CFRESULT OnAcknowledge (
    CFRESULT SystemError,
    IAlarmSetResultEnumerator *AlarmSetResult,
    CFBOOL MoreFollows)

```

- `SystemError`
[in]: Basic errors that occurred during the asynchronous transfer.
- `AlarmSetResult`
[in]: Points to an enumeration with the alarms of the callback
- `MoreFollows`
[in]: Status of the asynchronous transfer:
 - True: Further callbacks are to be expected.
 - False: This is the last callback.

"OnReset" method

Callback method is called when an active alarm was removed.

The "OnReset" callback method is called when the "IAlarmSet.Reset" method is used.

```

CFRESULT OnReset (
    CFRESULT SystemError,

```

9.4 Interfaces of the alarms

```
IArmSetResultEnumerator *AlarmSetResult,  
CFBOOL MoreFollows)
```

- `SystemError`
[in]: Basic errors that occurred during the asynchronous transfer.
- `AlarmSetResult`
[in]: Points to an enumeration with the alarms of the callback
- `MoreFollows`
[in]: Status of the asynchronous transfer:
 - True: Further callbacks are to be expected.
 - False: This is the last callback.

"OnDisable" method

Callback method is called when the generation of the alarm in the alarm source was deactivated.

The "OnDisable" callback method is called when the "IArmSet.Disable" method is used.

```
CFRESULT OnDisable(  
    CFRESULT SystemError,  
    IArmSetResultEnumerator *AlarmSetResult,  
    CFBOOL MoreFollows)
```

- `SystemError`
[in]: Basic errors that occurred during the asynchronous transfer.
- `AlarmSetResult`
[in]: Points to an enumeration with the alarms of the callback
- `MoreFollows`
[in]: Status of the asynchronous transfer:
 - True: Further callbacks are to be expected.
 - False: This is the last callback.

"OnEnable" method

Callback method is called when the generation of the alarm in the alarm source was reactivated.

The "OnEnable" callback method is called when the "IArmSet.Enable" method is used.

```
CFRESULT OnEnable(  
    CFRESULT SystemError,
```

```

        IAlarmSetResultEnumerator *AlarmSetResult,
        CFBOOL MoreFollows)

```

- `SystemError`
[in]: Basic errors that occurred during the asynchronous transfer.
- `AlarmSetResult`
[in]: Points to an enumeration with the alarms of the callback
- `MoreFollows`
[in]: Status of the asynchronous transfer:
 - True: Further callbacks are to be expected.
 - False: This is the last callback.

"OnShelve" method

Callback method is called when an active alarm was hidden.

The "OnShelve" callback method is called when the "IAlarmSet.Shelve" method is used.

```

CFRESULT OnShelve(
    CFRESULT SystemError,
    IAlarmSetResultEnumerator *AlarmSetResult,
    CFBOOL MoreFollows)

```

- `SystemError`
[in]: Basic errors that occurred during the asynchronous transfer.
- `AlarmSetResult`
[in]: Points to an enumeration with the alarms of the callback
- `MoreFollows`
[in]: Status of the asynchronous transfer:
 - True: Further callbacks are to be expected.
 - False: This is the last callback.

"OnUnShelve" method

Callback method is called when an active alarm was displayed.

The "OnUnshelve" callback method is called when the "IAlarmSet.Unshelve" method is used.

```

CFRESULT OnUnshelve(
    CFRESULT SystemError,
    IAlarmSetResultEnumerator *AlarmSetResult,
    CFBOOL MoreFollows)

```

- `SystemError`
[in]: Basic errors that occurred during the asynchronous transfer.
- `AlarmSetResult`
[in]: Points to an enumeration with the alarms of the callback
- `MoreFollows`
[in]: Status of the asynchronous transfer:
 - True: Further callbacks are to be expected.
 - False: This is the last callback.

Example

In the following section monitored active alarms of the "g_vecAlarmList" vector are acknowledged asynchronously. To this purpose the "AcknowledgeAlarmAsync" function uses a "CAlarmSourceCommandCB" object that implements the "IAlarmSourceCommandCallback" interface and that uses the "COdkTagSourceCBBBase" class. The service life of the "CAlarmSourceCommandCB" object is determined via reference counting.

Copy code

```
const uint32_t g_nMaxWaitTime = 6000;
vector<AlarmAttributes> g_vecAlarmList;

void AcknowledgeAlarmAsync(IRuntimePtr pRuntime)
{
    SubscribeAlarm(pRuntime);
    ICfUnknownPtr pUnk;
    CFRESULT errCode;
    if (CF_SUCCEEDED(pRuntime->GetObject(CCfString(L"AlarmSet"), &pUnk)))
    {
        IAlarmSetPtr pAlarmSet(pUnk);
        CAlarmSourceCommandCB* pAlarmSoureCommand = new CAlarmSourceCommandCB();
        pAlarmSoureCommand->AddRef();
        IAlarmSourceCommandCallback* pCB = pAlarmSoureCommand;
        if (g_vecAlarmList.size() > 0)
        {
            vector<AlarmAttributes>::iterator it = g_vecAlarmList.begin();
            // iterate through list of notified alarms and acknowledge each alarm
            while (it != g_vecAlarmList.end())
            {
                IAlarm* pAlarm = nullptr;
                CCfVariant vtAlarmName = it->m_strName;
                errCode = pAlarmSet->Add(vtAlarmName, &pAlarm);
                if (CF_FAILED(errCode))
                {
                    PrintErrorInformation(errCode, L"AlarmSet", pRuntime);
                }
                it++;
            }
            // acknowledged the AlarmSet
            errCode = pAlarmSet->Acknowledge(pCB);
            // wait for acknowledge callback
            if (CF_SUCCEEDED(errCode) && pAlarmSoureCommand-
                >WaitForCompletion(g_nMaxWaitTime) == CF_SUCCESS)
            {
                //Check if an alarm could not be acknowledged
                pAlarmSoureCommand->PrintError(pRuntime, L"Acknowledge");
            }
            g_vecAlarmList.clear();
        }
    }
}
```

See also

[IAlarmSet \(Page 281\)](#)

9.4.6 IAlarmSet

Description

The C++ interface "IAlarmSet" specifies properties and methods for optimized access to several active alarms of the Runtime system.

After initialization of the "IAlarmSet" object, you can execute asynchronous operations with multiple alarms in one call, e. g. acknowledgment or commenting. Simultaneous access demonstrates better performance and lower communication load than single access to multiple alarms.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

The following methods are specified in the interface:

"GetCount" method

Return the number of alarms of an AlarmSet list.

```
CFRESULT GetCount(uint32_t *value)
```

value

[out]: Points to the value for the number of alarms of the AlarmSet list.

"Remove" method

Remove a single alarm or an instance of an alarm from an AlarmSet.

```
CFRESULT Remove(CFSTR name, uint32_t instanceId)
```

- name
[in]: Name of the alarm that is removed from the AlarmSet.
- instanceId
[in]:
Value = "0": Remove all instances of an active alarm.
Value > "0": Remove instance with this ID.

"Add" method

Add an active alarm or instance of the alarm to an AlarmSet.

```
CFRESULT Add(  
    CFVARIANT p_varName,
```

```
IAalarm** pAlarm,
uint32_t instanceId)
```

- `p_varName`
[in]: Name of alarm tag that is added to the AlarmSet.
- `pAlarm`
[out]: Points to the added "IAalarm" object of the AlarmSet.
- `instanceId`
[in]:
Value = "0": Add all instances of an active alarm.
Value > "0": Add instance with this ID.

"Get" method

Reference an alarm or an instance of an alarm from an AlarmSet.

```
CFRESULT Get (
    const CFSTR p_varName,
    IAalarm** pAlarm,
    uint32_t instanceId)
```

- `p_varName`
[in]: Name of the alarm tag.
- `pAlarm`
[out]: Points to the "IAalarm" object of the AlarmSet.
- `instanceId`
[in]:
Value = "0": Reference all instances of an active alarm.
Value > "0": Reference instance with this ID.

"Disable" method

Deactivate generation of the alarms of the AlarmSet in the alarm source asynchronously.

```
CFRESULT Disable (IAalarmSourceCommandCallback* p_pCallback)
```

`p_pCallback`

[in/out]: Points to an "IAalarmSourceCommandCallback" object that implements the asynchronous operation.

"Enable" method

Reactivate generation of the alarms of the AlarmSet in the alarm source asynchronously.

```
CFRESULT Enable (IAalarmSourceCommandCallback* p_pCallback)
```

`p_pCallback`

[in/out]: Points to an "IAalarmSourceCommandCallback" object that implements the asynchronous operation.

"Shelve" method

Hide alarms of the AlarmSet asynchronously.

```
CFRESULT Shelve (IAalarmSourceCommandCallback* p_pCallback)
```

p_pCallback

[in/out]: Points to an "IAlarmSourceCommandCallback" object that implements the asynchronous operation.

"Unshelve" method

Display hidden alarms of the AlarmSet once again.

```
CFRESULT Unshelve(IAlarmSourceCommandCallback* p_pCallback)
```

p_pCallback

[in/out]: Points to an "IAlarmSourceCommandCallback" object that implements the asynchronous operation.

"Acknowledge" method

Acknowledge alarms of the AlarmSet asynchronously.

```
CFRESULT Acknowledge(IAlarmSourceCommandCallback* p_pCallback)
```

p_pCallback

[in/out]: Points to an "IAlarmSourceCommandCallback" object that implements the asynchronous operation.

"Reset" method

Acknowledge outgoing state of the alarms of the AlarmSet asynchronously.

```
CFRESULT Reset(IAlarmSourceCommandCallback* p_pCallback)
```

p_pCallback

[in/out]: Points to an "IAlarmSourceCommandCallback" object that implements the asynchronous operation.

"Clear" method

Remove all alarms from an AlarmSet.

```
CFRESULT Clear()
```

Example

Remove active alarms from the "g_vecAlarmList" list as an AlarmSet asynchronously:

Copy code

```
const uint32_t g_nMaxWaitTime = 6000;
vector<AlarmAttributes> g_vecAlarmList;
SubscribeAlarm(pRuntime);
ICfUnknownPtr pUnk;
CFRESULT errCode;
if (CF_SUCCEEDED(pRuntime->GetObject(CCfString(L"AlarmSet"), &pUnk))
{
    IAlarmSetPtr pAlarmSet(pUnk);
    CAlarmSourceCommandCB* pAlarmSourceCommand = new CAlarmSourceCommandCB();
    pAlarmSourceCommand->AddRef();
    IAlarmSourceCommandCallback* pCB = pAlarmSourceCommand;
    if (g_vecAlarmList.size() > 0)
    {
        vector<AlarmAttributes>::iterator it = g_vecAlarmList.begin();
        // iterate through list of notified alarms and reset each alarm
        while (it != g_vecAlarmList.end())
        {
            IAlarm* palarm = nullptr;
            CCfVariant vtAlarmName = it->m_strName;
            errCode = pAlarmSet->Add(vtAlarmName, &palarm);
            if (CF_FAILED(errCode))
            {
                PrintErrorInformation(errCode, L"AlarmSet", pRuntime);
            }
            it++;
        }
        // Reset the AlarmSet
        errCode = pAlarmSet->Reset(pCB);
        if (CF_SUCCEEDED(errCode) && pAlarmSourceCommand-
>WaitForCompletion(g_nMaxWaitTime) == CF_SUCCESS)
        {
            //Check if an alarm could not be Reset
            pAlarmSourceCommand->PrintError(pRuntime, L"Reset");
        }
        g_vecAlarmList.clear();
    }
}
```

See also

[IAlarmSourceCommandCallback \(Page 277\)](#)

[IAlarmSetResult \(Page 285\)](#)

[IAlarmSetResultEnumerator \(Page 286\)](#)

9.4.7 IAlarmSetResult

Description

The C++ interface "IAlarmSetResult" specifies methods for accessing properties of monitored alarms in AlarmSets.

All the methods return CF_SUCCESS following successful execution.

Members

The following methods are specified in the interface:

"GetSystemName" method

Return system name of the Runtime system of an alarm in the AlarmSet.

```
CFRESULT GetSystemName(CFSTR *value)
```

value

[out]: Points to the associated system name.

"GetName" method

Return name of an alarm in the AlarmSet.

```
CFRESULT GetName(CFSTR *value)
```

value

[out]: Points to the name of an alarm.

"GetInstanceID" method

Return InstanceID of an alarm in the AlarmSet.

```
CFRESULT GetInstanceID(uint32_t *value)
```

value

[out]: Points to the InstanceID of an alarm

"GetErrorCode" method

Return error code of an alarm in the AlarmSet.

```
CFRESULT GetErrorCode(uint32_t *value)
```

value

[out]: Points to the error code of an alarm

See also

IAlarmSet (Page 281)

IAlarmSetResultEnumerator (Page 286)

IAlarmSubscription (Page 295)

9.4.8 IAlarmSetResultEnumerator

Description

The "IAlarmSetResultEnumerator" interface is a C++ interface that specifies methods for handling the enumeration of monitored alarms of the Runtime system.

All the methods return CF_SUCCESS following successful execution.

Members

The following methods are specified in the interface:

"Current" method

Output the current element of the enumeration of a list.

```
CFRESULT Current(IAlarmSetResult **ppItem)
```

ppItem

[out]: Points to the current "IAlarmSetResult" object as an element of the list.

"MoveNext" method

Go to the next element of the enumeration of a list.

```
CFRESULT MoveNext()
```

"Reset" method

Reset the current position in the enumeration of a list.

```
CFRESULT Reset()
```

The "MoveNext" method moves afterwards to the first element of the list.

"Count" method

Output the size of the enumeration or the number of elements of a list.

```
CFRESULT Count(uint32_t *value)
```

value

[out]: Points to the value for the number of elements of the list.

See also

IAlarmSet (Page 281)

IAlarmSetResult (Page 285)

9.4.9 IAlarmTrigger

Description

The C++ interface "AlarmTrigger" specifies methods for triggering alarms.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

"CreateSystemAlarm" method

Generates an alarm of the class SystemAlarmWithoutClearEvent with the state machine alarm without outgoing status with acknowledgment.

```
CFRESULT CreateSystemAlarm(
CFVARIANT alarmText,
CFSTR area,
CFVARIANT p_AlarmParameterValue1,
CFVARIANT p_AlarmParameterValue2,
CFVARIANT p_AlarmParameterValue3,
CFVARIANT p_AlarmParameterValue4,
CFVARIANT p_AlarmParameterValue5,
CFVARIANT p_AlarmParameterValue6,
CFVARIANT p_AlarmParameterValue7)
```

- alarmText
[in]: The alarm text. You have the following options:
 - Transferring a text list of type "ITextList".
The list entries of the text list can contain the multilingual text or references to other text lists.

Note

Only user-defined text lists

This method processes only user-defined text lists.

- Transfer static string with monolingual text.
- area
[in]: The area of origin of the alarm
- p_AlarmParameterValue<Number>
[in]: User-defined comments

The alarm triggers an event with the following event path:

- For multilingual alarm
texts:
`@ScriptingSystemEvents.SystemAlarmWithoutClearEvent:SystemAlarmWithoutClearEvent`
- For monolingual alarm
text:
`@ScriptingSystemEvents.SystemAlarmWithoutClearEventText:SystemAlarmWithoutClearEvent`

"CreateSystemInformation" method

Generates an alarm of the class SystemInformation with the state machine alarm without outgoing status without acknowledgment.

```
CFRESULT CreateSystemInformation(
CFVARIANT alarmText,
CFSTR area,
CFVARIANT p_AlarmParameterValue1,
CFVARIANT p_AlarmParameterValue2,
CFVARIANT p_AlarmParameterValue3,
CFVARIANT p_AlarmParameterValue4,
CFVARIANT p_AlarmParameterValue5,
CFVARIANT p_AlarmParameterValue6,
CFVARIANT p_AlarmParameterValue7)
```

- alarmText
[in]: The alarm text. You have the following options:
 - Transferring a text list of the type "ITextList".
The list entries of the text list can directly contain multilingual alarm texts or references to other text lists with multilingual alarm texts.

Note

Only user-defined text lists

This method processes only user-defined text lists.

- Transfer static string with monolingual text.
- area
[in]: The area of origin of the alarm
- p_AlarmParameterValue<Number>
[in]: User-defined comments

The alarm triggers an event with the following event path:

- For multilingual alarm
texts: `@ScriptingSystemEvents.SystemInformation:SystemInformation`
- For monolingual alarm
text: `@ScriptingSystemEvents.SystemInformationText:SystemInformation`

"CreateOperatorInputInformation" method

Generates an alarm of the class OperatorInputInformation with the state machine alarm without outgoing status without acknowledgment.

```
CFRESULT CreateOperatorInputInformation(
CFVARIANT alarmText,
CFSTR area,
CFVARIANT p_AlarmParameterValue1,
CFVARIANT p_AlarmParameterValue2,
CFVARIANT p_AlarmParameterValue3,
CFVARIANT p_AlarmParameterValue4,
CFVARIANT p_AlarmParameterValue5,
CFVARIANT p_AlarmParameterValue6,
CFVARIANT p_AlarmParameterValue7)
```

- alarmText
[in]: The alarm text. You have the following options:
 - Transferring a text list of the type "ITextList".
The list entries of the text list can directly contain multilingual alarm texts or references to other text lists with multilingual alarm texts.

Note**Only user-defined text lists**

This method processes only user-defined text lists.

- Transfer static string with monolingual text.
- area
[in]: The area of origin of the alarm
- p_AlarmParameterValue<Number>
[in]: User-defined comments

The alarm triggers an event with the following event path:

- For multilingual alarm
texts:
@ScriptingSystemEvents.OperatorInputInformationText:OperatorInputInformation
- For monolingual alarm
text:
@ScriptingSystemEvents.OperatorInputInformationText:OperatorInputInformation

Example

The following code examples show how to trigger alarms of the class `SystemAlarmWithoutClearEvent`. Alarms of the classes `SystemInformation` and `OperatorInputInformation` are triggered in the same way.

Copying code

```

void CreateSystemAlarm(IRuntimePtr pRuntime)
{
    //Create SystemAlarm with monolingual alarm text
    ICfUnknownPtr pUnk;
    CFRESULT errCode;
    if (CF_SUCCEEDED(pRuntime->GetObject(CCfString(L"AlarmSubscription"), &pUnk))
    {
        IAlarmSubscriptionPtr pAlarm(pUnk);
        CAlarmValue* pAlarmValue = new CAlarmValue();
        pAlarmValue->AddRef();
        IAlarmCallback* pCB = pAlarmValue;
        CCfSafeArrayBound bounds(1UL, 0);
        CCfSafeArray daAttribute;
        CCfSafeArray daSystemID(CF_VT_SREF, 1, &bounds);
        CCfVariant daDataSource = 0;
        CCfVariant vSystemIDs = 0;
        CCfSREF id(L"SYSTEM1");
        int32_t index = 0;
        daSystemID.PutElement(&index, &id);
        daSystemID.Detach(&vSystemIDs);
        CCfSmartString strFilter = L"AlarmClassName = 'SystemAlarmWithoutClearEvent'";
        // Start Subscription
        pAlarm->SetFilter(strFilter.AllocCFSTR());
        pAlarm->SetLanguage(1033);
        pAlarm->SetSystemNames(vSystemIDs);
        errCode = pAlarm->Start(pCB);
        if (CF_SUCCEEDED(errCode))
        {
            ICfUnknownPtr pUnktrig;
            if (CF_SUCCEEDED(pRuntime->GetObject(CCfString(L"AlarmTrigger"), &pUnktrig))
            {
                IAlarmTriggerPtr pTrigger(pUnktrig);
                if (pTrigger != nullptr)
                {
                    errCode = pTrigger->CreateSystemAlarm(CCfVariant(L"Alarm Text"),
                    CCfString(L"Alarm Area"), CCfVariant(L"param1"), CCfVariant(L"param2"),
                    CCfVariant(L"param3"), CCfVariant(L"param4"), CCfVariant(L"param5"), CCfVariant(L"param6"),
                    CCfVariant(L"param7"));
                    if (CF_FAILED(errCode))
                    {
                        PrintErrorInformation(errCode, L"AlarmTrigger", pRuntime);
                    }
                }
            }
        }
        // Wait for alarm notifications
        if (CF_SUCCEEDED(errCode) && pAlarmValue->WaitForcompletion(g_nMaxWaitTime) ==
        CF_SUCCESS)
        {
            errCode = pAlarm->Stop();
            if (CF_FAILED(errCode))
            {
                PrintErrorInformation(errCode, L"AlarmSubscription", pRuntime);
            }
            pAlarmValue->Release();
        }
    }
}

```

9.4 Interfaces of the alarms

Copying code

```

    }
}

void CreateSystemAlarmWithAlarmTextAsTextList(IRuntimePtr pRuntime)
{
    //Create SystemAlarm with multilingual alarm text; the translations are directly stored
    in the text list
    ICfUnknownPtr pUnk;
    CFRESULT errCode;
    if (CF_SUCCEEDED(pRuntime->GetObject(CCfString(L"AlarmSubscription"), &pUnk)))
    {
        IAlarmSubscriptionPtr pAlarm(pUnk);
        CAlarmValue* pAlarmValue = new CAlarmValue();
        pAlarmValue->AddRef();
        IAlarmCallback* pCB = pAlarmValue;
        CCfSafeArrayBound bounds(1UL, 0);
        CCfSafeArray daAttribute;
        CCfSafeArray daSystemID(CF_VT_SREF, 1, &bounds);
        CCfVariant daDataSource = 0;
        CCfVariant vSystemIDs = 0;
        CCfSREF id(L"SYSTEM1");
        int32_t index = 0;
        daSystemID.PutElement(&index, &id);
        daSystemID.Detach(&vSystemIDs);
        CCfSmartString strFilter = L"AlarmClassName = 'SystemAlarmWithoutClearEvent'";
        // Start Subscription
        pAlarm->SetFilter(strFilter.AllocCFSTR());
        pAlarm->SetLanguage(1033);
        pAlarm->SetSystemNames(vSystemIDs);
        errCode = pAlarm->Start(pCB);
        if (CF_SUCCEEDED(errCode))
        {
            ICfUnknownPtr pUnktrig;
            if (CF_SUCCEEDED(pRuntime->GetObject(CCfString(L"AlarmTrigger"), &pUnktrig)))
            {
                IAlarmTriggerPtr pTrigger(pUnktrig);
                if (pTrigger != nullptr)
                {
                    if (CF_SUCCEEDED(pRuntime->GetObject(CCfString("TextList"), &pUnktrig)))
                    {
                        ITextListPtr ptextList(pUnktrig);
                        if (nullptr != ptextList)
                        {
                            ptextList->SetName(CCfString(L"AlarmTextTemplate"));
                            ptextList->SetTextListEntryIndex(101);
                            errCode = pTrigger->CreateSystemAlarm(CCfVariant(ptextList),
                                CCfString(L"Alarm Area"), CCfVariant(L"param1"), CCfVariant(L"param2"),
                                CCfVariant(L"param3"), CCfVariant(L"param4"), CCfVariant(L"param5"), CCfVariant(L"param6"),
                                CCfVariant(L"param7"));
                            if (CF_FAILED(errCode))
                            {
                                PrintErrorInformation(errCode, L"AlarmTrigger", pRuntime);
                            }
                        }
                    }
                }
            }
        }
    }
}

```

Copying code

```

        }
    }
}

// Wait for alarm notifications
if (CF_SUCCEEDED(errCode) && pAlarmValue->WaitForCompletion(g_nMaxWaitTime) ==
CF_SUCCESS)
{
    errCode = pAlarm->Stop();
    if (CF_FAILED(errCode))
    {
        PrintErrorInformation(errCode, L"AlarmSubscription", pRuntime);
    }
    pAlarmValue->Release();
}
}

void CreateSystemAlarmWithTextListAsParameterValue(IRuntimePtr pRuntime)
{
    //
    Create SystemAlarm with multilingual alarm text; the text list references other text lists
    with translations
    ICfUnknownPtr pUnk;
    CFRESULT errCode;
    if (CF_SUCCEEDED(pRuntime->GetObject(CCfString(L"AlarmSubscription"), &pUnk)))
    {
        IAlarmSubscriptionPtr pAlarm(pUnk);
        CAlarmValue* pAlarmValue = new CAlarmValue();
        pAlarmValue->AddRef();
        IAlarmCallback* pCB = pAlarmValue;
        CCfSafeArrayBound bounds(1UL, 0);
        CCfSafeArray daAttribute;
        CCfSafeArray daSystemID(CF_VT_SREF, 1, &bounds);
        CCfVariant daDataSource = 0;
        CCfVariant vSystemIDs = 0;
        CCfSREF id(L"SYSTEM1");
        int32_t index = 0;
        daSystemID.PutElement(&index, &id);
        daSystemID.Detach(&vSystemIDs);
        CCfSmartString strFilter = L"AlarmClassName = 'SystemAlarmWithoutClearEvent'";
        // Start Subscription
        pAlarm->SetFilter(strFilter.AllocCFSTR());
        pAlarm->SetLanguage(1033);
        pAlarm->SetSystemNames(vSystemIDs);
        errCode = pAlarm->Start(pCB);
        if (CF_SUCCEEDED(errCode))
        {
            ICfUnknownPtr pUnktrig;
            if (CF_SUCCEEDED(pRuntime->GetObject(CCfString(L"AlarmTrigger"), &pUnktrig)))
            {
                IAlarmTriggerPtr pTrigger(pUnktrig);
                if (pTrigger != nullptr)
                {
                    ICfUnknownPtr pUnktextList, pUnktextList1;

```

9.4 Interfaces of the alarms

Copying code

```

        if (CF_SUCCEEDED(pRuntime->GetObject(CCfString("TextList"),
&pUnktextList)) && CF_SUCCEEDED(pRuntime->GetObject(CCfString("TextList"),
&pUnktextList1)))
    {
        ITextListPtr pTextList_1(pUnktextList);
        if (nullptr != pTextList_1)
        {
            pTextList_1->SetName(CCfString(L"Text_List_2"));
            pTextList_1->SetTextListEntryIndex(1); //Eng TL @1%t#2T@ Val: @3@s@
        }
        ITextListPtr pTextList_2(pUnktextList1);
        if (nullptr != pTextList_2)
        {
            pTextList_2->SetName(CCfString(L"Text_List_2"));
        }
        errCode = pTrigger->CreateSystemAlarm(CCfVariant(pTextList_1),
CCfString(L"Alarm Area"),
            CCfVariant(1), // Index for Text_list_2
            CCfVariant(pTextList_2), // text list object
            CCfVariant(L"Hello"), // Dynamic value of @3@s@
            CCfVariant(), CCfVariant(), CCfVariant(), CCfVariant());
        if (CF_FAILED(errCode))
        {
            PrintErrorInformation(errCode, L"AlarmTrigger", pRuntime);
        }
    }
}

// Wait for alarm notifications
if (CF_SUCCEEDED(errCode) && pAlarmValue->WaitForcompletion(g_nMaxWaitTime) ==
CF_SUCCESS)
{
    errCode = pAlarm->Stop();
    if (CF_FAILED(errCode))
    {
        PrintErrorInformation(errCode, L"AlarmSubscription", pRuntime);
    }
    pAlarmValue->Release();
}
}
}

```

9.4.10 ITextList

Description

The C++ interface "ITextList" is used to transfer multilingual alarm texts for system alarms and operator input alarms. See section IAlarmTrigger (Page 287), CreateSystemInformation method. An ITextList instance is passed to the alarm text. When the operator input alarm is generated, it is replaced by the configured text from the text list.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

"GetName" method

Returns the name of the text list.

GetName (CFSTR* name)

- name:
[out]: The name

"SetName" method

Set the name of the text list.

SetName (CFSTR name)

- name:
[in]: The name

"GetTextListEntryIndex" method

Return the index of the list entry.

GetTextListEntryIndex) (OUT uint32_t* pIndex)

- pIndex
[out]: The index

"SetTextListEntryIndex" method

Set the index of the list entry.

SetTextListEntryIndex) (IN uint32_t pIndex)

- pIndex
[in]: The index

9.4.11 IAlarmSubscription

Description

The C++ interface "IAlarmSubscription" specifies methods for monitoring tags of the Runtime system. The subscribed tags are monitored for a change of alarm state.

Members

The following methods are specified in the interface:

"Start" method

Start monitoring of active alarms.

9.4 Interfaces of the alarms

```
CFRESULT Start (IAlarmCallback* callbackPtr)
```

callbackPtr

[in/out]: Points to an "IAlarmCallback" object that implements the asynchronous monitoring.

"Stop" method

Stop monitoring of active alarms.

Note

Start and Stop in Windows Forms applications

Do not call the "Stop" method for a Windows Forms application in the same thread where you called "Start".

```
CFRESULT Stop ()
```

"SetSystemNames" method

Set system names of Runtime systems for monitoring of active alarms.

```
CFRESULT SetSystemNames (CFVARIANT systemIDs)
```

systemIDs

[in]: System names of Runtime systems

"GetSystemNames" method

Read out system names of Runtime systems for monitoring of active alarms.

```
CFRESULT GetSystemNames (CFVARIANT* systemIDs)
```

systemIDs

[out]: Points to system names of Runtime systems.

"SetLanguage" method

Set country identification of the language for monitored alarms.

```
CFRESULT SetLanguage (uint32_t language)
```

language

[in]: Country identification of the language

"GetLanguage" method

Read out country identification of the language for monitored alarms.

```
CFRESULT GetLanguage (uint32_t* language)
```

language

[out]: Points to country identification of the language.

"GetFilter" method

Supplies the string by which the result set is filtered.

```
CFRESULT GetFilter (CFSTR* filter)
```

- filter

[out]: SQL-type string for filtering the result set of active alarms.

"SetFilter" method

Sets the string for filtering the result set of active alarms.

```
CFRESULT SetFilter(IN CFSTR filter)
```

- `filter`
[in]: SQL-type string for filtering the result set of active alarms.
All properties of an alarm can be used in the filter string. The filter string can contain operators. Refer to the section Syntax of the alarm filter (Page 19).

Example

In the following section monitored active alarms are written asynchronously into the "g_vecAlarmList" map at a change. To this purpose the "SubscribeAlarm" function uses a "CArmValue" object that implements the "IAlarmCallback" interface and that uses the "COdkTagSourceCBBBase" class. The service life of the "CArmValue" object is determined via reference counting.

Copy code

```

const uint32_t g_nMaxWaitTime = 6000;
vector<AlarmAttributes> g_vecAlarmList;

void SubscribeAlarm(IRuntimePtr pRuntime)
{
    ICfUnknownPtr pUnk;
    if (CF_SUCCEEDED(pRuntime->GetObject(CCFString(L"AlarmSubscription"), &pUnk))
    {
        IAlarmSubscriptionPtr pAlarm(pUnk);
        CAlarmValue* pAlarmValue = new CAlarmValue();
        pAlarmValue->AddRef();
        IAlarmCallback *pCB = pAlarmValue;

        CCfSafeArrayBound bounds(1UL, 0);

        CCfSafeArray daAttribute;
        CCfSafeArray daSystemID(CF_VT_SREF, 1, &bounds);
        CCfVariant daDataSource = 0;
        CCfVariant vSystemIDs = 0;

        CCfSmartString daFilter = L"";

        CCfSREF id(L"SYSTEM1");
        int32_t index = 0;
        daSystemID.PutElement(&index, &id);

        CCfVariant daLanguage = 1033;
        CFRESULT errCode;

        daSystemID.Detach(&vSystemIDs);

        CCfSmartString strFilter = L"";
        // Start Subscription
        pAlarm->SetFilter(strFilter.AllocCFSTR());
        pAlarm->SetLanguage(1033);
        pAlarm->SetSystemNames(vSystemIDs);

        errCode = pAlarm->Start(pCB);
        if (errCode == CF_SUCCESS)
        {
            std::wcout<< "Subscription Success"<<endl;
        }

        // Wait for alarm notifications
        if (pAlarmValue->WaitForCompletion(g_nMaxWaitTime) == CF_SUCCESS)
        {
            errCode = pAlarm->Stop();
            if (CF_FAILED(errCode))
            {
                std::wcout << "CancelSubscribe failed" << endl;
                PrintErrorInformation(errCode, L"CancelSubscribe", pRuntime);
            }

            // Get current alarms from callback
            pAlarmValue->GetAlarmAttributes(g_vecAlarmList);
        }
    }
}

```

Copy code

```
    }  
}  
}
```

See also

[IAlarmResult](#) (Page 260)

[IAlarmCallback](#) (Page 273)

[IAlarmSetResult](#) (Page 285)

9.4.12 ILoggedAlarmResult

Description

The C++ interface "ILoggedAlarmResult" specifies methods for accessing properties of logged alarms of a logging system.

An "ILoggedAlarmResult" object is a pure data object that maps all properties of a logged alarm.

Members

The following methods are specified in the interface:

"GetInstanceID" method

Return InstanceID of a logged alarm.

```
CFRESULT GetInstanceID(uint32_t *value)
```

value

[out]: Points to the InstanceID of the logged alarm.

"GetName" method

Return name of the logged alarm.

```
CFRESULT GetName(CFSTR *value)
```

value

[out]: Points to the name of the logged alarm.

"GetAlarmClassName" method

Return name of the alarm class.

```
CFRESULT GetAlarmClassName(CFSTR *value)
```

value

[out]: Points to the symbol of the alarm class of the logged alarm.

"GetAlarmClassSymbol" method

Return symbol of the alarm class

```
CFRESULT GetAlarmClassSymbol(CFSTR *value)
```

value

[out]: Points to the name of the alarm class of the logged alarm.

"GetState" method

Return alarm state of the logged alarm.

```
CFRESULT GetState(int32_t* *value)
```

value

[out]: Points to the alarm state.

"GetStateText" method

Return alarm state of the logged alarm as text, for example "Incoming" or "Outgoing".

```
CFRESULT GetStateText(CFSTR *value)
```

value

[out]: Points to the alarm state as text.

"GetEventText" method

Return text that describes the alarm event of the logged alarm.

```
CFRESULT GetEventText(CFSTR *value)
```

value

[out]: Points to the text that describes the alarm event.

"GetAlarmText1GetAlarmText9" method

Return alarm texts 1-9 of the logged alarm.

```
CFRESULT GetAlarmText1(CFSTR *value)
```

...

```
CFRESULT GetAlarmText9(CFSTR *value)
```

value

[out]: Points to the additional text of the logged alarm.

"GetTextColor" method

Return text color of the alarm state of the logged alarm.

```
CFRESULT GetTextColor(uint32_t *value)
```

value

[out]: Points to the text color of the alarm state.

"GetBackColor" method

Return background color of the alarm state of the logged alarm.

```
CFRESULT GetBackColor(uint32_t *value)
```

value

[out]: Points to the background color of the alarm state.

"GetFlashing" method

Return flashing background color of the alarm state of the logged alarm.

```
CFRESULT GetFlashing(CFBOOL *value)
```

value

[out]: Points to the flashing background color of the alarm state.

"GetModificationTime" method

Time of the last change of the alarm state of the logged alarm.

```
CFRESULT GetModificationTime(CFDATE64 *value)
```

value

[out]: Points to the time of the last change of the alarm state.

"GetChangeReason" method

Return trigger event for the change of the alarm state of the logged alarm.

```
CFRESULT GetChangeReason(uint16_t *value)
```

value

[out]: Points to the trigger event for the change of the alarm state.

"GetRaiseTime" method

Return time at which the logged alarm was triggered.

```
CFRESULT GetRaiseTime(CFDATE64 *value)
```

value

[out]: Points to the time of the logged alarm trigger.

"GetAcknowledgementTime" method

Return time at which the logged alarm was acknowledged.

```
CFRESULT GetAcknowledgementTime(CFDATE64 *value)
```

value

[out]: Points to the time of the logged alarm acknowledgment.

"GetClearTime" method

Return time at which the logged alarm was cleared.

```
CFRESULT GetClearTime(CFDATE64 *value)
```

value

[out]: Points to the time of the logged alarm clearing.

"GetResetTime" method

Return time at which the logged alarm was reset.

```
CFRESULT GetResetTime(CFDATE64 *value)
```

value

[out]: Points to the time of the logged alarm reset.

"GetSuppressionState" method

Return status of the visibility of the logged alarm.

```
CFRESULT GetSuppressionState(uint8_t *value)
```

value

[out]: Points to the status of the visibility of the logged alarm.

"GetPriority" method

Return relevance for display and sorting of the logged alarm.

```
CFRESULT GetPriority(uint8_t *value)
```

value

[out]: Points to the relevance of the logged alarm.

"GetOrigin" method

Return origin for display and sorting of the logged alarm.

```
CFRESULT GetOrigin(CFSTR *value)
```

value

[out]: Points to the origin of the logged alarm.

"GetArea" method

Return origin area for display and sorting of the logged alarm.

```
CFRESULT GetArea(CFSTR *value)
```

value

[out]: Points to the origin area of the logged alarm.

"GetValue" method

Return process value of the logged alarm.

```
CFRESULT GetValue(CFVARIANT *value)
```

value

[out]: Points to the process value of the logged alarm.

"GetValueQuality" method

Return quality of the process value of the logged alarm.

```
CFRESULT GetValueQuality(uint16_t *value)
```

value

[out]: Points to the quality of the process value of the logged alarm.

"GetValueLimit" method

Return limit of the process value of the logged alarm.

```
CFRESULT GetValueLimit(CFVARIANT *value)
```

value

[out]: Points to the limit of the process value of the logged alarm.

"GetUserName" method

Return user name of the logged operator input alarm.

```
CFRESULT GetUserName(CFSTR *value)
```

value

[out]: Points to the user name of the logged operator input alarm.

"GetLoopInAlarm" method

Return function that navigates from the alarm view to its origin.

```
CFRESULT GetLoopInAlarm(CFSTR *value)
```

value

[out]: Points to the function name that navigates to the origin of the logged alarm.

"GetAlarmParameterValues" method

Return parameter values of the logged alarm.

```
CFRESULT GetAlarmParameterValues(CFVARIANT *value)
```

value

[out]: Points to the parameter values of the logged alarm.

"GetInvalidFlags" method

Return marking of the logged alarm with invalid data.

```
CFRESULT GetInvalidFlags(uint8_t *value)
```

value

[out]: Points to the invalid data of the logged alarm.

"GetConnection" method

Return connection via which the logged alarm was triggered.

```
CFRESULT GetConnection CFSTR *value)
```

value

[out]: Points to the connection of the logged alarm.

"GetSystemSeverity" method

Return severity of the system error.

```
CFRESULT GetSystemSeverity(uint8_t *value)
```

value

[out]: Points to the severity of the system error.

"GetUserResponse" method

Return expected or required user response to the logged alarm.

```
CFRESULT GetUserResponse(uint8_t *value)
```

value

[out]: Points to the expected or required user response to the logged alarm.

"GetDeadBand" method

Return range of the triggering tag in which no alarms are generated.

```
CFRESULT GetDeadBand(CFVARIANT *value)
```

value

[out]: Points to the DeadBand of the logged alarm.

"GetHostName" method

Return name of the host that triggered the alarm.

```
CFRESULT GetHostName(CFSTR *value)
```

value

[out]: Points to host name.

"GetInfoText" method

Return text for the alarm that contains the associated work instruction.

```
CFRESULT GetInfoText(CFSTR *value)
```

value

[out]: Points to the text of the operator instruction.

"GetStateMachine" method

Return StateMachine model of the alarm. The StateMachine represents the behavior of alarms through arrangement of alarm states and alarm events, e.g. "RaiseClear", "RaiseRequiresAcknowledgment" or "RaiseClearOptionalAcknowledgment".

```
CFRESULT GetStateMachine(uint8_t *value)
```

value

[out]: Shows the model of the StateMachine of the logged alarm.

"GetSingleAcknowledgement" method

Returns whether an alarm may be acknowledged only individually or may be acknowledged as a group or multiple selection.

```
CFRESULT GetSingleAcknowledgement(CFBOOL *value)
```

value

[out]: Points to the acknowledgement specification.

"GetLoggedAlarmStateObjectID" method

Return ID of alarm state for referencing within the logging system.

```
CFRESULT GetLoggedAlarmStateObjectID(CFSTR *value)
```

value

[out]: Points to the ID of the alarm state of the logged alarm.

"GetID" method

Return user-defined ID of the alarm that is also used in the display.

```
CFRESULT GetID(uint32_t *value)
```

value

[out]: Points to the ID of the logged alarm.

"GetSourceType" method

Return source from which the alarm was generated, e.g. tag-based, controller-based or system-based alarm.

```
CFRESULT GetSourceType(uint16_t *value)
```

value

[out]: Points to the type of source of the logged alarm.

See also

ILoggedAlarmResultEnumerator (Page 306)

IArmLogging (Page 307)

IArmLoggingSubscription (Page 311)

9.4.13 ILoggedAlarmResultEnumerator

Description

The "ILoggedAlarmResultEnumerator" interface is a C++ interface that specifies methods for handling the enumeration of logged alarms of a logging system. Through the enumeration you access individual alarms from the set of logged alarm of a logging system.

All the methods return CF_SUCCESS following successful execution.

Members

The following methods are specified in the interface:

"Current" method

Output the current element of the enumeration of a list.

```
CFRESULT Current(ILoggedAlarmResult **ppItem)
```

ppItem

[out]: Points to the current "ILoggedAlarmResult" object as an element of the list.

"MoveNext" method

Go to the next element of the enumeration of a list.

```
CFRESULT MoveNext()
```

"Reset" method

Reset the current position in the enumeration of a list.

```
CFRESULT Reset()
```

The "MoveNext" method moves afterwards to the first element of the list.

"Count" method

Output the size of the enumeration or the number of elements of a list.

```
CFRESULT Count(uint32_t *value)
```

value

[out]: Points to the value for the number of elements of the list.

See also

ILoggedAlarmResult (Page 300)

IArmLogging (Page 307)

9.4.14 IAlarmLogging

Description

The C++ interface "IArmLogging" specifies methods for reading out logged alarms of a logging system.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

The following methods are specified in the interface:

"Read" method

Read out logged alarms of a time period synchronously from logging system.

```
CFRESULT Read(
    CFDATE64 begin,
    CFDATE64 end,
    CFSTR filter,
    uint32_t language,
    CFVARIANT systemIDs,
    ILoggedAlarmResultEnumerator **ppEnumerator)
```

- begin
[in]: Start date of the time period
- end
[in]: End date of the time period
- filter
[in]: Filter for limiting the read operation with properties of the "ILoggedAlarmResult" object.
- language
[in]: Country identification of the language of the logged alarm text

9.4 Interfaces of the alarms

- `systemIDs`
[in]: System names of the Runtime systems of the logged alarms. Default: local system
- `ppEnumerator`
[out]: Points to the logged alarms as an "ILoggedAlarmResultEnumerator" object.

"ReadAsync" method

Read out logged alarms of a time period asynchronously from logging system.

```
CFRESULT ReadAsync(  
    CFDATE64 begin,  
    CFDATE64 end,  
    CFSTR filter,  
    uint32_t language,  
    CFVARIANT systemIDs,  
    IAlarmLoggingCallback *pLoggedAlarmCb)
```

- `begin`
[in]: Start date of the time period
- `end`
[in]: End date of the time period
- `filter`
[in]: Filter for limiting the read operation with properties of the "ILoggedAlarmResult" object.
- `language`
[in]: Country identification of the language of the logged alarm text
- `systemIDs`
[in]: System names of the Runtime systems of the logged alarms. Default: local system
- `pLoggedAlarmCb`
[in]: Points to the "IAlarmLoggingCallback" object that implements the callback interface.

Example

Read historical alarms synchronously from the logging system:

Copy code

```

void LoggingReadAlarmSync(IRuntimePtr pRuntime)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CcString(L"LoggedAlarm"), &pUnk);

    if (pUnk != nullptr && CF_SUCCEEDED(errCode))
    {
        CcVariant vSystemIDs = 0;
        CF_SAFEARRAYBOUND* bounds = nullptr;
        CCfSafeArray daSystemID(CF_VT_SREF, 1, bounds);
        CCfSREF id(L"SYSTEM1");
        int32_t index = 0;
        daSystemID.PutElement(&index, &id);
        daSystemID.Detach(&vSystemIDs);

        IAlarmLoggingPtr pAlarm(pUnk);

        CCfDateTime64 begin, end;
        begin = CCfDateTime64::Now(true);
        end = begin;
        begin.SubtractTimeSpan(Get1Minute() * 3);

        ILoggedAlarmResultEnumeratorPtr pItems;
        // Read value of tag
        errCode = pAlarm->Read(begin, end, CcString(""), 1033, vSystemIDs, &pItems);

        if (pItems != nullptr && CF_SUCCEEDED(errCode))
        {
            std::wcout << "Read finished " << std::endl;
            PrintValues(pItems);
        }
        else
        {
            std::wcout << L"Read operation failed." << std::endl;
        }
    }
    else
    {
        std::wcout << L"Error, couldn't create ODK object." << std::endl;
    }
}

```

See also

[ILoggedAlarmResult \(Page 300\)](#)
[ILoggedAlarmResultEnumerator \(Page 306\)](#)
[IAlarmLoggingCallback \(Page 310\)](#)
[IAlarmLoggingSubscription \(Page 311\)](#)

9.4.15 IAlarmLoggingCallback

Description

The C++ interface "IAlarmLoggingCallback" defines methods for implementing asynchronous operations for monitoring active alarms. The methods are used by the "IAlarmLogging" and "IAlarmLoggingSubscription" interfaces.

All the methods return CF_SUCCESS after execution.

Members

The following methods are specified in the "IAlarmCallback" interface:

"OnReadComplete" method

Callback method is called on completion of asynchronous read operations in logging systems.

The "OnReadComplete" callback method is called when the "IAlarmLogging.ReadAsync" method is used.

```
CFRESULT OnReadComplete (ILoggedAlarmResultEnumerator *pEnumerator,  
                          uint32_t errorCode,  
                          int32_t contextId)
```

- pEnumerator
[out]: Points to an "ILoggedAlarmResultEnumerator" object that contains the enumeration of the logged alarms.
- errorCode
[out]: Error code for the asynchronous operation
- contextId
[out]: ContextID as additional identification feature of the logged alarms.

"OnDataChanged" method

Callback method is called upon a change of a monitored alarm in logging systems.

The "OnDataChanged" callback method is called when the "IAlarmLoggingSubscription.Start" method is used.

```
CFRESULT OnDataChanged (ILoggedAlarmResultEnumerator *pEnumerator,  
                        uint32_t errorCode,  
                        int32_t contextId)
```

- pEnumerator
[out]: Points to an "ILoggedAlarmResultEnumerator" object that contains the enumeration of the logged alarms.
- errorCode
[out]: Error code for the asynchronous operation
- contextId
[out]: ContextID as additional identification feature of the logged alarms.

See also

IAlarmLogging (Page 307)

IAlarmLoggingSubscription (Page 311)

9.4.16 IAlarmLoggingSubscription**Description**

The C++ interface "IAlarmLoggingSubscription" specifies methods for monitoring logged alarms of an archive system.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

The following methods are specified in the interface:

"SetFilter" method

Set SQL-like string for filtering the result set of the logged alarms.

```
CFRESULT SetFilter(CFSTR filter)
```

filter

[in]: Filter string for logged alarms

"SetLanguage" method

Set country identifier of the language for monitoring of logged alarms.

```
CFRESULT SetLanguage(uint32_t language)
```

language

[in]: Country identification of the language

"SetSystemName" method

Set system names of Runtime systems for monitoring of logged alarms.

```
CFRESULT SetSystemName(CFVARIANT systemIDs)
```

systemIDs

[in]: System name of Runtime systems

"Start" method

Start monitoring of logged alarms.

9.4 Interfaces of the alarms

```
CFRESULT Start (IAlarmLoggingCallback* pLoggedAlarmCb)
```

- `filter`
[in]: Filter for limiting the read operation with properties of the "ILoggedAlarmResult" object.
- `pLoggedAlramCb`
[in/out]: Points to an "IAlarmLoggingCallback" object that implements asynchronous monitoring.

"Stop" method

Stop monitoring of all logged alarms.

```
CFRESULT Stop ()
```


Example

Monitoring logged alarms. The values are returned by the "IAlarmLoggingCallback" object:

Copy code

```

void LoggingSubscribeAlarm(IRuntimePtr pRuntime)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CcString(L"LoggedAlarmSubscription"), &pUnk);

    if (pUnk != nullptr && CF_SUCCEEDED(errCode))
    {
        CcVariant vSystemIDs = 0;
        CF_SAFEARRAYBOUND* bounds = nullptr;
        CcSafeArray daSystemID(CF_VT_SREF, 1, bounds);
        CcSREF id(L"SYSTEM1");
        int32_t index = 0;
        daSystemID.PutElement(&index, &id);
        daSystemID.Detach(&vSystemIDs);

        IAlarmLoggingSubscriptionPtr pAlarm(pUnk);
        pAlarm->SetSystemName(vSystemIDs);
        pAlarm->SetLanguage(1033);

        CodkAlarmLoggingCB* pAlarmCB = new CodkAlarmLoggingCB();

        if (pAlarmCB != nullptr && CF_SUCCEEDED(errCode))
        {
            pAlarmCB->AddRef();

            // subscribe tags
            errCode = pAlarm->Start(pAlarmCB);
            if (CF_FAILED(errCode))
            {
                std::wcout << L"Error, couldn't create callback interface." << std::endl;
                PrintErrorInformation(errCode, L"Start", pRuntime);
            }
        }
    }
    else
    {
        std::wcout << L"Error, couldn't create ODK object." << std::endl;
        PrintErrorInformation(errCode, L"GetObject", pRuntime);
    }
}

```

See also

[IAlarmLogging \(Page 307\)](#)

[IAlarmLoggingCallback \(Page 310\)](#)

[ILoggedAlarmResult \(Page 300\)](#)

9.5 Interfaces for connections

9.5.1 IConnectionResult

Description

The C++ interface "IConnectionResult" provides methods for access to the details of connections.

Members

The following methods are specified in the interface:

"GetName" method

Return name of the connection.

```
CFRESULT GetName(CFSTR *pName)
```

pName

[out]: Points to the name of the connection.

"GetConnectionState" method

Return status of the connection.

```
CFRESULT GetConnectionState(CFENUM *pConnectionState)
```

pConnectionState

[out]: Points to the enumeration, which can contain the following values:

- Disabled (0)
- Connecting (1)
- Connected (2)
- Disconnecting (3)
- Disconnected (4)
- Reconnecting (5)

"GetEstablishmentMode" method

Return mode in which the connection is established.

```
CFRESULT GetEstablishmentMode(CFENUM *pEstablishmentMode)
```

pEstablishmentMode

[out]: Points to the enumeration, which can contain the following values:

- None (0)
- AutomaticActive (1)
- AutomaticPassive (2)

- OnDemandActive (3)
- OnDemandPassive (4)

"GetTimeSynchronizationMode" method

Mode of time synchronization between HMI system and AS.

```
CFRESULT GetTimeSynchronizationMode (CFENUM  
*pTimeSynchronizationMode)
```

```
pTimeSynchronizationMode
```

[out]: Points to the enumeration, which can contain the following values:

- None (0)
- Slave (1)
- Master (2)

"GetDisabledAtStartup" method

Indicates whether the connection is disabled at the start of Runtime.

```
CFRESULT GetDisabledAtStartup (CFBOOL *pDisabledAtStartup)
```

```
pDisabledAtStartup
```

[out]: Points to a Boolean value.

"GetEnabled" method

Indicates whether the connection is active.

```
CFRESULT GetEnabled (CFBOOL *pEnabled)
```

```
pbEnabled
```

[out]: Points to a Boolean value.

"GetConnectionType" method

Return protocol of a communication driver, e.g. "S7 Classic".

```
CFRESULT GetConnectionType (CFSTR *pConnectionType)
```

```
pConnectionType
```

[out]: Points to the name of the protocol.

"GetError" method

Return error code of the connection.

```
CFRESULT GetError (uint32_t *pError)
```

```
pError
```

[out]: Points to the error code.

Example

Output connection details:

Copy code

```

void DisplayConnectionInfo(IConnectionResultPtr pConnectionresult)
{
    if (nullptr != pConnectionresult)
    {
        CCfString strName;
        pConnectionresult->GetName(&strName);
        std::cout << "ConnectionName:" << strName.ToUTF8() << std::endl;

        CCfString strConnectiontype;
        pConnectionresult->GetConnectionType(&strConnectiontype);
        std::cout << "ConnectionType:" << strConnectiontype.ToUTF8() << std::endl;

        CFENUM enConnectionState;
        pConnectionresult->GetConnectionState(&enConnectionState);
        HmiConnectionState enumconnectionState =
static_cast<HmiConnectionState>(enConnectionState);

        ConnectionState(enumconnectionState);
        CFENUM enEstablishmentMode;
        pConnectionresult->GetEstablishmentMode(&enEstablishmentMode);
        HmiConnectionEstablishmentMode enumEstablishmentMode =
static_cast<HmiConnectionEstablishmentMode>(enEstablishmentMode);
        Establishmentmode(enumEstablishmentMode);

        CFENUM enTimeSynchronizationMode;
        pConnectionresult->GetTimeSynchronizationMode(&enTimeSynchronizationMode);
        HmiTimeSynchronizationMode enumTimeSynchronizationMode =
static_cast<HmiTimeSynchronizationMode>(enTimeSynchronizationMode);
        TimeSynchronizationmode(enumTimeSynchronizationMode);

        CFBOOL bDisableatStartup;
        pConnectionresult->GetDisabledAtStartup(&bDisableatStartup);
        std::cout << "DisableStartup:" << (int)bDisableatStartup << std::endl;

        CFBOOL bEnabled;
        pConnectionresult->GetEnabled(&bEnabled);
        std::cout << "Enabled:" << (int)bEnabled << std::endl;

        uint32_t nerror;
        pConnectionresult->GetError(&nerror);
        std::cout << "Error:" << nerror << std::endl;
    }
    std::cout << std::endl;
}

```

See also

[IConnection \(Page 322\)](#)

[IConnectionResultEnumerator \(Page 317\)](#)

9.5.2 IConnectionResultEnumerator

Description

The "IConnectionResultEnumerator" interface is a C++ interface that specifies methods for handling the enumeration of connection details of the Runtime system. The enumeration is used, for example, when reading out connections of a connection set.

All the methods return CF_SUCCESS following successful execution.

Members

The following methods are specified in the interface:

"Current" method

Output the current element of the enumeration of a list.

```
CFRESULT Current(IConnectionResult **ppItem)
```

ppItem

[out]: Points to the current "IConnectionResult" object as an element of the list.

"MoveNext" method

Go to the next element of the enumeration of a list.

```
CFRESULT MoveNext()
```

"Reset" method

Reset the current position in the enumeration of a list.

```
CFRESULT Reset()
```

The "MoveNext" method moves afterwards to the first element of the list.

"Count" method

Output the size of the enumeration or the number of elements of a list.

```
CFRESULT Count(uint32_t *pCount)
```

pCount

[out]: Points to the value for the number of elements of the list.

See also

IConnectionResult (Page 314)

IConnectionSet (Page 328)

9.5.3 IConnectionStatusResult

Description

The C++ interface "IConnectionStatusResult" provides methods for access to the status of connections.

Members

The following methods are specified in the interface:

"GetName" method

Return name of the connection.

```
CFRESULT GetName(CFSTR *pName)
```

pName

[out]: Points to the name of the connection.

"GetConnectionStatus" method

Return status of the connection.

```
CFRESULT GetConnectionStatus(CFENUM *pConnectionStatus)
```

pConnectionStatus

[out]: Points to the enumeration, which can contain the following values:

- Disabled (0)
- Connecting (1)
- Connected (2)
- Disconnecting (3)
- Disconnected (4)
- Reconnecting (5)

"GetError" method

Return error code of the connection.

```
CFRESULT GetError(uint32_t *pError)
```

pError

[out]: Points to the error code.

Example

Output status of a certain connection:

Copy code

```

void ConnectionSet_GetConnectionState(IRuntimePtr pRuntime)
{
    std::cout << "ConnectionSet_GetConnectionState  Start" << std::endl;
    CFRESULT hr = CF_ERROR;
    ICfUnknownPtr pUnk;
    CFRESULT errorCode = pRuntime->GetObject(CcString(L"ConnectionSet"), &pUnk);
    if (pUnk != nullptr && CF_SUCCEEDED(errorCode))
    {
        IConnectionSetPtr pConnectionsetPtr(pUnk);
        CCfSmartString strName(L"HMI-Connection");
        CCfString strName1(L"HMI-ConnectionS7Plus");
        CCfArrayVariant vtArrayVaiarnt;
        vtArrayVaiarnt.Append(strName);
        vtArrayVaiarnt.Append(strName1);
        ICfArrayVariantPtr connectionNames;
        vtArrayVaiarnt.DetachEnumerator(&connectionNames);
        hr = pConnectionsetPtr->Add(connectionNames);

        if (nullptr != pConnectionsetPtr)
        {
            IConnectionStatusResultEnumeratorPtr pConnectionStatusResultEnum;
            IConnectionStatusResultPtr pConnectionStatusresult;
            errorCode = pConnectionsetPtr->GetConnectionState(&pConnectionStatusResultEnum);
            if (CF_SUCCEEDED(errorCode))
            {
                uint32_t nCount;
                pConnectionStatusResultEnum->Count(&nCount);
                for (int32_t i = 0; i < nCount; i++)
                {
                    pConnectionStatusResultEnum->MoveNext();
                    if (CF_SUCCEEDED(pConnectionStatusResultEnum-
>Current(&pConnectionStatusresult)))
                    {
                        if (nullptr != pConnectionStatusresult)
                        {
                            CCfString strName;
                            pConnectionStatusresult->GetName(&strName);
                            std::cout << "ConnectionName:" << strName.ToUTF8() << std::endl;
                            uint32_t nerror;
                            pConnectionStatusresult->GetError(&nerror);
                            std::cout << "Error:" << nerror << std::endl;
                            CFENUM enConnectionState;
                            pConnectionStatusresult->GetConnectionStatus(&enConnectionState);
                            HmiConnectionState enumconnectionState =
static_cast<HmiConnectionState>(enConnectionState);
                            ConnectionState(enumconnectionState);
                        }
                    }
                }
            }
            else
            {
                std::cout << L" ConnectionSet_GetConnectionState  failed." << "errCode = "
<< errorCode << std::endl;
            }
        }
    }
}

```


Copy code

```

    }
}
std::cout << std::endl;
}

```

See also

[IConnection](#) (Page 322)

[IConnectionStatusResultEnumerator](#) (Page 321)

9.5.4 IConnectionStatusResultEnumerator**Description**

The "IConnectionStatusResultEnumerator" interface is a C++ interface that specifies methods for handling the enumeration of connection status of the Runtime system. The enumeration is used, for example, when reading out connections of a connection set.

All the methods return CF_SUCCESS following successful execution.

Members

The following methods are specified in the interface:

"Current" method

Output the current element of the enumeration of a list.

```
CFRESULT Current(IConnectionStatusResult **ppItem)
```

ppItem

[out]: Points to the current "IConnectionStatusResult" object as an element of the list.

"MoveNext" method

Go to the next element of the enumeration of a list.

```
CFRESULT MoveNext()
```

"Reset" method

Reset the current position in the enumeration of a list.

```
CFRESULT Reset()
```

The "MoveNext" method moves afterwards to the first element of the list.

"Count" method

Output the size of the enumeration or the number of elements of a list.

```
CFRESULT Count(uint32_t *pCount)
```

pCount

[out]: Points to the value for the number of elements of the list.

See also

IConnectionStatusResult (Page 318)

IConnectionSet (Page 328)

9.5.5 IConnection

Description

The C++ interface "IConnection" provides properties and methods for access to a connection.

Members

The following methods are specified in the interface:

"GetName" method

Return name of the connection.

```
CFRESULT GetName (CFSTR *pName)
```

pName

[out]: Points to the name of the connection.

"SetName" method

Change name of the connection.

```
CFRESULT SetName (CFSTR name)
```

name

[in]: Name of the connection

"Read" method

Read connection details synchronously from the Runtime system.

```
CFRESULT Read (IConnectionResult **ppConnectionResult)
```

ppConnectionResult

[out]: Points to an object of type "IConnectionResult" that contains the connection details.

"GetConnectionState" method

Return connection status of a connection.

```
CFRESULT GetConnectionState (IConnectionStatusResult  
**ppConnectionStatusResult)
```

ppConnectionStatusResult

[out]: Points to an object of type "IConnectionStatusResult" that contains the status of connections.

"SetConnectionMode" method

Change connection status of a connection.

```
CFRESULT SetConnectionMode(CFENUM connectionmode)
```

connectionmode

[in]: Enumeration which contains the mode of connections:

- Disabled (0)
- Enabled (1)

Examples

Change status of a connection:

Copy code

```
void Connection_SetConnectionState(IRuntimePtr pRuntime, ConnectionMode connectionMode)
{
    std::cout << "Connection_SetConnectionState Start :" << std::endl;
    ICfUnknownPtr pUnk;
    if (CF_SUCCEEDED(pRuntime->GetObject(CcString(L"Connection"), &pUnk))
    {
        IConnectionPtr pConnection(pUnk);

        CcSmartString daName = L"HMI-Connection";
        pConnection->SetName(daName.AllocCFSTR());
        CFRESULT hr = pConnection->SetConnectionMode(int32_t(connectionMode));
        if (CF_SUCCEEDED(hr))
            std::cout << "Connection_SetConnectionState Succeeded" << std::endl;
        else
        {
            std::cout << "Connection_SetConnectionState failed" << std::endl;
        }
    }
    std::cout << std::endl;
}
```

See also

[IConnectionResult \(Page 314\)](#)

[IConnectionStatusResult \(Page 318\)](#)

[IConnectionSet \(Page 328\)](#)

9.5.6 IConnectionReadNotification**Description**

The C++ interface "IConnectionReadNotification" defines a callback method for implementation of operations following read operations of connections.

Members

The following method is specified in the interface:

"OnReadComplete" method

Callback method is called following read operations of connections.

The "OnReadComplete" callback method is called when the `IConnectionSet.ReadAsync` method is used.

```
CFRESULT OnReadComplete(IConnectionResultEnumerator *pEnumerator,  
uint32_t errorCode, int32_t contextId)
```

- `pEnumerator`
[out]: Points to an "IConnectionResultEnumerator" object that contains the enumeration of the connection details.
- `errorCode`
[out]: Error code for the asynchronous operation.
- `contextId`
[out]: ContextID as additional identification feature of the connections.

Example

Read out connection asynchronously:

Copy code

```

void ConnectionSet_ReadAsync(IRuntimePtr pRuntime)
{
    std::cout << "ConnectionSet_ReadAsync Start :" << std::endl;
    CFRESULT hr = CF_ERROR;
    ICfUnknownPtr pUnk;

    CFRESULT errorCode = pRuntime->GetObject(CcString(L"ConnectionSet"), &pUnk);
    if (pUnk != nullptr && CF_SUCCEEDED(errorCode))
    {
        IConnectionSetPtr pConnectionset(pUnk);
        IConnectionResultPtr pConnectionResult;
        CcString strName(L"HMI-Connection");
        CcString strName1(L"HMI-Connections7Plus");
        CcArrayVariant vtArrayVaiarnt;
        vtArrayVaiarnt.Append(strName);
        vtArrayVaiarnt.Append(strName1);
        ICfArrayVariantPtr connectionNames;
        vtArrayVaiarnt.DetachEnumerator(&connectionNames);
        hr = pConnectionset->Add(connectionNames);
        if (nullptr != pConnectionset)
        {
            CcRefPtr<CConnectionReadNotification>pRead = new CConnectionReadNotification();
            IConnectionReadNotification *pNotification = pRead;
            errorCode = pConnectionset->ReadAsync(pNotification);
            if (CF_SUCCEEDED(errorCode))
            {
                if (pRead->WaitForcompletion(dwMilliseconds) == CF_SUCCESS)
                {
                    std::cout << " OnRead Succeeded." << "errCode = " << errorCode << std::endl;
                }
            }
            else
            {
                std::cout << L" ConnectionSet_ReadAsync failed." << "errCode = " << errorCode
<< std::endl;
            }
        }
    }
    std::cout << std::endl;
}

```

See also

IConnectionSet (Page 328)

9.5.7 IConnectionStateChangeNotification

Description

The C++ interface "IConnectionStateChangeNotification" defines a callback method for implementing asynchronous change monitoring of connections.

Members

The following method is specified in the interface:

"OnDataChanged" method

Callback method is called after changes of a monitored connection.

The "OnDataChanged" callback method is called when the `IConnectionSet.Subscribe` method is used.

```
CFRESULT OnDataChanged(IConnectionStatusResultEnumerator*  
pEnumerator, uint32_t errorCode, int32_t contextId)
```

- `pEnumerator`
[out]: Points to an "IConnectionStatusResultEnumerator" object that contains the enumeration of the connection status.
- `errorCode`
[out]: Error code for the asynchronous operation.
- `contextId`
[out]: ContextID as additional identification feature of the connections.

Example

Monitor connection status:

Copy code

```

void ConnnectionSet_Subscribe(IRuntimePtr pRuntime)
{
    std::cout << "ConnnectionSet_Subscribe Start" << std::endl;
    CFRESULT hr = CF_ERROR;
    ICfUnknownPtr pUnk;

    CFRESULT errorCode = pRuntime->GetObject(CcString(L"ConnnectionSet"), &pUnk);
    if (pUnk != nullptr && CF_SUCCEEDED(errorCode))
    {
        IConnnectionSetPtr pConnnectionsetPtr(pUnk);
        CcString strName(L"RUNTIME_1::Connnection3");
        CcString strName1(L"HMI-ConnnectionS7Plus");
        CcArrayVariant vtArrayVaiarnt;
        vtArrayVaiarnt.Append(strName);
        vtArrayVaiarnt.Append(strName1);
        ICfArrayVariantPtr connectionNames;
        vtArrayVaiarnt.DetachEnumerator(&connectionNames);

        hr = pConnnectionsetPtr->Add(connectionNames);

        if (nullptr != pConnnectionsetPtr)
        {
            CConnnectionSubscriptionNotification *pSubscribe = new
CConnnectionSubscriptionNotification();
            IConnnectionStateChangeNotification *pNotification = pSubscribe;
            errorCode = pConnnectionsetPtr->Subscribe(pNotification);
            if (CF_SUCCEEDED(errorCode))
            {
                if (pSubscribe->WaitForcompletion(dwMilliseconds) == CF_SUCCESS)
                {
                    //cancel subscription
                    pConnnectionsetPtr->CancelSubscribe();
                }
            }
            else
            {
                std::cout << L" ConnnectionSet_Subscribe failed." << "errCode = " << errorCode
<< std::endl;
            }
        }
        std::cout << std::endl;
    }
}

```

See also

IConnnectionSet (Page 328)

9.5.8 IConnectionSet

Description

The C++ interface "IConnectionSet" specifies properties and methods for optimized access to several connections of the Runtime system.

After initialization of the "IConnectionSet" object, you have read/write access to multiple connections in one call. Simultaneous access takes place with better performance and lower communication load than single access to multiple connections.

Members

The following methods are specified in the interface:

"SetContextId" method

Change ID as additional identification feature of a connection. The ContextId can, for example, be used to recognize identically named connections.

```
CFRESULT SetContextId(uint32_t id)
```

id

[in]: ContextID of the connection

"GetContextId" method

Return ID as additional identification feature of a connection. The ContextId can, for example, be used to recognize identically named connections.

```
CFRESULT GetContextId(uint32_t *pId)
```

pId

[out]: Points to the ContextID of the connection.

"Add" method

Add connections to a connection set.

```
CFRESULT Add(ICfArrayVariant *connectionNames)
```

connectionNames

[in]: Points to an array that contains the names of connections.

"Remove" method

Remove individual connection from connection set.

```
CFRESULT Remove(CFSTR connectionName)
```

connectionName

[in]: Name of the connection.

"Clear" method

Remove all connections from connection set.

```
CFRESULT Clear()
```


"GetCount" method

Return number of connections of a connection set list.

```
CFRESULT GetCount(int32_t *pCount)
```

pCount

[out]: Points to the number of connections in the connection set.

"Read" method

Read connection details of all connections of the connection set synchronously from the Runtime system.

```
CFRESULT Read(IConnectionResultEnumerator  
**ppConnectionResultEnumerator)
```

ppConnectionResultEnumerator

[out]: Points to the enumeration of the details of the individual connections.

"ReadAsync" method

Read connection details of all connections of the connection set asynchronously from the Runtime system.

```
CFRESULT ReadAsync(IConnectionReadNotification *pReadReply)
```

pReadReply

[in]: Points to the "IConnectionReadNotification" callback interface for read operations and returns the "IConnectionResultEnumerator" object.

"GetConnectionState" method

Read connection status synchronously from the Runtime system.

```
CFRESULT GetConnectionState(IConnectionStatusResultEnumerator  
**ppConnectionStatusResultEnumerator)
```

ppConnectionStatusResultEnumerator

[out]: Points to an object of type "IConnectionStatusResultEnumerator" that contains the enumeration of the connection status.

"Subscribe" method

Subscribe all connections of a connection set asynchronously for change monitoring.

```
CFRESULT Subscribe(IConnectionStateChangeNotification  
*ppNotificationCB)
```

ppNotificationCB

[in]: Points to the "IConnectionStateChangeNotification" callback interface for monitoring and returns the "IConnectionStatusResultEnumerator" object following a change.

"CancelSubscribe" method

Cancel change monitoring of all connections of a connection set.

```
CFRESULT CancelSubscribe()
```

Examples

Monitor connection:

Copy code

```

void CConnectionSet_Subscribe(IRuntimePtr pRuntime)
{
    std::cout << "CConnectionSet_Subscribe Start" << std::endl;
    CFRESULT hr = CF_ERROR;
    ICfUnknownPtr pUnk;

    CFRESULT errorCode = pRuntime->GetObject(CcString(L"ConnectionSet"), &pUnk);

    if (pUnk != nullptr && CF_SUCCEEDED(errorCode))
    {
        IConnectionSetPtr pConnectionsetPtr(pUnk);
        CcString strName(L"RUNTIME_1::Connection3");
        CcString strName1(L"HMI-ConnectionS7Plus");
        CcArrayVariant vtArrayVaiarnt;
        vtArrayVaiarnt.Append(strName);
        vtArrayVaiarnt.Append(strName1);
        ICfArrayVariantPtr connectionNames;
        vtArrayVaiarnt.DetachEnumerator(&connectionNames);

        hr = pConnectionsetPtr->Add(connectionNames);

        if (nullptr != pConnectionsetPtr)
        {
            CConnectionSubscriptionNotification *pSubscribe = new
CConnectionSubscriptionNotification();
            IConnectionStateChangeNotification *pNotification = pSubscribe;
            errorCode = pConnectionsetPtr->Subscribe(pNotification);
            if (CF_SUCCEEDED(errorCode))
            {
                if (pSubscribe->WaitForcompletion(dwMilliseconds) == CF_SUCCESS)
                {
                    //cancel subscription
                    pConnectionsetPtr->CancelSubscribe();
                }
            }
            else
            {
                std::cout << L" CConnectionSet_Subscribe failed." << "errCode = " << errorCode
<< std::endl;
            }
        }
    }

    std::cout << std::endl;
}

```

Read out connection asynchronously:

Copy code

```
void ConnectionSet_ReadAsync(IRuntimePtr pRuntime)
{
    std::cout << "ConnectionSet_ReadAsync Start :" << std::endl;
    CFRESULT hr = CF_ERROR;
    ICfUnknownPtr pUnk;

    CFRESULT errorCode = pRuntime->GetObject(CcString(L"ConnectionSet"), &pUnk);

    if (pUnk != nullptr && CF_SUCCEEDED(errorCode))
    {
        IConnectionSetPtr pConnectionset(pUnk);
        IConnectionResultPtr pConnectionResult;
        CcString strName(L"HMI-Connection");
        CcString strName1(L"HMI-ConnectionS7Plus");
        CcArrayVariant vtArrayVaiarnt;
        vtArrayVaiarnt.Append(strName);
        vtArrayVaiarnt.Append(strName1);
        ICfArrayVariantPtr connectionNames;
        vtArrayVaiarnt.DetachEnumerator(&connectionNames);

        hr = pConnectionset->Add(connectionNames);

        if (nullptr != pConnectionset)
        {
            CCfRefPtr<CConnectionReadNotification>pRead = new CConnectionReadNotification();
            IConnectionReadNotification *pNotification = pRead;
            errorCode = pConnectionset->ReadAsync(pNotification);
            if (CF_SUCCEEDED(errorCode))
            {
                if (pRead->WaitForCompletion(dwMilliseconds) == CF_SUCCESS)
                {
                    std::cout << " OnRead Succeeded." << "errCode = " << errorCode << std::endl;
                }
            }
            else
            {
                std::cout << L" ConnectionSet_ReadAsync failed." << "errCode = " << errorCode
<< std::endl;
            }
        }
    }
    std::cout << std::endl;
}
```

Add/remove connection for connection set

Copy code

```

void Connection_AddRemove(IRuntimePtr pRuntime)
{
    std::cout << "Connection_AddRemove Start :" << std::endl;
    CFRESULT hr = CF_ERROR;
    ICfUnknownPtr pUnk;

    CFRESULT errorCode = pRuntime->GetObject(CcString(L"ConnectionSet"), &pUnk);

    if (pUnk != nullptr && CF_SUCCEEDED(errorCode))
    {
        IConnectionSetPtr pConnectionsetPtr(pUnk);
        CcString strName(L"HMI-Connection");
        CcSmartString strName1(L"HMI-ConnectionS7Plus");
        CcArrayVariant vtArrayVaiarnt;
        vtArrayVaiarnt.Append(strName);
        vtArrayVaiarnt.Append(strName1);
        ICfArrayVariantPtr connectionNames;
        vtArrayVaiarnt.DetachEnumerator(&connectionNames);
        hr = pConnectionsetPtr->Add(connectionNames);
        int32_t count;
        pConnectionsetPtr->GetCount(&count);
        std::cout << "Count = " << count << std::endl;
        pConnectionsetPtr->Remove(strName1.AllocCFSTR());
        pConnectionsetPtr->GetCount(&count);
        std::cout << "Count = " << count << std::endl;
        pConnectionsetPtr->Clear();
        pConnectionsetPtr->GetCount(&count);
        std::cout << "Count = " << count << std::endl;
    }
    std::cout << std::endl;
}

```

See also[IConnectionReadNotification \(Page 323\)](#)[IConnectionStateChangeNotification \(Page 326\)](#)[IConnection \(Page 322\)](#)[IConnectionResultEnumerator \(Page 317\)](#)[IConnectionStatusResultEnumerator \(Page 321\)](#)

9.6 Interfaces of the Plant Model

9.6.1 IPlantModel

Description

The C++ interface "IPlantModel" specifies methods for access to object instances of the plant model of a Runtime system. The "IPlantModel" object represents the plant model of the graphical Runtime system.

The interface inherits from the "ICfDispatch" interface.

Formatting of a hierarchy path

A hierarchy path of object instances consists of several components and has the following syntax:

```
[SystemName].HierarchyName::[PlantObjectID/.../]PlantObjectID
```

The system name can be omitted for referencing a local hierarchy. The dot before the hierarchy name must stay.

Members

The following methods are specified in the interface:

"GetPlantObject" method

Supplies an "IPlantObject" instance.

```
CFRESULT GetPlantObject(const CFSTR Node, IPlantObject**  
ppPlantObject)
```

- Node
[in]: Identifies an IPlantObject instance by its name or its path in the hierarchy.
- ppPlantObject
[out]: Points to an "IPlantObject" instance.

"GetPlantObjectsByType" method

Supplies an enumeration with "IPlantObject" instances that have a specific type.

```
GetPlantObjectsByType(const CFSTR plantObjectTypeFilter, const CFSTR  
ViewFilter, IPlantObjectEnumerator** ppPlantObjects)
```

- plantObjectTypeFilter
[in]: Filter for the "IPlantObject" type on which the instances are based.
- ViewFilter
[in]: Filter for the path in the hierarchy. Only instances from a specific node are returned.
- ppPlantObjects
[out]: Points to an "IPlantObjectEnumerator" enumeration with "IPlantObject" instances.

"GetObjectsByPropertyName" method

Supplies an enumeration with "IPlantObject" instances that have specific properties and originate in a specific plant node.

```
GetPlantObjectsByPropertyNames(const CFVARIANT PropertyNames, const
CFSTR ViewFilter, IPlantObjectEnumerator** ppPlantObjects)
```

- `PropertyNames`
[in]: Property names
- `ViewFilter`
[in]: Filter for a hierarchy path.
- `ppPlantObjects`
[out]: Points to an "IPlantObjectEnumerator" enumeration with "IPlantObject" instances.

"GetPlantObjectsByExpression" method

Supplies an enumeration with "IPlantObject" instances. The instances are filtered by type and property values.

```
GetPlantObjectsByExpression(const CFVARIANT PropertyNames, const
CFSTR plantObjectTypeFilter, const CFSTR expressionFilter, const
CFSTR ViewFilter, IPlantObjectEnumerator** ppPlantObjects)
```

- `PropertyNames`
[in]: Property names
If the list contains multiple values, all properties must be available at the object.
- `plantObjectTypeFilter`
[in]: Filter for the object type on which the instances are based.
- `expressionFilter`
[in]: An expression that is a filter for the property values.
- `ViewFilter`
Filter for a hierarchy path.
- `ppCpmNodes`
[out]: Points to an "IPlantObjectEnumerator" enumeration with "IPlantObject" instances.

Example

Hierarchy path	Referenced object instance
<code>System2.TechnologicalHierarchy::P1/S1/L2/LeftPump</code>	References the "LeftPump" object instance in the "TechnologicalHierarchy" of system2.
<code>.TechnologicalHierarchy::P1/S1/L2/LeftPump</code>	References the "LeftPump" object instance in the "TechnologicalHierarchy" of the local system.
<code>U4711</code>	References the "U4711" object instance of the local system.
<code>System2::U4711</code>	References the "U4711" object instance of System2.

Copy code

```

void PlantModelGetPlantObjectsByType(IRuntimePtr pRuntime)
{
    if (nullptr != pRuntime)
    {
        ICfUnknownPtr pUnk;
        CFRESULT errCode = pRuntime->GetObject(CcFString(L"PlantModel"), &pUnk);
        if (CF_SUCCEEDED(errCode) && nullptr != pUnk)
        {
            IPlantModelPtr pPlantModel(pUnk);
            IPlantObjectEnumeratorPtr pPlantObjectEnum;
            errCode = pPlantModel->GetPlantObjectsByType(CcFString(L"BLOWER").Get(),
CcFString(L"").Get(),
&pPlantObjectEnum);
            if (CF_SUCCEEDED(errCode) && nullptr != pPlantObjectEnum)
            {
                IPlantObjectPtr pItem; pPlantObjectEnum->MoveNext();
                while (CF_SUCCEEDED(pPlantObjectEnum->Current(&pItem)))
                {
                    displayNodeInfo(pItem);
                    pPlantObjectEnum->MoveNext();
                }
            }
            else
            {
                std::wcout << L"PlantModelGetPlantObjectsByType operation Failed" << L"ErrCode
= " << errCode <<
                endl;
                PrintErrorInformation(errCode, CcFSmartString(L"objectbytype"), pRuntime);
            }
        }
        else
        {
            std::cout << L"Error, couldn't create ODK object." << "errCode= " << errCode <<
std::endl;
        }
    }
}

```

```

void PlantModelGetPlantObjectsByExpressionByAllFilter(IRuntimePtr pRuntime)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CcFString(L"PlantModel"), &pUnk);
    if (pUnk != nullptr && CF_SUCCEEDED(errCode))
    {
        IPlantModelPtr pPlantModel(pUnk);
        IPlantObjectEnumeratorPtr pPlantObjectEnum;
        CcFString str1 = L"NumberOfNodes";
        CcFString str2 = L"Quality";
        CcFSafeArray daSafeArray(CF_VT_STR, 1U, 2U);
        int32_t index = 0;
        daSafeArray.PutElement(&index, (void*)&str1);
        ++index;
        daSafeArray.PutElement(&index, (void*)&str2);
        ++index;
    }
}

```

Copy code

```

CCfVariant vtProperties;
daSafeArray.Detach(&vtProperties);
errCode = pPlantModel->GetPlantObjectsByExpression(vtProperties,
CCfString(L"BLOWER").Get(), CCfString(L"NumberOfNodes>=0"),
        CCfString(L"RUNTIME_1.hierarchy").Get(), &pPlantObjectEnum);
if (pPlantObjectEnum != nullptr && CF_SUCCEEDED(errCode))
{
    pPlantObjectEnum->MoveNext();
    IPlantObject* pPlantObject;
    while (CF_SUCCEEDED(pPlantObjectEnum->Current(&pPlantObject)))
    {
        displayNodeInfo(pPlantObject);
        pPlantObjectEnum->MoveNext();
    }
}
else
{
    std::cout << L"GetPlantObjectsByExpression operation failed." << "errCode
= " << errCode << std::endl;
}
}
else
{
    std::cout << L"Error, couldn't create ODK object." << "errCode = " << errCode
<< std::endl;
}
}

```

9.6.2 IPlantObject

Description

The C++ interface "IPlantObject" specifies methods for handling object instances of the plant model of a Runtime system.

An object instance in the plant model is based on an object type and its data structure. Each object instance receives its position within the plant hierarchy by assigning it to a hierarchy node.

The interface inherits from the "ICfDispatch" interface.

Formatting of a hierarchy path

A hierarchy path of object instances consists of several components and has the following syntax:

```
[SystemName].HierarchyName::[NodeID/.../]NodeID
```

The system name can be omitted for referencing a local hierarchy. The dot before the hierarchy name must stay.

Members

The following methods are specified in the interface:

"GetName" method

Supplies the name of the "IPlantObject" instance for unique identification.

```
CFRESULT GetName(CFSTR* pName)
```

- pName
[out]: Name of the instance

"GetParent" method

Supplies the parent of the "IPlantObject" instance in the hierarchy.

```
CFRESULT GetParent(IPlantObject** ppParent)
```

- ppParent
[out]: The parent as "IPlantObject" instance.

"GetChildren" method

Supplies an enumeration with the children of the "IPlantObject" instance in the hierarchy.

```
CFRESULT GetChildren(IPlantObjectEnumerator** ppChildren)
```

- ppChildren
[out]: An enumeration of the type "IPlantObjectEnumerator" with child object instances.

"GetPlantViewPaths" method

Supplies a map with hierarchy names and hierarchy paths that the "IPlantObject" instance has in all hierarchies in which it is included.

```
CFRESULT GetPlantViewPaths(ICfMapStringToVariant **pViewPaths)
```

- pViewPaths
[out]: A map with String/String pairs (hierarchy name to hierarchy path).

"GetCurrentPlantView" method

Supplies the path and names of the "IPlantObject" instance in the current hierarchy.

If the "IPlantObject" instance is only contained in one hierarchy, this path is returned.

```
CFRESULT GetCurrentPlantView)(CFSTR* pView)
```

- pView
[out]: Hierarchy path and name of the "IPlantObject" instance.

"SetCurrentPlantView" method

"CurrentPlantView" is the basis for navigation with the "GetParent" or "GetChildren" methods. If the "IPlantObject" instance is contained in several hierarchies, the path must be set via "SetCurrentPlantView" before the "GetParent" or "GetChildren" method can be used.

```
CFRESULT SetCurrentPlantView)(CFSTR const& View)
```

- View
[in]: The current hierarchy

"GetProperty" method

Supplies a property of the "IPlantObject" instance.

```
CFRESULT GetProperty(const CFSTR propertyName,
IPlantObjectProperty** ppPlantObjectProperty)
```

- `propertyName`
[in]: Name of a property of the "IPlantObject" instance
- `ppPlantObjectProperty`
[out]: Points to an "IPlantObjectProperty" instance.

"GetProperties" method

Supplies a two-dimensional list (name-object pairs) of the data structure of the "IPlantObject" instance. The list allows access to the instance properties.

```
CFRESULT GetProperties(const CFVARIANT propertyNamees,
IPlantObjectPropertySet** ppPlantObjectPropertySet)
```

- `Optional: propertyNamees`
[in]: List with names of one or multiple properties of the "IPlantObject" instance.
Without "propertyNamees" parameter, all properties of the instance are referenced in the list.
- `ppPlantObjectPropertySet`
[out]: Points to the list of the type "IPlantObjectPropertySet" that contains the names of one or multiple properties of the "IPlantObject" instance.

"GetActiveAlarms" method

Supplies all active alarms that the "IPlantObject" instance contains at the time it is called in the active hierarchy. Unlike with an AlarmSubscription, no status changes or new alarms are signaled that occur after the function call. Users can filter the alarms or specify a SystemName if they only want to receive the active alarms of a specific system.

```
CFRESULT GetActiveAlarms(uint32_t language, CFBOOL IncludeChildren,
CFSTR filter, IN IPlantObjectAlarmCallback* pCallback)
```

- `language`
[in]: Language code of the language for all alarm texts and the filters. See chapter Locale IDs of the supported languages (Page 21).
- `IncludeChildren`
[in]: The active alarms of the child instances are returned as well.
- `filter`
[in]: SQL-type string for filtering the alarm texts. The filter can contain operators. See also Syntax of the alarm filter (Page 19).
- `pCallback`
[in]: Callback pointer.

"CreateAlarmSubscription" method

Supplies a "PlantObjectAlarmSubscription" that can be used to start and stop an alarm subscription.

```
CFRESULT CreateAlarmSubscription) (IPlantObjectAlarmSubscription**  
ppPlantObjectAlarmSubscription)
```

- ppPlantObjectAlarmSubscription
[out] Points to a "PlantObjectAlarmSubscription" instance.

Example

Copy code

```

void PlantObjectGetProperties(IRuntimePtr pRuntime)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CcFString(L"PlantModel"), &pUnk);
    {
        if (nullptr != pUnk && CF_SUCCEEDED(errCode))
        {
            IPlantModelPtr pPlantModel(pUnk);
            IPlantObjectPtr pPlantObject;
            CcFString strNode = L".hierarchy::PlantView\\Unit1";

            //gets node for specified Node path
            errCode = pPlantModel->GetPlantObject(strNode.Get(), &pPlantObject);
            if (nullptr != pPlantObject && CF_SUCCEEDED(errCode))
            {
                //empty property - so all node members should be read
                CCfVariant vtProperties;
                IPlantObjectPropertySetPtr pPlantObjectPropertySet;
                // get the PlantObject properties by property names
                errCode = pPlantObject->GetProperties(vtProperties,
                &pPlantObjectPropertySet);
                if (nullptr != pPlantObjectPropertySet && CF_SUCCEEDED(errCode))
                {
                    uint32_t nCount = 0;
                    pPlantObjectPropertySet->GetCount(&nCount);
                    std::cout << "Count :" << nCount << std::endl;
                }
                else
                {
                    std::cout << L" GetProperties operation failed." << "errCode = " <<
errCode << std::endl;
                    PrintErrorInformation(errCode, L"Getproperties", pRuntime);
                }
            }
            else
            {
                std::cout << L"GetPlantObject operation failed." << "errCode = " <<
errCode << std::endl;
            }
        }
        else
        {
            std::cout << L"Error, couldn't create ODK object." << "errCode = " <<
errCode << std::endl;
        }
    }
}

```

9.6.3 IPlantObjectProperty

Description

The C++ interface "IPlantObjectProperty" specifies the handling of properties of object instances of the plant model of a Runtime system. The properties represent the data structure of an object instance.

The object instance communicates with the automation system through the properties of the data structure. The values of the properties are obtained from linked process tags or internal tags.

You reference an object using the `IPlantObject.GetProperty` or `IPlantObject.GetProperties` methods.

The interface inherits from the "ICfDispatch" interface.

Members

The following methods are specified in the interface:

"GetName" method

Supplies the name of the property.

```
CFRESULT GetName(CFSTR *pName)
```

- `pName`
[out]: Points to the name of the property.

"Read" method

Reads the value of the "IPlantObjectProperty" instance synchronously and returns it as an "IPlantObjectPropertyValue" object. The value, the quality code and the time stamp of the property are determined when the property is read.

```
CFRESULT Read(IPlantObjectPropertyValue**  
ppPlantObjectPropertyValue)
```

- `ppPlantObjectPropertyValue`
[out]: Values of the property as "IPlantObjectPropertyValue" instance

"Write" method

Writes the value synchronously to the "IPlantObjectProperty" instance.

```
CFRESULT Write(const CFVARIANT value)
```

- `value`
[in]: New process value of the property

Example**Copy code**

```

void PlantObjectReadproperty(IRuntimePtr pRuntime)
{
    std::cout << "PlantObjectReadproperty Start" << std::endl;
    ICfUnknownPtr pUnk;CFRESULT errCode = pRuntime->GetObject(CcString(L"PlantModel"),
    &pUnk);
    {
        if (nullptr != pUnk && CF_SUCCEEDED(errCode))
        {
            IPlantModelPtr pPlantModel(pUnk);
            IPlantObjectPtr pPlantObject;CcString strNode = L".hierarchy::PlantView\
\Unit1";

            //gets node for specified Node path
            errCode = pPlantModel->GetPlantObject(strNode.Get(), &pPlantObject);
            if (nullptr != pPlantObject && CF_SUCCEEDED(errCode))
            {

                CcString strName(L"NumberOfNodes");
                IPlantObjectPropertyPtr PlantObjectProperty;
                //get the PlantObject property by name
                pPlantObject->GetProperty(strName.Get(), &PlantObjectProperty);

                IPlantObjectPropertyValuePtr pPlantObjectPropertyValue;
                // Read PlantObject Property
                PlantObjectProperty->Read(&pPlantObjectPropertyValue);
                if (nullptr != pPlantObjectPropertyValue)
                {
                    CcString strName;
                    pPlantObjectPropertyValue->GetPlantObjectPropertyName(&strName);
                    std::cout << "PlantModelPropertyName:" << strName.ToUTF8() << std::endl;
                    int64_t qc;
                    pPlantObjectPropertyValue->GetQuality(&qc);
                    std::cout << "Quality:" << qc << std::endl;
                    int64_t ec;
                    pPlantObjectPropertyValue->GetError(&ec);
                    std::cout << "Error:" << ec << std::endl;
                    CcDateTime64 dt;
                    pPlantObjectPropertyValue->GetTimeStamp(&dt);
                    std::cout << "DateTime:" << dt.GetDateTimeString().ToUTF8() << std::endl;
                    CcVariant vtVal;
                    pPlantObjectPropertyValue->GetValue(&vtVal);
                    std::cout << "Value:" << vtVal.uint64 << std::endl;
                }
                else
                {
                    std::cout << L" GetProperties operation failed." << "errCode = " <<
errCode << std::endl;
                    PrintErrorInformation(errCode, L"GetProperties", pRuntime);
                }
            }
            else
            {
                std::cout << L"GetPlantObject operation failed." << "errCode = " << errCode
<< std::endl;

```

Copy code

```

        PrintErrorInformation(errCode, L"GetPlantObject", pRuntime);
    }
}
else
{
    std::cout << L"Error, couldn't create ODK object." << "errCode = " << errCode
<< std::endl;
}
}
}

```

9.6.4 IPlantObjectPropertyValue

Description

The C++ interface "IPlantObjectPropertyValue" specifies the handling of process tag properties of the Runtime system.

Members

The following methods are specified in the interface:

"GetPlantObjectPropertyName" method

Supplies the name of the tag.

```
CFRESULT GetPlantObjectPropertyName(CFSTR* pName)
```

- pName
[out]: Points to the name.

"GetValue" method

Supplies the tag value.

```
CFRESULT GetValue(CFVARIANT* pValue)
```

- pValue
[out]: Points to the process value.

"GetQuality" method

Supplies the quality code of the tag.

```
CFRESULT GetQuality(int64_t* pQualityCode)
```

- pQualityCode
[out]: Points to the quality code.

"GetTimeStamp" method

Supplies the time stamp of the last modification to the tag.

```
CFRESULT GetTimeStamp(CFDATE64* pTimeStamp)
```

- pTimeStamp
[out]: Points to the time stamp.

"GetError" method

Supplies the error code of the tag.

```
CFRESULT GetError(int64_t* pErrorCode)
```

- pErrorCode
[out]: Points to the error code.

Example

For example, see IPlantObjectProperty (Page 341).

9.6.5 IPlantModelPropertySubscriptionNotification

Description

The C++ interface "IPlantModelPropertySubscriptionNotification" defines methods for implementing asynchronous change monitoring of object instance properties. The methods are used by the C++ interface "ICpmNodePropertySet".

All the methods return CF_SUCCESS following successful execution.

The interface inherits the "ICfUnknown" interface.

Members

The following methods are specified in the interface:

"OnDataChanged" method

Callback method is called when a monitored object instance property is changed.

```
CFRESULT OnDataChanged(IPlantObjectPropertyValueEnumerator*  
pEnumerator)
```

- pEnumerator:
[in]: Points to the "IPlantObjectPropertyValueEnumerator" object that provides access to an enumeration of "IPlantObjectPropertyValue" instances.

Example

Register a PropertySet for monitoring:

Copy code

```

void PlantObjectSubscribePropertySet(IRuntimePtr pRuntime)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCcString(L"PlantModel"), &pUnk);
    {
        if (nullptr != pUnk && CF_SUCCEEDED(errCode))
        {
            IPlantModelPtr pPlantModel(pUnk);
            IPlantObjectPtr pPlantObject;
            CCcString strNode = L".hierarchy::PlantView\\Unit1\\Filler1";

            //gets node for specified Node path
            errCode = pPlantModel->GetPlantObject(strNode.Get(), &pPlantObject);
            if (nullptr != pPlantObject && CF_SUCCEEDED(errCode))
            {
                //empty property - so all node members should be read
                CCfVariant vtProperties;
                IPlantObjectPropertySetPtr pPlantObjectPropertySet;
                // get the PlantObject properties by property names
                errCode = pPlantObject->GetProperties(vtProperties,
                &pPlantObjectPropertySet);
                if (nullptr != pPlantObjectPropertySet && CF_SUCCEEDED(errCode))
                {
                    CPlantModelPropertySubscriptionNotification* pSubscribe = new
                    CPlantModelPropertySubscriptionNotification();
                    IPlantModelPropertySubscriptionNotification* pNotification = pSubscribe;
                    //Subscribe for all PlantObject properties errCode =
                    pPlantObjectPropertySet->Subscribe(pNotification);
                }
                else
                {
                    std::cout << L" GetProperties operation failed." << "errCode = " <<
                    errCode << std::endl;
                }
            }
            else
            {
                std::cout << L"GetPlantObject operation failed." << "errCode = " << errCode
                << std::endl;
            }
        }
    }
}

CFRESULT CFCALLTYPE
CPlantModelPropertySubscriptionNotification::OnDataChanged(IPlantObjectPropertyValueEnumer
ator* pEnumerator)
{
    if (nullptr != pEnumerator)
    {
        uint32_t nCount = 0;
        pEnumerator->Count(&nCount);
        for (int i = 0; i < (int32_t)nCount; i++)
        {
            pEnumerator->MoveNext();
            IPlantObjectPropertyValue* pPlantModelPropertyValue;

```

Copy code

```

    if (CF_SUCCEEDED(pEnumerator->Current(&pPlantModelPropertyValue)) && nullptr !=
        pPlantModelPropertyValue)
    {
        CCfString strName;
        pPlantModelPropertyValue->GetPlantObjectPropertyName(&strName);
        std::cout << "Name:" << strName.ToUTF8() << std::endl;
        int64_t qc;
        pPlantModelPropertyValue->GetQuality(&qc);
        std::cout << "Quality:" << qc << std::endl;
        int64_t ec;
        pPlantModelPropertyValue->GetError(&ec);
        std::cout << "Error:" << ec << std::endl;
        CCfDateTime64 dt;
        pPlantModelPropertyValue->GetTimeStamp(&dt);
        std::cout << "DateTime:" << dt.GetDateTimeString().ToUTF8() << std::endl;
        CCfVariant vtVal;
        pPlantModelPropertyValue->GetValue(&vtVal);
        std::cout << "Value:" << vtVal.uint64 << std::endl;
    }
}
}
return CF_SUCCESS;
}

```

9.6.6 IPlantObjectPropertyValueEnumerator

Description

The C++ interface "IPlantObjectPropertyValueEnumerator" specifies methods for handling the enumeration of "IPlantObjectPropertyValue" instances.

All the methods return `CF_SUCCESS` following successful execution.

The interface inherits the "ICfUnknown" interface.

Members

The following methods are specified in the interface:

"Current" method

Supplies the current element of the enumeration.

```
Current(IPlantObjectPropertyValue** ppItem)
```

- `ppItem`
[out]:

"MoveNext" method

Move to the next element of the enumeration.

```
CFRESULT MoveNext()
```

"Reset" method

Resets the current position in the enumeration to the first element.

```
CFRESULT Reset()
```

"MoveNext" then supplies the first element of the enumeration.

"Count" method

Output the size of the enumeration or the number of elements in an enumeration.

```
CFRESULT Count(uint32_t *pValue)
```

- `value`
[out]: Points to the value for the number of elements in the enumeration.

Example

For example, see `IPlantModelPropertySubscriptionNotification` (Page 344).

9.6.7 IPlantObjectPropertySet**Description**

The C++ interface "IPlantObjectPropertySet" specifies methods for optimized access to several "IPlantObjectProperty" instances of an "IPlantObject" instance of the Runtime system.

After initialization of the "IPlantObjectPropertySet" object, you have read/write access to multiple "IPlantObjectProperty" instances in one call. Simultaneous access has better performance and a lower communication load than single access to multiple properties.

The interface inherits from the "ICfDispatch" interface.

Members

The following methods are specified in the interface:

"GetContextID" method

Supplies an ID as additional identification feature of a property. "ContextId" can, for example, be used to recognize properties having the same name but from different systems.

Default value -1: "ContextId" is not used.

```
CFRESULT GetContextId(int32_t* pContextId)
```

- `pContextId`
[out]: "ContextId" of the property

"SetContextID" method

Sets an ID as additional identification feature of a property. "ContextId" can, for example, be used to recognize properties having the same name but from different systems.

Default value -1: "ContextId" is not used.

```
CFRESULT SetContextId(int32_t contextId)
```

- contextId
[in]: "ContextId" of the property

"Read" method

Supplies all values of the "IPlantObjectProperty" instances contained in the "IPlantObjectPropertySet" instance. The values are read synchronously.

```
CFRESULT Read(IPlantObjectPropertyValueEnumerator**  
ppPlantObjectPropertyValueEnumerator)
```

- ppPlantObjectPropertyValueEnumerator
[out]: Points to the enumeration of the tag values as an "IPlantObjectPropertyValueEnumerator" object.

"ReadAsync" method

Reads the values of all "IPlantObjectProperty" instances of the "IPlantObjectPropertySet" instance asynchronously.

```
CFRESULT ReadAsync(IPlantObjectPropertySetReadReply* pReadReply)
```

- pReadReply
[in]: Points to the "IPlantObjectPropertySetReadReply" object that implements the callback interface for read operations.

"Write" method

Writes the values of the "IPlantProperty" instances of the "PlantObjectPropertySet" instance synchronously to the Runtime system.

```
CFRESULT Write(HmiUnified::Rt::IErrorResultEnumerator**  
ppEnumerator)
```

- pWriteReply
[out]: Points to an "IErrorResultEnumerator" object that contains the enumeration with errors for the write operations of the "IPlantObjectProperty" instances.

"WriteAsync" method

Writes the values of all "IPlantObjectProperty" instances of the "PlantObjectPropertySet" instance asynchronously to the Runtime system.

```
CFRESULT WriteAsync(IPlantObjectPropertySetWriteReply *pWriteReply)
```

- pWriteReply
[in]: Points to the "IPlantObjectPropertySetWriteReply" object that implements the callback interface for write operations.

"GetCount" method

Supplies the number of properties of the "IPlantObjectPropertySet" instance.

```
CFRESULT GetCount(uint32_t *pCount)
```

- pCount
[out]: Points to the number of properties.

"Subscribe" method

Subscribes all properties of the "IPlantObjectPropertySet" instance asynchronously for change monitoring.

```
CFRESULT Subscribe(IPlantModelPropertySubscriptionNotification*
pPlantModelSubscriptionCallback)
```

- `pPlantModelSubscriptionCallback`
[in]: Points to the "IPlantObjectPropertySubscriptionNotification" object that implements the callback interface of the change monitoring.

"CancelSubscribe" method

Cancels change monitoring for all properties of the "IPlantObjectPropertySet" instance.

```
CFRESULT CancelSubscribe()
```

"Add" method

Adds an "IPlantObjectProperty" instance with property value or more "IPlantObjectProperty" instances to the "IPlantObjectPropertySet" instance.

```
CFRESULT Add(ICfArrayVariantPtr propertyNames)
```

- `propertyNames`
[in]: Array with names of several "IPlantObjectProperty" instances

or

```
CFRESULT Add(const CFSTR propertyName, const CFVARIANT value)
```

- `propertyName`
[in]: Name of the "IPlantObjectProperty" instance
- `value`
[in]: New process value of the "IPlantObjectProperty" instance

"Remove" method

Removes a property from the "IPlantObjectPropertySet" instance.

```
CFRESULT Remove(const CFSTR propertyName)
```

- `propertyName`
[in]: Name of the property that is being removed.

"Clear" method

Removes all properties from the "IPlantObjectPropertySet" instance.

```
CFRESULT Clear()
```

New example**Copy code**

```

void PlantObjectWriteAsyncPropertySet(IRuntimePtr pRuntime)
{
    CFRESULT hr = CF_ERROR;
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CcFString(L"PlantModel"), &pUnk);
    {
        if (nullptr != pUnk && CF_SUCCEEDED(errCode))
        {
            IPlantModelPtr pPlantModel(pUnk);
            IPlantObjectPtr pPlantObject;
            CcFString strNode = L".hierarchy::PlantView\\Unit1";

            //gets node for specified Node path
            errCode = pPlantModel->GetPlantObject(strNode.Get(), &pPlantObject);
            if (nullptr != pPlantObject && CF_SUCCEEDED(errCode))
            {
                //empty property - so all node members should be read
                CcFVariant vtProperties;
                IPlantObjectPropertySetPtr pPlantObjectPropertySet;
                // get the PlantObject properties by property names
                errCode = pPlantObject->GetProperties(vtProperties,
                &pPlantObjectPropertySet);
                if (nullptr != pPlantObjectPropertySet && CF_SUCCEEDED(errCode))
                {
                    CcFString strName(L"NumberOfNodes");
                    CcFVariant vtValue(1000);
                    hr = pPlantObjectPropertySet->Add(strName, vtValue);
                    if (CF_SUCCEEDED(hr))
                    {
                        CPlantObjectPropertySetWriteReply* pReply = new
CPlantObjectPropertySetWriteReply();
                        IPlantObjectPropertySetWriteReplyPtr pWritReply = pReply;

                        //Write PlantObject properties values asynchronously
                        pPlantObjectPropertySet->WriteAsync(pWritReply);
                    }
                }
                else
                {
                    std::cout << L" GetProperties operation failed." << "errCode = " <<
errCode << std::endl;
                }
            }
            else
            {
                std::cout << L"GetPlantObject operation failed." << "errCode = " <<
errCode << std::endl;
            }
        }
        else
        {
            std::cout << L"Error, couldn't create ODK object." << "errCode = " << errCode
<< std::endl;
        }
    }
}

```

Copy code

```
}  
}
```

9.6.8 IPlantObjectPropertySetReadReply

Description

The C++ interface "IPlantObjectPropertySetReadReply" defines the "OnReadComplete" method as callback method of a "ReadAsync" call. The method is used by the C++ interface "IPlantObjectPropertySet".

Members

The following methods are specified in the interface:

"OnReadComplete" method

The "OnReadComplete" callback method is called when the "ReadAsync" method is used.

```
CFRESULT OnReadComplete(CFVARIANT systemError,  
IPlantObjectPropertyValueEnumerator*  
pPlantObjectPropertyValueEnumerator)
```

- `systemError`
[in]: Supplies an error code when a global error has occurred.
- `ppPlantObjectPropertyValueEnumerator`
[in]: Points to an "IPlantObjectPropertyValueEnumerator" object that contains the enumeration of the process values of object instance properties.

Example

Copy code

```
void PlantObjectReadAsyncPropertySet(IRuntimePtr pRuntime)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCFString(L"PlantModel"), &pUnk);
    {
        if (nullptr != pUnk && CF_SUCCEEDED(errCode))
        {
            IPlantModelPtr pPlantModel(pUnk);
            IPlantObjectPtr pPlantObject;
            CCFString strNode = L".hierarchy::PlantView\\Unit1";

            //gets node for specified Node path
            errCode = pPlantModel->GetPlantObject(strNode.Get(), &pPlantObject);
            if (nullptr != pPlantObject && CF_SUCCEEDED(errCode))
            {
                //empty property - so all node members should be read
                CCFVariant vtProperties;
                IPlantObjectPropertySetPtr pPlantObjectPropertySet;

                // get the PlantObject properties by property names
                errCode = pPlantObject->GetProperties(vtProperties,
                &pPlantObjectPropertySet);
                if (nullptr != pPlantObjectPropertySet && CF_SUCCEEDED(errCode))
                {
                    CPlantObjectPropertySetReadReply* pReply = new
                    CPlantObjectPropertySetReadReply();
                    IPlantObjectPropertySetReadReplyPtr pReadReply = pReply;

                    // Read PlantObject properties values asynchronously
                    pPlantObjectPropertySet->ReadAsync(pReadReply);
                }
                else
                {
                    std::cout << L" GetProperties operation failed." << "errCode = "
                    << errCode << std::endl;
                }
            }
            else
            {
                std::cout << L"GetPlantObject operation failed." << "errCode = " <<
                errCode << std::endl;
            }
        }
        else
        {
            std::cout << L"Error, couldn't create ODK object." << "errCode = " << errCode
            << std::endl;
        }
    }
}

CFRESULT CFCALLTYPE CPlantObjectPropertySetReadReply::OnReadComplete(IN CFVARIANT
SystemError, IN IPlantObjectPropertyValueEnumerator* pPlantObjectPropertySet)
{
    if (nullptr != pPlantObjectPropertySet)
```

Copy code

```

{
    uint32_t nCount = 0; pPlantObjectPropertySet->Count(&nCount);
    for (int i = 0; i < (int32_t)nCount; i++)
    {
        pPlantObjectPropertySet->MoveNext();
        IPlantObjectPropertyValue* pPlantModelPropertyValue;
        if (CF_SUCCEEDED(pPlantObjectPropertySet->Current(&pPlantModelPropertyValue)) && nullptr != pPlantModelPropertyValue)
        {
            CCfString strName;
            pPlantModelPropertyValue->GetPlantObjectPropertyName(&strName);
            std::cout << "Name:" << strName.ToUTF8() << std::endl;
            int64_t qc;
            pPlantModelPropertyValue->GetQuality(&qc);
            std::cout << "Quality:" << qc << std::endl;
            int64_t ec;
            pPlantModelPropertyValue->GetError(&ec);
            std::cout << "Error:" << ec << std::endl;
            CCfDateTime64 dt; pPlantModelPropertyValue->GetTimeStamp(&dt);
            std::cout << "DateTime:" << dt.GetDateTimeString().ToUTF8() <<
std::endl;

            CCfVariant vtVal;
            pPlantModelPropertyValue->GetValue(&vtVal);
            std::cout << "Value:" << vtVal.uint64 << std::endl;
        }
    }
}
return CF_SUCCESS;
}

```

9.6.9 IPlantObjectPropertySetWriteReply**Description**

The C++ interface "IPlantObjectPropertySetWriteReply" defines the "OnWriteComplete" method as callback method of a "WriteAsync" call. The method is used by the C++ interface "IPlantObjectPropertySet".

Members

The following methods are specified in the interface:

"OnWriteComplete" method

The "OnWriteComplete" callback method is called when the "WriteAsync" method is used.

```
CFRESULT OnWriteComplete(CFVARIANT systemError,
    IPlantObjectPropertyValueEnumerator*
    pPlantObjectPropertyValueEnumerator) P
```

- `systemError`
[in]: Supplies an error code when a global error has occurred.
- `ppPlantObjectPropertyValueEnumerator`
[in]: Points to an "IPlantObjectPropertyValueEnumerator" object that contains the enumeration of the process values of "IPlantObjectProperty" instances.

Example

Copy code

```
CFRESULT CFCALLTYPE CPlantObjectPropertySetWriteReply::OnWriteComplete(IN CFVARIANT
SystemError, IN IPlantObjectPropertyValueEnumerator* pPlantObjectPropertySet)
{
    if (nullptr != pPlantObjectPropertySet)
    {
        uint32_t nCount = 0;
        pPlantObjectPropertySet->Count(&nCount);
        for (int i = 0; i < (int32_t)nCount; i++)
        {
            pPlantObjectPropertySet->MoveNext();
            IPlantObjectPropertyValue* pPlantModelPropertyValue;
            if (CF_SUCCEEDED(pPlantObjectPropertySet->
>Current(&pPlantModelPropertyValue)) && nullptr !=
pPlantModelPropertyValue)
            {
                CCfString strName;
                pPlantModelPropertyValue->GetPlantObjectPropertyName(&strName);
                std::cout << "Name:" << strName.ToUTF8() << std::endl;
                int64_t qc;
                pPlantModelPropertyValue->GetQuality(&qc);
                std::cout << "Quality:" << qc << std::endl;
                int64_t ec;
                pPlantModelPropertyValue->GetError(&ec);
                std::cout << "Error:" << ec << std::endl;
                CCfDateTime64 dt;
                pPlantModelPropertyValue->GetTimeStamp(&dt);
                std::cout << "DateTime:" << dt.GetDateTimeString().ToUTF8() <<
std::endl;

                CCfVariant vtVal;
                pPlantModelPropertyValue->GetValue(&vtVal);
                std::cout << "Value:" << vtVal.uint64 << std::endl;
            }
        }
    }
    return CF_SUCCESS;
}
```

9.6.10 IPlantObjectEnumerator

Description

The "IPlantObjectEnumerator" interface is a C++ interface that specifies methods for handling the enumeration of object instances of the plant model of a Runtime system.

All the methods return CF_SUCCESS following successful execution.

The interface inherits from the "ICfDispatch" interface.

Members

The following methods are specified in the interface:

"Current" method

Supplies the current element of the enumeration.

```
CFRESULT Current(IPlantObject **ppItem)
```

- ppItem
[out]: Points to the current "IPlantObject" object as an element of the list.

"MoveNext" method

Move to the next element of the enumeration.

```
CFRESULT MoveNext()
```

"Reset" method

Resets the current position in the enumeration to the first element.

```
CFRESULT Reset()
```

"MoveNext" then supplies the first element of the enumeration.

"Count" method

Output the size of the enumeration or the number of elements in an enumeration.

```
CFRESULT Count(uint32_t *pValue)
```

- value
[out]: Points to the value for the number of elements in the enumeration.

Example

Copy code

```
void PlantObjectCurrentPlantViewPath(IRuntimePtr pRuntime)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCfString(L"PlantModel"),
    &pUnk);
    if (pUnk != nullptr && CF_SUCCEEDED(errCode))
    {
        IPlantModelPtr pPlantModel(pUnk);
        IPlantObjectEnumeratorPtr pPlantObjectEnum;
        CCfString str1 = L"NumberOfNodes";
        CCfString str2 = L"Quality";
        CCfString str3 = L"Quantity";
        CCfString str4 = L"DateActivation";
        CCfSafeArray daSafeArray(CF_VT_STR, 1U, 4U);
        int32_t index = 0;
        daSafeArray.PutElement(&index, (void*)&str1);
        ++index;
        daSafeArray.PutElement(&index, (void*)&str2);
        ++index;
        daSafeArray.PutElement(&index, (void*)&str3);
        ++index;
        daSafeArray.PutElement(&index, (void*)&str4);
        CCfVariant vtProperties;
        daSafeArray.Detach(&vtProperties);

        // gets PlantObjects for specified filter
        errCode = pPlantModel->GetPlantObjectsByExpression(vtProperties,
        CCfString(L"BLOWER").Get(),
        CCfString(L"NumberOfNodes>=0"), CCfString
        (L"RUNTIME_1.hierarchy::PlantView\\Unit1\\Blower1").Get
        (), &pPlantObjectEnum);
        if (pPlantObjectEnum != nullptr && CF_SUCCEEDED(errCode))
        {
            pPlantObjectEnum->MoveNext();
            IPlantObject* pPlantObject;
            while (CF_SUCCEEDED(pPlantObjectEnum-
            >Current(&pPlantObject)))
            {
                displayNodeInfo(pPlantObject);
                pPlantObjectEnum->MoveNext();
            }
        }
        else
        {
            std::cout << L"GetPlantObjectsByExpression operation failed."
            << "errCode = " << errCode << std::endl;
        }
    }
    else
    {
        std::cout << L"Error, couldn't create ODK object." << "errCode = "
        << errCode << std::endl;
    }
}
```

Copy code

}

9.6.11 IPlantObjectAlarmSubscription

Description

The C++ interface "IPlantObjectAlarmSubscription" specifies methods for starting and stopping an "AlarmSubscription".

The interface inherits from the "ICfDispatch" interface.

Members

The following methods are specified in the interface:

"Start" method

Subscribe systems for monitoring of changes of active alarms.

```
CFRESULT Start(IPlantObjectAlarmSubscriptionCallback* callbackPtr)
```

- `callbackPtr`
[in]: Points to the "IPlantObjectAlarmSubscriptionCallback" object that implements the callback interface of the change monitoring.

"Stop" method

Unsubscribe monitoring of active alarms.

```
CFRESULT Stop(void)
```

"GetFilter" method

Supplies the string by which the result set is filtered.

```
CFRESULT GetFilter(CFSTR* filter)
```

- `filter`
[out]: SQL-type string for filtering the result set of active alarms.

"SetFilter" method

Sets the string for filtering the result set of active alarms.

```
CFRESULT SetFilter(IN CFSTR filter)
```

- `filter`
[in]: SQL-type string for filtering the result set of active alarms.
All properties of an alarm can be used in the filter string. The filter string can contain operators. Refer to the section Syntax of the alarm filter (Page 19).

"GetLanguage" method

Supplies the country identifier of the language of the monitored alarms.

```
CFRESULT GetLanguage(uint32_t* language)
```

- language
[out]: Code of the country identification. See also section Locale IDs of the supported languages (Page 21).

"SetLanguage" method

Sets the country identifier of the language of the monitored alarms.

```
CFRESULT SetLanguage)(uint32_t language)
```

- language
[in]: Code of the country identification See also section Locale IDs of the supported languages (Page 21).

"GetIncludeChildren" method

Supplies the setting for the child instances.

```
CFRESULT GetIncludeChildren(CFBOOL* bIsIncludeChildren)
```

- bIsIncludeChildren
[out]: Reads out whether the child instances are part of monitoring.

"SetIncludeChildren" method

Determines the setting for the child instances.

```
CFRESULT SetIncludeChildren(CFBOOL bIsIncludeChildren)
```

- bIsIncludeChildren
[in]: Controls whether the child instances are part of monitoring.

Example

Copy code

```

void PlantObjectSubscription(IRuntimePtr pRuntime)
{
    if (nullptr != pRuntime)
    {
        ICfUnknownPtr pUnk;
        CFRESULT errCode = pRuntime->GetObject(CCfString(L"PlantModel").Get(), &pUnk);
        if (CF_SUCCEEDED(errCode) && nullptr != pUnk)
        {
            IPlantModelPtr pPlantModel(pUnk);
            CAlarmValue* pAlarmValue = new CAlarmValue();
            pAlarmValue->AddRef();
            IPlantObjectAlarmCallback *pCB = pAlarmValue;
            CCfString strNode = L".hierarchy::RootNodeName\\Node1";
            IPlantObjectPtr pPlantObject;
            errCode = pPlantModel->GetPlantObject(strNode.Get(), &pPlantObject);
            if (pPlantObject != nullptr && CF_SUCCEEDED(errCode))
            {
                CCfSmartString daFilter; //= L"AlarmClassName = 'Alarm'";
                uint32_t daLanguage = 1033;
                //IPlantObjectAlarmCallbackPtr pAlarmCallback;
                IPlantObjectAlarmSubscriptionPtr pPlantObjectAlarmSubscription;
                errCode = pPlantObject->
                >CreateAlarmSubscription(&pPlantObjectAlarmSubscription);
                if (errCode == CF_SUCCESS && nullptr != pPlantObjectAlarmSubscription)
                {
                    pPlantObjectAlarmSubscription->SetFilter(daFilter.AllocCFSTR());
                    pPlantObjectAlarmSubscription->SetLanguage(1033);
                    pPlantObjectAlarmSubscription->SetFilter(false);
                    pPlantObjectAlarmSubscription->SetIncludeChildren(false);
                    CCfRefPtr<CPlantObjectAlarmSubscriptionCallback> pCallback =
                                                                new
CPlantObjectAlarmSubscriptionCallback();
                    pPlantObjectAlarmSubscription->Start(pCallback);
                }
            }
        }
    }
}

```

9.6.12 IPlantObjectAlarmCallback

Description

The C++ interface "IPlantObjectAlarmCallback" defines the callback method "OnAlarm".

Member

The following methods are defined in the interface:

"OnAlarm" method

Specifies the signature of the event handling method for the "OnAlarm" event of an "IPlantObject" instance.

```
OnAlarm(uint32_t systemError,  
        CFSTR  systemName,
```

```
        Siemens::Runtime::HmiUnified::Alarms::IAlarmResultEnumerator*  
        pItems,  
        CFBOOL  completed)
```

- `systemError`
Supplies an error code when a global error has occurred. When the error code is set, `pItems` is irrelevant.
- `systemName`
Name of the Runtime system that is subscribed for alarm monitoring by the user.
- `pItems`
Supplies a pointer to "IAlarmResultEnumerator" that can be used to enumerate the active alarms.
- `completed`
Status of the asynchronous transfer:
 - `True`: All alarms are read out.
 - `False`: Not all alarms are yet read out.

Example

Copy code

```
CFRESULT CFCALLTYPE CPlantModelAlarmValue::OnAlarm( uint32_t systemError, CFSTR
systemName, Siemens::Runtime::HmiUnified::Alarms::IAlarmResultEnumerator* pItems, CFBOOL
completed)
{
    CFRESULT hr = CF_FALSE;
    if (!completed)
    {
        uint32_t nsize;
        pItems->Count(&nsize);
        if (nsize > 0 && CF_SUCCEEDED(systemError))
        {
            while (CF_SUCCEEDED(pItems->MoveNext()))
            {
                IAlarmResultPtr ppValues;
                if (CF_SUCCEEDED(pItems->Current(&ppValues)))
                {
                    CCfString strId;
                    ppValues->GetSourceID(&strId);
                    std::cout << "String ID = " << strId.ToUTF8().c_str() << std::endl;
                    CCfString strName;
                    ppValues->GetName(&strName);
                    std::cout << "Name = " << strName.ToUTF8().c_str() << std::endl;

                    CCfString strClassName;
                    ppValues->GetAlarmClassName(&strClassName);
                    std::cout << "Alarm Class Name = " << strClassName.ToUTF8().c_str()
<< std::endl;
                }
            }
        }
        else
        {
            this->SetEvent();
        }
        hr = CF_SUCCESS;
        return hr;
    }
}
```

9.6.13 IPlantObjectAlarmSubscriptionCallback

Description

The C++ interface "IPlantObjectAlarmCallbackSubscription" defines the callback method "OnAlarm".

The interface inherits the "ICfUnknown" interface.

Members

The following methods are specified in the interface:

"OnAlarm" method

Specifies the signature of the event handling method for the "OnAlarm" event of an "IPlantObject" instance.

```
OnAlarm(uint32_t systemError,  
        CFSTR systemName,
```

```
        Siemens::Runtime::HmiUnified::Alarms::IAlarmResultEnumerator*  
        pItems)
```

- `systemError`
Supplies an error code when a global error has occurred. When the error code is set, `pItems` is irrelevant.
- `systemName`
Name of the Runtime system that is subscribed for alarm monitoring by the user.
- `pItems`
 - Supplies a pointer to "IAlarmResultEnumerator" that can be used to enumerate the active alarms.

9.7 Interfaces of the Calendar option

9.7.1 ISHCCalendarOption

Description

The C++ interface "ISHCCalendarOption" specifies the "GetObject" method. The method supplies the calendar object of an "IPlantObject" instance. A calendar is always integrated via an "IPlantObject" instance.

Members

"GetObject" method

Supplies an error code of the type `CFRESULT`.

9.7 Interfaces of the Calendar option

CFRESULT

```
GetObject(Siemens::Runtime::HmiUnified::PlantModel::IPlantObject *  
pCpmNode, CFSTR objectName, ICfUnknown** ppObject)
```

- cpmNode
[in]: Reference to the "IPlantObject" instance currently selected in the hierarchy
- objectName
[in]: The name of the "IPlantObject" instance
- ppObject
[out]: Returns an object of the type "ICfUnknown". The object is cast to a calendar object using the "QueryInterface" method.

Example:

```
ISHCCalendarOptionPtr pShcOption;  
ISHCCalendarPtr pCalendar;  
pRuntime->GetOption(CCfString(ODK_SHC_OPTION),  
(IOption**) &pShcOption);  
if (nullptr != pShcOption)  
{  
    ICfUnknownPtr pUnk;  
    pShcOption->GetObject(pPlantObject, ODK_SHC_CALENDAR, &pUnk);  
    if (nullptr != pUnk)  
    {  
        pUnk->QueryInterface(IID_ISHCCalendar,  
(ICfUnknown**) &pCal);  
    }  
}
```

Example

The following example serves as a basis for the other examples for the C++ interfaces of the Calendar option.

It shows how you can obtain the "IPlantObject" instance and also an "ISHCCalendar" instance. The "ISHCCalendar" instance referenced via `pCalendar` is also used in the other examples.

Copy code

```

#include <iostream>
#include "IODkShcInterface.h"
#include "IODkRt.h"
#include "IODkRtAlarm.h"
using namespace Siemens::Runtime::HmiUnified::Rt;
using namespace Siemens::Runtime::HmiUnified::Common;
using namespace Siemens::Runtime::HmiUnified::PlantModel;
IRuntimePtr pRuntime;
ISHCCalendarPtr pCalendar;
ISHCCalendarSettingsProviderPtr pCalendarProvider;
ISHCCategoryProviderPtr pShcCategoryProvider;
ISHCShiftTemplatesProviderPtr pShcShiftTemplateProvider;
ISHCDayProviderPtr pShcDayProvider;
ISHCDayTemplatesProviderPtr pShcDayTemplateProvider;
ISHCActionTemplatesProviderPtr pShcActionTemplateProvider;
CCfString projectName = L"";
if (CF_SUCCEEDED(Connect(projectName, &pRuntime)))
{
    ICfUnknownPtr pPlantModelUnk;
    CFRESULT errCode = pRuntime->GetObject(CCfString(L"PlantModel"), &pPlantModelUnk);
    IPlantObjectPtr pPlantObject;
    if (pPlantModelUnk != nullptr && CF_SUCCEEDED(errCode))
    {
        IPlantModelPtr pPlantModel(pPlantModelUnk);
        CCfString strNode = L".hierarchy::Plant/Unit1";
        //gets Object for specified Node path
        errCode = pPlantModel->GetPlantObject(strNode.Get(), &pPlantObject);
        if (CF_SUCCEEDED(errCode) && pPlantObject != nullptr)
        {
            IOptionPtr pOdkOption;
            pRuntime->GetOption(CCfString(ODK_SHC_OPTION), &pOdkOption);
            if (nullptr != pOdkOption)
            {
                ISHCCalendarOptionPtr pShcOption;
                pOdkOption->QueryInterface(IID_ISHCCalendarOption,
                (ICfUnknown**) &pShcOption);
                if (nullptr != pShcOption)
                {
                    ICfUnknownPtr pUnk;
                    pShcOption->GetObject(pPlantObject, ODK_SHC_CALENDAR, &pUnk);
                    if (nullptr != pUnk)
                    {
                        pUnk->QueryInterface(IID_ISHCCalendar, (ICfUnknown**) &pCalendar);
                        if (pCalendar != nullptr)
                        {
                            // Get all data provider
                            pCalendar->GetActionTemplatesProvider(&pShcActionTemplateProvider);
                            pCalendar->GetCategoryProvider(&pShcCategoryProvider);
                            pCalendar->GetDayProvider(&pShcDayProvider);
                            pCalendar->GetDayTemplateProvider(&pShcDayTemplateProvider);
                            pCalendar->GetShiftTemplateProvider(&pShcShiftTemplateProvider);
                            pCalendar->GetSettings(&pCalendarProvider);
                        }
                    }
                }
            }
        }
    }
}

```

Copy code

```
    }  
    }  
    }  
}
```

9.7.2 ISHCCalendar

Description

The C++ interface "ISHCCalendar" specifies the methods of a calendar.
The interface inherits from the "ICfUnknown" interface.

Members

"GetSettings" method

Supplies the "ISHCCalendarSettings" instances of the calendar.

```
CFRESULT GetSettings(ISHCCalendarSettings** calendar)
```

- calendar
[out]: The "ISHCCalendarSettings" instance

"GetCategoryProvider" method

Supplies an "ISHCCategoryProvider" instance. The provider enables access to the "ISHCategory" instances of the calendar.

```
CFRESULT GetCategoryProvider(ISHCCategoryProvider**  
ppCategoryProvider)
```

- ppCategoryProvider
[out]: The "ISHCCategoryProvider" instance

"GetShiftTemplateProvider" method

Supplies an "ISHCShiftTemplatesProvider" instance. The provider enables access to the "ISHCShiftTemplate" instances of the calendar.

```
CFRESULT GetShiftTemplateProvider(ISHCShiftTemplatesProvider**  
ppShiftTemplateProvider)
```

- ppShiftTemplateProvider
[out]: The "ISHCShiftTemplatesProvider" instance

"GetDayTemplateProvider" method

Supplies an "ISHCDayTemplatesProvider" instance. The provider enables access to the "ISHCShiftTemplate" instances of the calendar.

```
CFRESULT GetDayTemplateProvider(ISHCDayTemplatesProvider**
ppDayTemplateProvider)
```

- ppDayTemplateProvider
[out]: The "ISHCDayTemplatesProvider" instance

"GetActionTemplatesProvider" method

Supplies an "ISHCActionTemplatesProvider" instance. The provider enables access to the "ISHCActionTemplate" instances of the calendar.

```
CFRESULT GetActionTemplatesProvider(OUT
ISHCActionTemplatesProvider** ppActionTemplatesProvider)
```

- ppActionTemplatesProvider
[out]: The "ISHCActionTemplatesProvider" instance

"GetDayProvider" method

Supplies an "ISHCDayProvider" instance. The provider enables access to the "ISHCDay" instances of the calendar.

```
CFRESULT GetDayProvider(ISHCDayProvider** ppDayProvider)
```

- ppDayProvider
[out]: The "ISHCDayProvider" instance

"GetObject" method

Creates an instance of the type defined in `value` and supplies a reference to the instance.

```
CFRESULT GetObject(const CFSTR value, ICfUnknown** ppObject)
```

- value
[in]: Possible values:
 - "ODK_SHC_OPTION"
 - "ODK_SHC_CALENDAR"
 - "ODK_SHC_TIME_SLICE"
 - "ODK_SHC_DAY_TEMPLATE"
 - "ODK_SHC_DAY"
 - "ODK_SHC_DAY_TEMPLATE"
 - "ODK_SHC_ACTION_TEMPLATE"
 - "ODK_SHC_ACTION_TEMPLATE_ELEMENT"
- ppObject
[out]: Reference to an object of the type "ICfUnknown", which can be cast to the corresponding type.

Example:

```
ICfUnknownPtr pUnk;
ISHCShiftTemplatePtr pShcShiftTemplate;
pCalendar->GetObject(ODK_SHC_SHIFT_TEMPLATE, &pUnk);
pShcShiftTemplate = (ISHCShiftTemplatePtr)pUnk;
```


9.7.3 ISHCCalendarSettings

Description

The C++ interface "ISHCCalendarSettings" specifies methods for the calendar settings of an "ISHCCalendar" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"GetPlantObject" method

Supplies the name of the "IPlantObject" instance to which the calendar belongs.

`GetPlantObject (CFSTR* pCpmNode)`

- `pCpmNode`
[out]: The name of the "IPlantObject" instance

"GetFirstDayOfWeek" method

Supplies the first day of the week.

`CFRESULT GetFirstDayOfWeek (CFENUM * pvarRet)`

- `pvarRet`
Points to the enumeration "ShcWeekDay", which can contain the following values:
 - Sunday (0)
 - Monday (1)
 - Tuesday (2)
 - Wednesday (3)
 - Thursday (4)
 - Friday (5)
 - Saturday (6)

"GetFirstWeekOfYear" method

Supplies the first week of the year.

`CFRESULT GetFirstWeekOfYear (CFENUM * pvarRet)`

- `pvarRet`
[out]: Points to the enumeration "ShcWeekStart", which can contain the following values:
 - FirstOfJanuary (0): The first week starts on the first of January.
 - AtLeastFourDays (1): The first week must have at least four days.
 - WholeWeek (2): The first week must have at least seven days.

"GetFiscalYearStartDay" method

Supplies the first day of the fiscal year.

9.7 Interfaces of the Calendar option

```
CFRESULT GetFiscalYearStartDay(uint8_t* pvarRet)
```

- pvarRet
[out]: The first day of the fiscal year of the calendar.

"GetFiscalYearStartMonth" method

Supplies the first month of the fiscal year.

```
CFRESULT GetFiscalYearStartMonth(uint8_t* pvarRet)
```

- pvarRet
[out]: The first month of the fiscal year of the calendar.

"GetDayOffset" method

Supplies the offset with which the workday begins, calculated from midnight.

```
CFRESULT GetDayOffset(CFTIMESPAN64 * pvarRet)
```

- pvarRet
[out]: Supplies the number of hours after midnight with which the day begins.

"GetWorkDays" method

Supplies the workdays of the calendar.

```
CFRESULT GetWorkDays(uint8_t* pvarRet)
```

- pvarRet
[out]: The workdays of the calendar.

"GetTimeZone" method

Supplies the time zone of the calendar.

```
CFRESULT GetTimeZone(uint32_t* pTimeZone)
```

- pvarRet
[out]: The time zone ID of Microsoft.

9.7.4 ISHCCategory

Description

The C++ interface "ISHCCategory" specifies the methods of a time category of an "ISHCCalendar" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"GetName" method

Supplies the name of the "ISHCCategory" instance.

```
CFRESULT GetName(CFSTR * pvarRet)
```

- pvarRet
[out]: The name

"GetDescriptions" method

Supplies a map with the descriptions of the "ISHCCategory" instance and their language code IDs.

```
CFRESULT GetDescriptions(OUT ICfMapIDToVariant** ppDisplayNames)
```

- ppDisplayNames
[out]: A map with int32/string pairs (language code ID for description).

Example:

```
ICfMapIDToVariantPtr pDescriptions;  
hr = pShcCategory->GetDescriptions(&pDescriptions);  
if (pDescriptions != nullptr && CF_SUCCEEDED(hr))  
{  
    std::cout << "Descriptions::" << std::endl << std::endl;  
    uint32_t nCount2 = 0;  
    pDescriptions->Count(&nCount2);  
    for (uint32_t nIndex2 = 0; nIndex2 < nCount2; nIndex2++)  
    {  
        int32_t nLanguageID; pDescriptions->KeyAt(nIndex2,  
&nLanguageID);  
        CCfVariant strDescription;  
        pDescriptions->ValueAt(nLanguageID, &strDescription);  
        std::cout << "LanguageID =" << nLanguageID << "  
Description=" << CCfSmartString(strDescription).ToUTF8().c_str() <<  
std::endl;  
    }  
}
```

"GetDisplayNames" method

Supplies a map with the display names of the "ISHCCategory" instance and their language code IDs.

```
CFRESULT GetDisplayNames(ICfMapIDToVariant** ppDisplayNames)
```

- ppDisplayNames
[out]: A map with int32/string pairs (language code ID for display name).

Example: Similar to "GetDescriptions"

"GetColor" method

Supplies the color of the "ISHCCategory" instance.

```
CFRESULT GetColor(uint32_t* pColor)
```

- pColor
[out]: Returns a 4-byte value for an RGBA color value.

"GetIsDeleted" method

Supplies the information on whether the "ISHCCategory" instance was deleted in Engineering.

```
CFRESULT GetIsDeleted(CFBOOL* p_bIsDeleted) = 0;
```

- `p_bIsDeleted`
[out]:
 - 0: Was not deleted (default)
 - 1: Was deleted

See also

Locale IDs of the supported languages (Page 21)

9.7.5 ISHCCategoryEnumerator

Description

The C++ interface "ISHCCategoryEnumerator" specifies methods for handling the enumeration of the time categories of an "ISHCCalendar" instance. The enumeration is returned by the `Read` method of an "ISHCCategoryProvider" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"MoveNext" method

Go to the next element of the enumeration.

```
CFRESULT MoveNext()
```

"Current" method

Output the current element of the enumeration.

```
CFRESULT Current(ISHCCategory** ppItem)
```

- `ppItem`
[out]: The current "ISHCCategory" instance

"Reset" method

Reset the current position in the enumeration. The "MoveNext" method moves afterwards to the first element.

```
CFRESULT Reset()
```

"Count" method

Output the size of the enumeration or the number of its elements.

```
CFRESULT Count(uint32_t* pCount)
```

- `pCount`
[out]: Number of categories

Example

Copy code

```
void PrintCategory(const ISHCCategoryEnumeratorPtr& p_pShcCategoryEnum)
{
    std::cout << std::endl << "*****PrintCategory*****" << std::endl << endl;
    if (p_pShcCategoryEnum != nullptr)
    {
        uint32_t nCount = 0;
        p_pShcCategoryEnum->Count(&nCount);
        for (uint32_t nIndex = 0; nIndex < nCount; nIndex++)
        {
            if (CF_SUCCEEDED(p_pShcCategoryEnum->MoveNext()))
            {
                ISHCCategoryPtr pShcCategory;
                p_pShcCategoryEnum->Current(&pShcCategory);
                if (pShcCategory != nullptr)
                {
                    cout << endl;
                    CFRESULT hr = CF_ERROR;
                    CCfString strName;
                    hr = pShcCategory->GetName(&strName);
                    if (CF_SUCCEEDED(hr) && (!strName.IsEmpty()))
                    {
                        cout << "CategoryName= " << strName.ToUTF8().c_str() << endl;
                    }
                    uint32_t nColor;
                    hr = pShcCategory->GetColor(&nColor);
                    if (CF_SUCCEEDED(hr))
                    {
                        cout << "Color= " << nColor << endl;
                    }
                    CFBOOL bIsDeleted;
                    hr = pShcCategory->GetIsDeleted(&bIsDeleted);
                    if (CF_SUCCEEDED(hr))
                    {
                        cout << "IsDeleted= " << (uint32_t)bIsDeleted << endl;
                    }
                    ICfMapIDToVariantPtr pDisplayNames;
                    hr = pShcCategory->GetDisplayNames(&pDisplayNames);
                    if (pDisplayNames != nullptr && CF_SUCCEEDED(hr))
                    {
                        std::cout << "DisplayNames::" << std::endl << std::endl;
                        uint32_t nCount1 = 0;
                        pDisplayNames->Count(&nCount1);
                        for (uint32_t nIndex1 = 0; nIndex1 < nCount1; nIndex1++)
                        {
                            int32_t nLanguageID;
                            pDisplayNames->KeyAt(nIndex1, &nLanguageID);
                            CCfVariant strDisplayname;
                            pDisplayNames->ValueAt(nLanguageID, &strDisplayname);
                            std::cout << "LangauageID =" << nLanguageID << " DisplayName
                            =" << CCfSmartString(strDisplayname).ToUTF8().c_str() << std::endl;
                        }
                    }
                    ICfMapIDToVariantPtr pDescriptions;
                    hr = pShcCategory->GetDescriptions(&pDescriptions);
```

Copy code

```

        if (pDescriptions != nullptr && CF_SUCCEEDED(hr))
        {
            std::cout << "Descriptions:" << std::endl << std::endl;
            uint32_t nCount2 = 0;
            pDescriptions->Count(&nCount2);
            for (uint32_t nIndex2 = 0; nIndex2 < nCount2; nIndex2++)
            {
                int32_t nLanguageID;
                pDescriptions->KeyAt(nIndex2, &nLanguageID);
                CCfVariant strDescription;
                pDescriptions->ValueAt(nLanguageID, &strDescription);
                std::cout << "LangauageID =" << nLanguageID << "
Description=" << CCfSmartString(strDescription).ToUTF8().c_str() << std::endl;
            }
        }
    }
}

```

9.7.6 ISHCCategoryProvider

Description

The C++ interface "ISHCCategoryProvider" provides you with read access to an "ISHCCategoryEnumerator" instance which contains an enumeration with the time categories of an "ISHCCalendar" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"Browse" method

Supplies an "ISHCCategoryEnumerator" instance which has access to an enumeration with the "ISHCCategory" instances of the calendar.

```
CRFESULT Browse(ISHCCategoryEnumerator** data)
```

- data
[out]: The enumerator

Example:

```

ISHCCategoryEnumeratorPtr pShcCategoryEnum;
CFRESULT hr = pShcCategoryProvider->Browse(&pShcCategoryEnum);
if (CF_SUCCEEDED(hr))
    PrintCategory(pShcCategoryEnum);

```

9.7.7 ISHCTimeSlice

Description

The C++ interface "ISHCTimeSlice" specifies the methods of a time slice.
The interface inherits from the "ICfUnknown" interface.

Members

"GetStartTime" method

Returns the start time of the "ISHCTimeSlice" instance.

```
CFRESULT GetStartTime(CFDATE64* pStartTime)
```

- `pStartTime`
[out]: Time stamp with the start time of the time slice

"GetDuration" method

Supplies the duration of the "ISHCTimeSlice" instance.

```
CFRESULT GetDuration(CFTIMESPAN64* pDuration)
```

- `pDuration`
[out]: The duration of the time slice

"GetCategory" method

Returns the time category time of the "ISHCTimeSlice" instance.

```
CFRESULT GetCategory(CFSTR* pstrCategoryName)
```

- `pstrCategoryName`
[out]: The name of the time category

"SetCategory" method

Sets the time category of the "ISHCTimeSlice" instance.

```
CFRESULT SetCategory(CFSTR pstrCategoryName)
```

- `pstrCategoryName`
[in]: The name of the new time category.

"SetStartTime" method

Sets the start time of the "ISHCTimeSlice" instance.

```
CFRESULT SetStartTime(CFDATE64 startTime)
```

- `startTime`
[in]: The new start time of the time slice.

"SetDuration" method

Sets the duration of the "ISHCTimeSlice" instance.

```
CRFESULT SetDuration(CFTIMESPAN64 duration)
```

- duration
[in]: The new duration of the time slice

9.7.8 ISHCTimeSliceEnumerator

Description

The C++ interface "ISHCTimeSliceEnumerator" specifies methods for handling the enumeration of the time slices of an "ISHCShiftTemplate" instance or "ISHCShift" instance.

The enumeration is returned by the `Read` method of these instances.

The interface inherits from the "ICfUnknown" interface.

Members

"MoveNext" method

Go to the next element of the enumeration.

```
CFRESULT MoveNext()
```

"Current" method

Output the current element of the enumeration.

```
CFRESULT Current(ISHCTimeSlice** ppItem)
```

- ppItem
[out]: The current "ISHCTimeSlice" instance

"Reset" method

Reset the current position in the enumeration. The "MoveNext" method moves afterwards to the first element.

```
CFRESULT Reset()
```

"Count" method

Output the size of the enumeration or the number of its elements.

```
CRFESULT Count(uint32_t* pCount)
```

- pCount
[out]: Number of time slices

Example

Copy code

```
void printTimeSlice(const ISHCTimeSliceEnumeratorPtr& p_pShcTimeSliceEnum)
{
    std::cout << endl << "*****TimeSlice*****" << std::endl << endl;
    if (p_pShcTimeSliceEnum != nullptr)
    {
        CFRESULT hr = CF_ERROR;
        uint32_t nCout = 0;
        p_pShcTimeSliceEnum->Count(&nCout);
        for (uint32_t nIndex = 0; nIndex < nCout; nIndex++)
        {
            if (CF_SUCCEEDED(p_pShcTimeSliceEnum->MoveNext()))
            {
                cout << endl;
                ISHCTimeSlicePtr pTimeSlice;
                p_pShcTimeSliceEnum->Current(&pTimeSlice);
                if (pTimeSlice != nullptr)
                {
                    CCfString strCategoryName;
                    hr = pTimeSlice->GetCategory(&strCategoryName);
                    if (CF_SUCCEEDED(hr) && (!strCategoryName.IsEmpty()))
                    {
                        std::cout << "Category= " << strCategoryName.ToUTF8().c_str() <<
std::endl;
                    }
                    CCfTimeSpan64 tsDuration;
                    hr = pTimeSlice->GetDuration(&tsDuration);
                    if (CF_SUCCEEDED(hr))
                    {
                        CCfString strDuration = tsDuration.GetTimeSpanString();
                        std::cout << "Durations= " << strDuration.ToUTF8().c_str() <<
std::endl;
                    }
                    CCfDateTime64 dtStartTime;
                    hr = pTimeSlice->GetStartTime(&dtStartTime);
                    if (CF_SUCCEEDED(hr))
                    {
                        CCfString strStarttime = dtStartTime.GetDateTimeString(false);
                        std::cout << "Start Time= " << strStarttime.ToUTF8().c_str() <<
std::endl;
                    }
                }
            }
        }
    }
}
```

9.7.9 ISHCDay

Description

The C++ interface "ISHCDay" specifies the methods of a day.

The interface inherits from the "ICfUnknown" interface.

Members

"CreateShift" method

Returns a new "ISHCShift" instance.

```
CRFESULT CreateShift(ISHCShiftTemplate* pShiftTemplate, CFTIMESPAN64
startTime, ISHCShift** pShift)
```

- `pShiftTemplate`
[in]: The "ISHCShiftTemplate" instance on which the new shift is to be based.
- `startTime`
[in]: Time stamp for the start time of the new shift.
- `pShift`
[out]: The new shift

"DeleteShift" method

Deletes an "ISHCShift" instance.

```
CRFESULT DeleteShift(ISHCShift* pShift)
```

- `pShift`
[in]: Reference to the shift to be deleted.

"GetShifts" method

Supplies an "ISHCShiftEnumerator" instance via which you access the "ISHCShift" instances of the "ISHCDay" instance.

```
CRFESULT GetShifts(ISHCShiftEnumerator** ppSHCShiftEnumerator)
```

- `ppSHCShiftEnumerator`
[out]: The enumerator with which you access the shifts of the "ISHCDay" instance. The shifts are contained in an array.

"GetComments" method

Supplies a map with the comments of the "ISHCDay" instance and their language code IDs.

```
CRFESULT GetComments(ICfMapIDToVariant** ppComments)
```

- `ppComments`
[out]: A map with int32/string pairs (language code ID for comment).

Example:

```
ICfMapIDToVariantPtr pComments;
hr = pShcshift->GetComments(&pComments);
if (pComments != nullptr && CF_SUCCEEDED(hr))
```

```
{
    std::cout << "comments:-" << std::endl << std::endl;
    uint32_t nCount = 0;  pComments->Count(&nCount);
    for (uint32_t nIndex1 = 0; nIndex1 < nCount; nIndex1++)
    {
        int32_t nLanguageID;
        pComments->KeyAt(nIndex1, &nLanguageID);
        CCfVariant strComments;
        pComments->ValueAt(nLanguageID, &strComments);
        std::cout << "LangauageID =" << nLanguageID << " Comments="
<< CCfSmartString(strComments).ToUTF8().c_str() << std::endl;
    }
}
```

"SetComments" method

Adds a new comment in the specified language to the "ISHCDay" instance.

CRFESULT SetComment(CFLCID languageId , CFSTR pComments)

- languageId
[in]: The language code ID
- pComments
[in]: The comment text

"GetStartTime" method

Returns the start time of the "ISHCDay" instance.

CRFESULT GetStartTime(CFDATEETIME64* pStartTime)

- pStartTime
[out]: The start time

"SetStartTime" method

Sets the start time of the "ISHCDay" instance.

CRFESULT SetStartTime(CFDATEETIME64 startTime)

- startTime
[in]: The new start time

"GetIsCustomized" method

Supplies the information on whether the "ISHCDay" instance was edited by users.

CRFESULT GetIsCustomized)(CFBOOL* pIsCustomized)

- pIsCustomized
[out]:
 - 0: Was not processed
 - 1: Was processed

"GetDayTemplate" method

Supplies the "ISHCDayTemplate" instance on which the "ISHCDay" instance is based.

```
CRFESULT GetDayTemplate(CFSTR* pDayTemplate)
```

- pDayTemplate
[out]: The day template

"SetDayTemplate" method

Sets the "ISHCDayTemplate" instance of the "ISHCDay" instance.

```
CRFESULT SetDayTemplate(CFSTR pDayTemplate)
```

- pDayTemplate
[in]: The new day template

See also

Locale IDs of the supported languages (Page 21)

9.7.10 ISHCDayEnumerator

Description

The C++ interface "ISHCDayEnumerator" specifies methods for handling the enumeration of the days of an "ISHCCalendar" instance. The enumeration is returned by the "Read" method of an "ISHCDayProvider" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"MoveNext" method

Go to the next element of the enumeration.

```
CFRESULT MoveNext()
```

"Current" method

Output the current element of the enumeration.

```
CFRESULT Current(ISHCTimeSlice** ppItem)
```

- ppItem
[out]: The current "ISHCDay" instance

"Reset" method

Reset the current position in the enumeration. The "MoveNext" method moves afterwards to the first element.

"Count" method

Output the size of the enumeration or the number of its elements.

CRFESULT Count(uint32_t* pCount)

- pCount
[out]: Number of days

Example

Copy code

```

void PrintDay(const ISHCDayEnumeratorPtr& p_pShcdEnum)
{
    cout << endl << "*****PrintDay*****" << endl << endl;
    if (p_pShcdEnum != nullptr)
    {
        CFRESULT hr = CF_ERROR;
        uint32_t nCount = 0;
        p_pShcdEnum->Count(&nCount);
        for (uint32_t nIndex = 0; nIndex < nCount; nIndex++)
        {
            cout << endl;
            if (CF_SUCCEEDED(p_pShcdEnum->MoveNext()))
            {
                ISHCDayPtr pShcd;
                p_pShcdEnum->Current(&pShcd);
                if (pShcd != nullptr)
                {
                    CCfString strDaytemplate;
                    hr = pShcd->GetDayTemplate(&strDaytemplate);
                    if (CF_SUCCEEDED(hr))
                    {
                        cout << "DayTemplate= " << strDaytemplate.ToUTF8().c_str() << endl;
                    }
                    ICfMapIDToVariantPtr pComments;
                    hr = pShcd->GetComments(&pComments);
                    if (CF_SUCCEEDED(hr) && pComments != nullptr)
                    {
                        std::cout << "comments:-" << std::endl << std::endl;
                        uint32_t nCount1 = 0;
                        pComments->Count(&nCount1);
                        for (uint32_t nIndex1 = 0; nIndex1 < nCount1; nIndex1++)
                        {
                            int32_t nLanguageID;
                            pComments->KeyAt(nIndex1, &nLanguageID);
                            CCfVariant strComments;
                            pComments->ValueAt(nLanguageID, &strComments);
                            std::cout << "LangauageID = " << nLanguageID << " Comments= " <<
CCfSmartString(strComments).ToUTF8().c_str() << std::endl;
                        }
                    }
                    CFBOOL bIsCustomized;
                    hr = pShcd->GetIsCustomized(&bIsCostomized);
                    if (CF_SUCCEEDED(hr))
                    {
                        cout << "IsCustomized=" << (uint32_t)bIsCostomized << endl;
                    }
                    CCfDateTime64 dtStartTime;
                    hr = pShcd->GetStartTime(&dtStartTime);
                    if (CF_SUCCEEDED(hr))
                    {
                        cout << "StartTime= " <<
dtStartTime.GetDateTimeString(false).ToUTF8().c_str() << endl;
                    }
                    ISHCShiftEnumeratorPtr pDayShifts;

```

Copy code

```
        hr = pShcday->GetShifts(&pDayShifts);
        if (CF_SUCCEEDED(hr))    printShift(pDayShifts);
    }
}
}
```

9.7.11 ISHCDayProvider

Description

The C++ interface "ISHCDayProvider" provides you with access to an "ISHCDayEnumerator" instance which contains an enumeration with the days of an "ISHCCalendar" instance. With the methods of the provider, you can create, read, update and delete days. The provider is returned by the "GetDayProvider" method of an "ISHCCalendar" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"Browse" method

Supplies an "ISHCDayEnumerator" instance which has access to an enumeration with the "ISHCDay" instances of a specific time period of the calendar.

```
CRFESULT Browse(CFDATE64 StartTime, CFDATE64 endTime,
ISHCDayEnumerator** ppISHCDayEnumerator)
```

- `startTime`
[in]: Start time
- `endTime`
[in]: End time
- `ppISHCDayEnumerator`
[out]: The enumerator

Example:

Copy code

```
CCfTimeSpan64 Get1Hour()
{
    CCfDateTime64 dt = CCfDateTime64::Now(false);
    CFDATE TIMEEST st2;
    dt.GetDateTimeStruct(st2);
    st2.cHours = st2.cHours - 1;
    CCfDateTime64 dt2;
    dt2.SetFromDateTimeStruct(&st2);
    return dt.GetDifference(dt2);
}

CCfDateTime64 GetStartoftheDay()
{
    CCfDateTime64 dt = CCfDateTime64::Now(false);
    CFDATE TIMEEST st;
    dt.GetDateTimeStruct(st);
    st.cHours = 0;
    st.cMinutes = 0;
    st.cSeconds = 0;
    st.sHundredNanoSeconds = 0;
    st.sMicroSeconds = 0;
    st.sMilliSeconds = 0;
    dt.SetFromDateTimeStruct(&st);
    return dt;
}

ISHCDayEnumeratorPtr pDayEnum; // Get day instances for a timespan of three days
CFRESULT hr = pShcDayProvider->Browse(GetStartoftheDay() - (Get1Hour() * 24),
GetStartoftheDay() + (Get1Hour() * 48), &pDayEnum);
```

"Create" method

Adds new days to the enumeration with the "ISHCDay" instances of the calendar.

```
CFRESULT Create(ICfArrayIUnknown* pDays) i
```

- pDays
[in]: An array with the days to be added.

Example:

Copy code

```

void CreateDayWithShift()
{
    if (nullptr != pShcDayTemplateProvider && nullptr != pCalendar && nullptr !=
pShcDayProvider && pShcShiftTemplateProvider != nullptr)
    {
        CFRESULT hr = CF_ERROR;
        ISHCDayTemplateEnumeratorPtr pShcDayTemplates;
        CCfArrayIUnknown arrDays;
        ICfArrayIUnknownPtr pDays;
        hr = pShcDayTemplateProvider->Browse(CF_FALSE, &pShcDayTemplates);
        if (CF_SUCCEEDED(hr) && pShcDayTemplates != nullptr)
        {
            uint32_t nCount = 0;
            pShcDayTemplates->Count(&nCount);
            if (nCount > 0)
            {
                if (CF_SUCCEEDED(pShcDayTemplates->MoveNext()))
                {
                    ISHCDayTemplatePtr pShcDayTemplate;
                    hr = pShcDayTemplates->Current(&pShcDayTemplate);
                    CCfString strDayTemplateName;
                    pShcDayTemplate->GetName(&strDayTemplateName);
                    ICfUnknownPtr pUnk;
                    pCalendar->GetObject(ODK_SHC_DAY, &pUnk);
                    ISHCDayPtr pShcDay = (ISHCDayPtr)pUnk;
                    if (pShcDay != nullptr)
                    {
                        hr = pShcDay->SetDayTemplate(strDayTemplateName);
                        if (CF_FAILED(hr))
                        {
                            std::cout << "failed to SetDayTemplate" << std::endl;
                        }
                        CCfDateTime64 dt = GetStartoftheDay();
                        hr = pShcDay->SetStartTime(dt);
                        if (CF_FAILED(hr))
                        {
                            std::cout << "failed to SetStartTime" << std::endl;
                        }
                        hr = pShcDay->SetComment(1033, CCfString(L"DayComments"));
                        if (CF_FAILED(hr))
                        {
                            std::cout << "failed to SetComment" << std::endl;
                        }
                    }
                }
                arrDays.Append(pShcDay);
                arrDays.DetachEnumerator(&pDays);
                hr = pShcDayProvider->Create(pDays);
                ISHCShiftTemplateEnumeratorPtr pShcShiftTemplateEnum;
                hr = pShcShiftTemplateProvider->Browse(CF_FALSE, &pShcShiftTemplateEnum);
                if (CF_SUCCEEDED(hr) && pShcShiftTemplateEnum != nullptr)
                {
                    if (CF_SUCCEEDED(pShcShiftTemplateEnum->MoveNext()))
                    {
                        ISHCShiftTemplatePtr pshcShiftTemplate;
                        pShcShiftTemplateEnum->Current(&pshcShiftTemaplate);

```

Copy code

```
ISHCShiftPtr pShcdayShift;
CCfTimeSpan64 starttime = Get1Hour() * 10;
hr = pShcDay->CreateShift(pshcShiftTemaplate, starttime,
&pShcdayShift);

if (CF_FAILED(hr) || pShcdayShift == nullptr)
{
    std::cout << "failed to Create Shift" << std::endl;
}
}
}
}
}
}
}
}
```

"Update" method

Updates "ISHCDay" instances of the enumeration.

CRFESULTUpdate(ICfArrayIUnknown* pDays)

- pDays
[in]: An array with the days to be updated.

Example:

Copy code

```
void UpdateDayWithShift()
{
    if (pShcDayProvider != nullptr)
    {
        CFRESULT hr = CF_ERROR;
        ISHCDayEnumeratorPtr pDayEnum;
        hr = pShcDayProvider->Browse(GetStartoftheDay() - (Get1Hour() * 24),
        GetStartoftheDay() + (Get1Hour() * 24), &pDayEnum);
        if (pDayEnum != nullptr && CF_SUCCEEDED(hr))
        {
            hr = pDayEnum->MoveNext();
            if (CF_SUCCEEDED(hr))
            {
                CCfArrayIUnknown ArrayDays;
                ICfArrayIUnknownPtr pArrayDays;
                ISHCDayPtr pShcDay;
                pDayEnum->Current(&pShcDay);
                if (pShcDay != nullptr)
                {
                    CCfDateTime64 dt;
                    pShcDay->GetStartTime(&dt);
                    dt.AddTimeSpan(Get1Hour());
                    hr = pShcDay->SetStartTime(dt); //Update StartTime Not Supported
                    hr = pShcDay->SetComment(1033, CCfString(L"DayComment"));
                    if (CF_FAILED(hr))
                    {
                        std::cout << "Failed to SetComment" << std::endl;
                    }
                    ISHCShiftEnumeratorPtr pShiftEnum;
                    hr = pShcDay->GetShifts(&pShiftEnum);
                    if (CF_SUCCEEDED(hr) && pShiftEnum != nullptr)
                    {
                        hr = pShiftEnum->MoveNext();
                        if (CF_SUCCEEDED(hr))
                        {
                            ISHCShiftPtr pShcShift;
                            pShiftEnum->Current(&pShcShift);
                            if (pShcShift != nullptr)
                            {
                                CCfTimeSpan64 ts;
                                hr = pShcShift->GetDuration(&ts);
                                if (CF_FAILED(hr))
                                {
                                    std::cout << "failed to GetDuration" << std::endl;
                                }
                                hr = pShcShift->SetDuration(ts + Get1Hour());
                                if (CF_FAILED(hr))
                                {
                                    std::cout << "failed to SetDuration" << std::endl;
                                }
                                hr = pShcShift->SetComment(1033,
                                CCfString(L"UpdatedShiftComments"));
                                if (CF_FAILED(hr))
                                {
                                    std::cout << "failed to SetComment" << std::endl;
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

Copy code

```
        }
    }
}

ArrayDays.Append(pShcDay);
ArrayDays.DetachEnumerator(&pArrayDays);
hr = pShcDayProvider->Update(pArrayDays);
if (CF_FAILED(hr))
{
    std::cout << "failed to Update" << std::endl;
}
}
}
}
}
```

"Delete" method

Deletes "ISHCDay" instances of the calendar from the enumeration.

CRFESULT Delete(ICfArrayIUnknown* pActionTemplates)

- pActionTemplates
[in]: An array with the days to be deleted.

Example:

Copy code

```

void DeleteDayWithShift()
{
    if (pShcDayProvider != nullptr)
    {
        CFRESULT hr = CF_ERROR;
        ISHCDayEnumeratorPtr pDayEnum;
        hr = pShcDayProvider->Browse(GetStartoftheDay() - (Get1Hour() * 24),
GetStartoftheDay() + (Get1Hour() * 48), &pDayEnum);
        if (pDayEnum != nullptr && CF_SUCCEEDED(hr))
        {
            uint32_t nSize = 0;
            pDayEnum->Count(&nSize);
            CCfArrayIUnknown ArrayDays;
            ICfArrayIUnknownPtr pArrayDays;
            for (uint32_t nIdx = 0; nIdx < nSize; nIdx++)
            {
                hr = pDayEnum->MoveNext();
                if (CF_SUCCEEDED(hr))
                {
                    ISHCDayPtr pShcDay;
                    pDayEnum->Current(&pShcDay);
                    if (pShcDay != nullptr)
                    {
                        ISHCShiftEnumeratorPtr pShiftEnum;
                        hr = pShcDay->GetShifts(&pShiftEnum);
                        if (CF_SUCCEEDED(hr) && pShiftEnum != nullptr)
                        {
                            hr = pShiftEnum->MoveNext();
                            if (CF_SUCCEEDED(hr))
                            {
                                ISHCShiftPtr pShcShift;
                                pShiftEnum->Current(&pShcShift);
                                if (pShcShift != nullptr)
                                {
                                    hr = pShcDay->DeleteShift(pShcShift);
                                    if (CF_FAILED(hr))
                                    {
                                        std::cout << "failed to DeleteShift" << std::endl;
                                    }
                                }
                            }
                        }
                    }
                    ArrayDays.Append(pShcDay);
                }
            }
            ArrayDays.DetachEnumerator(&pArrayDays);
            hr = pShcDayProvider->Delete(pArrayDays);
            if (CF_FAILED(hr))
            {
                std::cout << "failed to Delete" << std::endl;
            }
        }
    }
}

```


9.7.12 ISHCDayTemplate

Description

The C++ interface "ISHCDayTemplate" specifies the methods of a day template.
The interface inherits from the "ICfUnknown" interface.

Members

"GetName" method

Supplies the name of the "ISHCDayTemplate" instance.

```
CFRESULT GetName(CFSTR * pvarRet)
```

- pvarRet
[out]: The name of the day template

"SetName" method

Sets the name of the "ISHCDayTemplate" instance.

```
CRFRESULT SetName(CFSTR value)
```

"GetDisplayNames" method

Supplies a map with the display names of the "ISHCDayTemplate" instance and their language code IDs.

```
CFRESULT GetDisplayNames(ICfMapIDToVariant** ppDisplayNames)
```

- ppDisplayNames
[out]: A map with int32/string pairs (language code ID for display name).

Example:

```
ICfMapIDToVariantPtr pDisplayNames;  
CFRESULT hr = pShcDayTemplate->GetDisplayNames(&pDisplayNames);  
if (pDisplayNames != nullptr && CF_SUCCEEDED(hr))  
{  
    std::cout << "DisplayNames::" << std::endl << std::endl;  
    uint32_t nCount2 = 0;  
    pDisplayNames->Count(&nCount2);  
    for (uint32_t nIndex2 = 0; nIndex2 < nCount2; nIndex2++)  
    {  
        int32_t nLanguageID;  
        pDisplayNames->KeyAt(nIndex2, &nLanguageID);  
        CCfVariant strDisplayname;  
        pDisplayNames->ValueAt(nLanguageID, &strDisplayname);  
        std::cout << "LangauageID =" << nLanguageID << "  
Displayname=" << CCfSmartString(strDisplayname).ToUTF8().c_str() <<  
std::endl;  
    }  
}
```

"SetDisplayName" method

Adds a new display name in the specified language to the "ISHCDayTemplate" instance.

```
CRFESULT SetDisplayName(CFLCID languageId, CFSTR pDisplayName)
```

- languageId
[in]: The language code ID
- pDisplayName
[in]: The comment text

"GetDescriptions" method

Supplies a map with the descriptions of the "ISHCDayTemplate" instance and their language code IDs.

```
CFRESULT GetDescriptions(ICfMapIDToVariant** ppDisplayNames)
```

- ppDisplayNames
[out]: A map with int32/string pairs (language code ID for description).

Example: Similar to "GetDescriptions" of "ISHCCategory".

"SetDescription" method

Adds a new description in the specified language to the "ISHCDayTemplate" instance.

```
CRFESULT SetDescription(CFLCID languageId, CFSTR pDescriptions)
```

- languageId
[in]: The language code ID
- pDescriptions
[in]: The description text

"GetShifts" method

Supplies an "ISHCShiftEnumerator" instance. The enumerator provides you with access to the "ISHCShift" instances of the "ISHCDayTemplate" instance.

```
GetShifts(ISHCShiftEnumerator** ppSHCShiftEnumerator)
```

- ppSHCShiftEnumerator
[out]: The enumerator

"CreateShift" method

Adds an "ISHCShift" instance to the "ISHCDayTemplate" instance.

```
CRFESULT CreateShift(ISHCShiftTemplate* pShiftTemplate, CFTIMESPAN64 pStartTime, ISHCShift** pShift)
```

- pShiftTemplate
[in]: Reference to the shift template on which the shift is based.
- pStartTime
[in] Time stamp with the start time of the shift
- pShift
[out] reference to the new shift

"DeleteShift" method

Deletes an "ISHCShift" instance of the "ISHCDayTemplate" instance.

```
CFRESULT DeleteShift (ISHCShift* pShift)
```

- pShift
[in]: Reference to the shift to be deleted

"GetIsDeleted" method

Supplies the information on whether the "ISHCDayTemplate" instance was deleted by users.

```
CFRESULT GetIsDeleted (CFBOOL* pIsDeleted)
```

- pIsDeleted
[out]:
 - 0: Was not deleted
 - 1: Was deleted

9.7.13 ISHCDayTemplatesProvider

Description

The C++ interface "ISHCDayTemplatesProvider" provides you with access to an "ISHCDayTemplateEnumerator" instance which contains an enumeration with the day templates of an "ISHCCalendar" instance. With the methods of the provider, you can create, read, update and delete day templates. The provider is returned by the "GetDayTemplateProvider" method of an "ISHCCalendar" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"Browse" method

Supplies an "ISHCDayTemplateEnumerator" instance which has access to an enumeration with the "ISHCDayTemplate" instances of the calendar.

```
CFRESULT Browse (CFBOOL includeDeleted, ISHCDayTemplateEnumerator**  
ppISHCDayTemplateEnumerator)
```

- includeDeleted
Saves whether the enumerator also has access to the deleted day templates.
- ppISHCDayTemplateEnumerator
[out]: The enumerator

Example:

```
ISHCDayTemplateEnumeratorPtr pShcDayTemplates;  
CFRESULT hr = pShcDayTemplateProvider->Browse (CF_FALSE,  
&pShcDayTemplates);
```

"Create" method

Adds new "ISHCDayTemplate" instances to the enumeration with the "ISHCDayTemplate" instances of the calendar.

CRFESULTUpdate(ICfArrayIUnknown* pDayTemplates)

- pDayTemplates
[in]: An array with the day templates to be added.

Example:

Copy code

```

void CreateDayTemplateWithShift()
{
    if (pCalendar != nullptr && pShcDayTemplateProvider != nullptr &&
        pShcShiftTemplateProvider != nullptr)
    {
        CFRESULT hr = CF_ERROR;
        ICfUnknownPtr pUnk;
        pCalendar->GetObject(ODK_SHC_DAY_TEMPLATE, &pUnk);
        CCfArrayIUnknown ArrayTemplate;
        ICfArrayIUnknownPtr pArrayTemplate;
        ISHCDayTemplatePtr pShcDayTemplate = (ISHCDayTemplatePtr)pUnk;
        if (pShcDayTemplate != nullptr)
        {
            hr = pShcDayTemplate->SetName(CCfString(L"DayTemplateName"));
            if (CF_FAILED(hr))
            {
                std::cout << "failed to SetName" << std::endl;
            }
            hr = pShcDayTemplate->SetDisplayName(1033,
                CCfString(L"DayTemplateDisplayName"));
            if (CF_FAILED(hr))
            {
                std::cout << "failed to SetDisplayName" << std::endl;
            }
            hr = pShcDayTemplate->SetDescription(1033, CCfString(L"DayTemplate
                Descriptions"));
            if (CF_FAILED(hr))
            {
                std::cout << "failed to SetDescription" << std::endl;
            }
        }
        ArrayTemplate.Append(pShcDayTemplate);
        ArrayTemplate.DetachEnumerator(&pArrayTemplate);
        hr = pShcDayTemplateProvider->Create(pArrayTemplate);
        if (CF_SUCCEEDED(hr))
        {
            ISHCShiftTemplateEnumeratorPtr pShcShiftTemplates;
            hr = pShcShiftTemplateProvider->Browse(CF_FALSE, &pShcShiftTemplates);
            if (CF_SUCCEEDED(hr) && pShcShiftTemplates != nullptr)
            {
                if (CF_SUCCEEDED(pShcShiftTemplates->MoveNext()))
                {
                    ISHCShiftTemplatePtr pShcshiftTemplate;
                    pShcShiftTemplates->Current(&pShcshiftTemplate);
                    if (pShcshiftTemplate != nullptr)
                    {
                        CCfString strShiftTemplateName;
                        pShcshiftTemplate->GetName(&strShiftTemplateName);
                        ISHCShiftPtr pShift;
                        hr = pShcDayTemplate->CreateShift(pShcshiftTemplate, Get1Hour() * 1,
                            &pShift);

                        if (CF_FAILED(hr) || pShift == nullptr)
                        {
                            cout << "Failed to Create Shift" << endl;
                        }
                    }
                }
            }
        }
    }
}

```

Copy code

```
    }  
    }  
    }  
    }  
}
```

"Update" method

Updates "ISHCDayTemplate" instances of the enumeration.

CRFESULTUpdate(ICfArrayIUnknown* pDayTemplates)

- pDayTemplates
[in]: An array with the day templates to be updated.

Example:

Copy code

```
void UpdateDayTemplateWithShift()
{
    if (pShcDayTemplateProvider != nullptr)
    {
        CFRESULT hr = CF_ERROR;
        ISHCDayTemplateEnumeratorPtr pShcDayTemplates;
        hr = pShcDayTemplateProvider->Browse(false, &pShcDayTemplates);
        CCfArrayIUnknown ArrayDayTemplate;
        ICfArrayIUnknownPtr pArrayDayTemplate;
        if (CF_SUCCEEDED(hr) && pShcDayTemplates != nullptr)
        {
            uint32_t nCount = 0; pShcDayTemplates->Count(&nCount);
            for (uint32_t i = 0; i < nCount; i++)
            {
                if (CF_SUCCEEDED(pShcDayTemplates->MoveNext()))
                {
                    ISHCDayTemplatePtr pShcDayTemplate;
                    hr = pShcDayTemplates->Current(&pShcDayTemplate);
                    if (CF_SUCCEEDED(hr))
                    {
                        hr = pShcDayTemplate->SetName(CCfString(L"UpdatedDayTemplateName"));
                        if (CF_FAILED(hr))
                        {
                            std::cout << "SetName failed" << std::endl;
                        }
                        ISHCShiftEnumeratorPtr pShiftEnum;
                        hr = pShcDayTemplate->GetShifts(&pShiftEnum);
                        if (CF_SUCCEEDED(hr) && pShiftEnum != nullptr)
                        {
                            uint32_t nlength = 0;
                            pShiftEnum->Count(&nlength);
                            for (uint32_t nIndex = 0; nIndex < nlength; nIndex++)
                            {
                                if (CF_SUCCEEDED(pShiftEnum->MoveNext()))
                                {
                                    ISHCShiftPtr pShcShift;
                                    hr = pShiftEnum->Current(&pShcShift);
                                    if (CF_SUCCEEDED(hr) && pShcShift != nullptr)
                                    {
                                        ISHCTimeSliceEnumeratorPtr pShcTimeSliceEnum;
                                        hr = pShcShift->GetTimeSlices(&pShcTimeSliceEnum);
                                        pShcShift->SetDuration(Get1Hour() * 8);
                                        pShcShift->SetComment(1033, CCfString("ShiftComment"));
                                        if (CF_SUCCEEDED(hr) && pShcTimeSliceEnum != nullptr)
                                        {
                                            uint32_t nCount1 = 0;
                                            hr = pShcTimeSliceEnum->Count(&nCount1);
                                            for (uint32_t nIndex1 = 0; nIndex1 < nCount1; nIndex1+
+
                                                {
                                                    if (CF_SUCCEEDED(pShcTimeSliceEnum->MoveNext()))
                                                    {
                                                        ISHCTimeSlicePtr pShctimeslice;
                                                        hr = pShcTimeSliceEnum->
>Current(&pShctimeslice);
```

[illegible]

Deletes "ISHCDayTemplate" instances of the calendar from the enumeration.

```
CRFRESULT Delete(ICfArrayIUnknown* pDayTemplates)
```

- `pDayTemplates`
[in]: An array with the day templates to be deleted.

Example:

Copy code

```

void DeleteDaytemplateWithShift()
{
    if (pShcDayTemplateProvider != nullptr)
    {
        CFRESULT hr = CF_ERROR;
        ISHCDayTemplateEnumeratorPtr pShcDayTemplates;
        hr = pShcDayTemplateProvider->Browse(CF_FALSE, &pShcDayTemplates);
        CCfArrayIUnknown ArrayDayTemplate;
        ICfArrayIUnknownPtr pArrayDayTemplate;
        if (CF_SUCCEEDED(hr) && pShcDayTemplates != nullptr)
        {
            uint32_t nCount = 0;
            pShcDayTemplates->Count(&nCount);
            for (uint32_t i = 0; i < nCount; i++)
            {
                if (CF_SUCCEEDED(pShcDayTemplates->MoveNext()))
                {
                    ISHCDayTemplatePtr pShcDayTemplate;
                    hr = pShcDayTemplates->Current(&pShcDayTemplate);
                    if (CF_SUCCEEDED(hr) && pShcDayTemplate != nullptr)
                    {
                        ISHCShiftEnumeratorPtr pShiftEnum;
                        hr = pShcDayTemplate->GetShifts(&pShiftEnum);
                        if (CF_SUCCEEDED(hr) && pShiftEnum != nullptr)
                        {
                            uint32_t nlength = 0;
                            pShiftEnum->Count(&nlength);
                            for (uint32_t nIndex = 0; nIndex < nlength; nIndex++)
                            {
                                if (CF_SUCCEEDED(pShiftEnum->MoveNext()))
                                {
                                    ISHCShiftPtr pShcShift;
                                    hr = pShiftEnum->Current(&pShcShift);
                                    if (CF_SUCCEEDED(hr) && pShcShift != nullptr)
                                    {
                                        hr = pShcDayTemplate->DeleteShift(pShcShift);
                                        if (CF_FAILED(hr))
                                        {
                                            std::cout << "failed to Update" << std::endl;
                                        }
                                    }
                                }
                            }
                        }
                    }
                    ArrayDayTemplate.Append(pShcDayTemplate);
                }
            }
            ArrayDayTemplate.DetachEnumerator(&pArrayDayTemplate);
            hr = pShcDayTemplateProvider->Delete(pArrayDayTemplate);
            if (CF_FAILED(hr))
            {
                std::cout << "failed to Update" << std::endl;
            }
        }
    }
}

```

Copy code

```
}  
}  
}
```

9.7.14 ISHCShiftTemplate

Description

The C++ interface "ISHCShiftTemplate" specifies the methods of a shift template.
The interface inherits from the "ICfUnknown" interface.

Members

"GetName" method

Supplies the name of the "ISHCShiftTemplate" instance.

```
GetName (CFSTR * pvarRet)
```

- pvarRet
[out]: Name

"SetName" method

Sets the name of the "ISHCShiftTemplate" instance.

```
CRFESULT SetName (CFSTR value)
```

"GetDisplayNames" method

Supplies a map with the display names of the "ISHCShiftTemplate" instance and their language code IDs.

```
CFRESULT GetDisplayNames (ICfMapIDToVariant** ppDisplayNames)
```

- ppDisplayNames
[out]: A map with int32/string pairs (language code ID for display name).

"SetDisplayName" method

Adds a new entry to a map with the display names of the "ISHCShiftTemplate" instance and their language code IDs.

```
CRFESULT SetDisplayName (CFLCID languageId, CFSTR pDisplayName)
```

- languageId
[in]: The language code ID
- pDisplayName
[in]: The comment text

"GetDescriptions" method

Supplies a map with the descriptions of the "ISHCShiftTemplate" instance and their language code IDs.

```
CFRESULT GetDescriptions(ICfMapIDToVariant** ppDisplayNames)
```

- ppDisplayNames
[out]: A map with int32/string pairs (language code ID for description).

"SetDescription" method

Adds a new entry to a map with the description of the "ISHCShiftTemplate" instance and their language code IDs.

```
CRFRESULT SetDescription(CFLCID languageId, CFSTR pDescriptions)
```

- languageId
[in]: The language code ID
- pDescriptions
[in]: The description text

"GetIsDeleted" method

Supplies the information on whether the "ISHCShiftTemplate" instance was deleted by users.

```
CFRESULT GetIsDeleted(CFBOOL* pIsDeleted)
```

- pIsDeleted
[out]:
 - 0: Was not deleted
 - 1: Was deleted

"GetDuration" method

Supplies the duration of the "ISHCShiftTemplate" instance.

```
CFRESULT GetDuration(CFTIMESPAN64* pDuration)
```

- pDuration
[out]: The duration of the shift template

"SetDuration" method

Sets the duration of the "ISHCShiftTemplate" instance.

```
CFRESULT SetDuration(CFTIMESPAN64 duration)
```

- duration
[in]: The duration of the shift template

"GetTimeSlices" method

Supplies an "ISHCTimeSliceEnumerator" instance. The enumerator provides you with access to the "ISHCTimeSlice" instances of the "ISHCShiftTemplate" instance.

```
CRFRESULT GetTimeSlices(ISHCTimeSliceEnumerator**  
ppSHCTimeSliceEnumerator)
```

- ppSHCTimeSliceEnumerator
[out]: The enumerator

"CreateTimeSlice" method

Adds an "ISHCTimeSlice" instance to the "ISHCShiftTemplate" instance.

```
CRFESULT CreateTimeSlice(ISHCTimeSlice* pSlice)
```

- pSlice
[in]: Reference to the new time slice

"DeleteTimeSlice" method

Deletes an "ISHCTimeSlice" instance of the "ISHCShiftTemplate" instance.

```
CRFESULT DeleteTimeSlice(ISHCTimeSlice* pSlice)
```

- pSlice
[in]: Reference to the time slice to be deleted

9.7.15 ISHCShiftTemplateEnumerator

Description

The C++ interface "ISHCShiftTemplateEnumerator" specifies methods for handling the enumeration of the shift templates of an "ISHCCalendar" instance. The enumeration is returned by the "Read" method of an "ISHCShiftTemplatesProvider" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"MoveNext" method

Go to the next element of the enumeration.

```
CFRESULT MoveNext()
```

"Current" method

Output the current element of the enumeration.

```
CFRESULT Current(ISHCShiftTemplate** ppItem)
```

- ppItem
[out]: The current shift template

"Reset" method

Reset the current position in the enumeration. The "MoveNext" method moves afterwards to the first element.

```
CFRESULT Reset()
```

"Count" method

Output the size of the enumeration or the number of its elements.

Supplies the number of shift templates in the list.

9.7 Interfaces of the Calendar option

```
CRFESULT Count(uint32_t* pCount)
```

- pCount
[out]: Number of shift templates

Example

Copy code

```
void PrintShiftTemplate(const ISHCShiftTemplateEnumeratorPtr& p_pShcShiftTemplateEnum)
{
    std::cout << std::endl << "*****PrintShiftTemplate*****" << std::endl <<
    endl;
    if (p_pShcShiftTemplateEnum != nullptr)
    {
        CFRESULT hr = CF_ERROR;
        uint32_t nCout = 0;
        p_pShcShiftTemplateEnum->Count(&nCout);
        for (uint32_t nIndex = 0; nIndex < nCout; nIndex++)
        {
            cout << endl;
            if (CF_SUCCEEDED(p_pShcShiftTemplateEnum->MoveNext()))
            {
                ISHCShiftTemplatePtr pShcShiftTemplate;
                p_pShcShiftTemplateEnum->Current(&pShcShiftTemplate);
                if (pShcShiftTemplate != nullptr)
                {
                    CCfString strName;
                    hr = pShcShiftTemplate->GetName(&strName);
                    if (CF_SUCCEEDED(hr) && (!strName.IsEmpty()))
                    {
                        std::cout << "Name=" << strName.ToUTF8().c_str() << std::endl;
                    }
                    CCfTimeSpan64 tsDuration;
                    hr = pShcShiftTemplate->GetDuration(&tsduration);
                    if (CF_SUCCEEDED(hr))
                    {
                        CCfString strDuration = tsduration.GetTimeSpanString();
                        std::cout << "Duration=" << strDuration.ToUTF8().c_str() <<
std::endl;
                    }
                    CFBOOL bIsDeleted;
                    hr = pShcShiftTemplate->GetIsDeleted(&bIsDeleted);
                    if (CF_SUCCEEDED(hr))
                    {
                        std::cout << "ISDeleted=" << (uint32_t)bIsDeleted << std::endl;
                    }
                    ICfMapIDToVariantPtr pDescriptions;
                    hr = pShcShiftTemplate->GetDescriptions(&pDescriptions);
                    if (pDescriptions != nullptr && CF_SUCCEEDED(hr))
                    {
                        std::cout << "Descriptions=" << std::endl << std::endl; uint32_t
nCount = 0;
                        pDescriptions->Count(&nCount);
                        for (uint32_t nIndex1 = 0; nIndex1 < nCount; nIndex1++)
                        {
                            int32_t nLanguageID;
                            pDescriptions->KeyAt(nIndex1, &nLanguageID);
                            CCfVariant strDescription;
                            pDescriptions->ValueAt(nLanguageID, &strDescription);
                            std::cout << "LanguageID =" << nLanguageID << " Description="
<< CCfSmartString(strDescription).ToUTF8().c_str() << std::endl;
                        }
                    }
                }
            }
        }
    }
}
```

Copy code

```

    }
    ICfMapIDToVariantPtr pDisplayNames;
    hr = pShcShiftTemplate->GetDisplayNames(&pDisplayNames);
    if (pDisplayNames != nullptr && CF_SUCCEEDED(hr))
    {
        std::cout << "DisplayNames::" << std::endl << std::endl;
        uint32_t nCount = 0;
        pDisplayNames->Count(&nCount);
        for (uint32_t nIndex1 = 0; nIndex1 < nCount; nIndex1++)
        {
            int32_t nLanguageID;
            pDisplayNames->KeyAt(nIndex1, &nLanguageID);
            CCfVariant strDisplayname;
            pDisplayNames->ValueAt(nLanguageID, &strDisplayname);
            std::cout << "LangauageID =" << nLanguageID << " DisplayName="
<< CCfSmartString(strDisplayname).ToUTF8().c_str() << std::endl;
        }
    }
    ISHCTimeSliceEnumeratorPtr pShiftTemplatetimeSlice;
    hr = pShcShiftTemplate->GetTimeSlices(&pShiftTemplatetimeSlice);
    if (CF_SUCCEEDED(hr)) printTimeSlice(pShiftTemplatetimeSlice);
}
}
}
}
}

```

9.7.16 ISHCShiftTemplatesProvider**Description**

The C++ interface "ISHCShiftTemplatesProvider" provides you with access to an "ISHCShiftTemplateEnumerator" instance which contains an enumeration with the shift templates of an "ISHCCalendar" instance. With the methods of the provider, you can create, read, update and delete shift templates. The provider is returned by the "GetShiftTemplateProvider" method of an "ISHCCalendar" instance.

The interface inherits from the "ICfUnknown" interface.

Members**"Browse" method**

Supplies an "ISHCShiftTemplateEnumerator" instance which has access to an enumeration with the "ISHCShiftTemplate" instances of the calendar.

```
CFRESULT Browse(CFBOOL includeDeleted, ISHCShiftTemplateEnumerator**  
ppISHCShiftTemplateEnumerator)
```

- `includeDeleted`
Saves whether the enumerator also has access to the deleted shift templates.
- `ppISHCShiftTemplateEnumerator`
[out]: The enumerator

Example:

```
ISHCDayTemplateEnumeratorPtr pShcDayTemplates;  
CFRESULT hr = pShcDayTemplateProvider->Browse(CF_FALSE,  
&pShcDayTemplates);
```

"Create" method

Adds new shift templates to the enumeration with the "ISHCShiftTemplate" instances of the calendar.

```
CFRESULT Create(ICfArrayIUnknown* pShiftTemplates)
```

- `pShiftTemplates`
[in]: An array with the shift templates to be added.

Example:

Copy code

```
void CreateShiftTemplateWithTimeslice()
{
    if (pShcShiftTemplateProvider != nullptr)
    {
        CFRESULT hr = CF_ERROR;
        ISHCShiftTemplatePtr pShcShiftTemplate;
        ICfUnknownPtr pUnk;
        pCalendar->GetObject(ODK_SHC_SHIFT_TEMPLATE, &pUnk);
        pShcShiftTemplate = (ISHCShiftTemplatePtr)pUnk;
        if (pShcShiftTemplate != nullptr)
        {
            hr = pShcShiftTemplate->SetName(CcString(L"ShiftTemplateName"));
            if (CF_FAILED(hr))
            {
                cout << "Failed to SetName" << endl;
            }
            hr = pShcShiftTemplate->SetDisplayName(1033, CcString(L"ShiftDisplayName"));
            if (CF_FAILED(hr))
            {
                cout << "Failed to SetDisplayName" << endl;
            }
            hr = pShcShiftTemplate->SetDescription(1033,
CcString(L"ShiftTemplateDescription"));
            if (CF_FAILED(hr))
            {
                cout << "Failed to SetDescription" << endl;
            }
            hr = pShcShiftTemplate->SetDuration(Get1Hour() * 8);
            if (CF_FAILED(hr))
            {
                cout << "Failed to SetDuration" << endl;
            }
            ICfArrayIUnknownPtr pArrayShiftTemplate;
            CCfArrayIUnknown ArrayShiftTemplate;
            ArrayShiftTemplate.Append(pShcShiftTemplate);
            ArrayShiftTemplate.DetachEnumerator(&pArrayShiftTemplate);
            hr = pShcShiftTemplateProvider->Create(pArrayShiftTemplate);
            if (CF_FAILED(hr))
            {
                cout << "Failed to v" << endl;
            }
            ICfUnknownPtr pUnkTimeSlice;
            pCalendar->GetObject(ODK_SHC_TIME_SLICE, &pUnkTimeSlice);
            if (pUnkTimeSlice != nullptr)
            {
                ISHCTimeSlicePtr pShcTimeSlice;
                pUnkTimeSlice->QueryInterface(IID_ISHCTimeSlice,
(ICfUnknown**) &pShcTimeSlice);
                if (pShcTimeSlice != nullptr)
                {
                    pShcTimeSlice->SetCategory(CcString(L"Working"));
                    pShcTimeSlice->SetDuration(Get1Hour() * 1);
                    CCfDateTime64 dt = GetStartoftheDay();
                    pShcTimeSlice->SetStartTime(dt);
                    hr = pShcShiftTemplate->CreateTimeSlice(pShcTimeSlice);
                }
            }
        }
    }
}
```

Copy code

```
        if (CF_FAILED(hr))
        {
            std::cout << "Failed to Create TimeSlcie" << std::endl;
        }
    }
}
}
```

"Update" method

Updates the enumeration with the "ISHCShiftTemplate" instances of the calendar.

CRFESULT Update(ICfArrayIUnknown* pShiftTemplates)

- pShiftTemplates
[in]: An array with the shift templates to be updated.

Example:

Copy code

```

void UpdateShiftTemplateWithTimeSlice()
{
    if (pShcShiftTemplateProvider != nullptr)
    {
        CFRESULT hr = CF_ERROR;
        ISHCShiftTemplateEnumeratorPtr pShcShiftTemplates;
        hr = pShcShiftTemplateProvider->Browse(CF_FALSE, &pShcShiftTemplates);
        CCfArrayIUnknown ArrayShiftTemplate;
        ICfArrayIUnknownPtr pArrayShiftTemplate;
        if (CF_SUCCEEDED(hr) && pShcShiftTemplates != nullptr)
        {
            uint32_t nCount = 0;
            pShcShiftTemplates->Count(&nCount);
            for (uint32_t i = 0; i < nCount; i++)
            {
                if (CF_SUCCEEDED(pShcShiftTemplates->MoveNext()))
                {
                    ISHCShiftTemplatePtr pShcShiftTemplate;
                    hr = pShcShiftTemplates->Current(&pShcShiftTemplate);
                    if (CF_SUCCEEDED(hr))
                    {
                        hr = pShcShiftTemplate->SetDuration(Get1Hour() * 6);
                        if (CF_FAILED(hr))
                        {
                            std::cout << "SetDuration failed" << std::endl;
                        }
                        hr = pShcShiftTemplate-
>SetName(CCfString(L"UpdatedShiftTemplateName"));
                        if (CF_FAILED(hr))
                        {
                            std::cout << "SetName failed" << std::endl;
                        }
                        ISHCTimeSliceEnumeratorPtr pTimeSlilceEnum;
                        hr = pShcShiftTemplate->GetTimeSlices(&pTimeSlilceEnum);
                        if (CF_SUCCEEDED(hr) && pTimeSlilceEnum != nullptr)
                        {
                            uint32_t nlength = 0;
                            pTimeSlilceEnum->Count(&nlength);
                            for (uint32_t nIndex = 0; nIndex < nlength; nIndex++)
                            {
                                if (CF_SUCCEEDED(pTimeSlilceEnum->MoveNext()))
                                {
                                    ISHCTimeSlicePtr pShcTimeSlice;
                                    hr = pTimeSlilceEnum->Current(&pShcTimeSlice);
                                    if (CF_SUCCEEDED(hr) && pShcTimeSlice != nullptr)
                                    {
                                        pShcTimeSlice->SetDuration(Get1Hour() * 4);
                                        pShcTimeSlice->SetCategory(CCfString(L"Break"));
                                    }
                                }
                            }
                        }
                        ArrayShiftTemplate.Append(pShcShiftTemplate);
                    }
                }
            }
        }
    }
}

```


Copy code

```
    }  
    ArrayShiftTemplate.DetachEnumerator(&pArrayShiftTemplate);  
}  
hr = pShcShiftTemplateProvider->Update(pArrayShiftTemplate);  
if (CF_FAILED(hr))  
{  
    std::cout << "failed to Update" << std::endl;  
}  
}  
}
```

"Delete" method

Deletes an "ISHCShiftTemplate" instance of the calendar from the enumeration.

CRFESULT Delete(ICfArrayIUnknown* pShiftTemplates)

- pShiftTemplates
[in]: An array with the shift templates to be deleted.

Example:

Copy code

```
void DeleteShiftTemplateWithTimeSlice()
{
    if (pShcShiftTemplateProvider != nullptr)
    {
        CFRESULT hr = CF_ERROR;
        ISHCShiftTemplateEnumeratorPtr pShcShiftTemplates;
        hr = pShcShiftTemplateProvider->Browse(CF_FALSE, &pShcShiftTemplates);
        CCfArrayIUnknown ArrayShiftTemplate;
        ICfArrayIUnknownPtr pArrayShiftTemplate;
        if (CF_SUCCEEDED(hr) && pShcShiftTemplates != nullptr)
        {
            uint32_t nCount = 0;
            pShcShiftTemplates->Count(&nCount);
            for (uint32_t i = 0; i < nCount; i++)
            {
                if (CF_SUCCEEDED(pShcShiftTemplates->MoveNext()))
                {
                    ISHCShiftTemplatePtr pShcShiftTemplate;
                    hr = pShcShiftTemplates->Current(&pShcShiftTemplate);
                    if (CF_SUCCEEDED(hr) && pShcShiftTemplate != nullptr)
                    {
                        ISHCTimeSliceEnumeratorPtr pTimeSlilceEnum;
                        hr = pShcShiftTemplate->GetTimeSlices(&pTimeSlilceEnum);
                        if (CF_SUCCEEDED(hr) && pTimeSlilceEnum != nullptr)
                        {
                            uint32_t nlength = 0;
                            pTimeSlilceEnum->Count(&nlength);
                            for (uint32_t nIndex = 0; nIndex < nlength; nIndex++)
                            {
                                if (CF_SUCCEEDED(pTimeSlilceEnum->MoveNext()))
                                {
                                    ISHCTimeSlicePtr pShcTimeSlice;
                                    hr = pTimeSlilceEnum->Current(&pShcTimeSlice);
                                    if (pShcTimeSlice != nullptr)
                                    {
                                        hr = pShcShiftTemplate->DeleteTimeSlice(pShcTimeSlice);
                                        if (CF_FAILED(hr))
                                        {
                                            std::cout << "failed to DeleteTimeSlice" <<
std::endl;
                                        }
                                    }
                                }
                            }
                        }
                        ArrayShiftTemplate.Append(pShcShiftTemplate);
                    }
                }
            }
            ArrayShiftTemplate.DetachEnumerator(&pArrayShiftTemplate);
        }
        hr = pShcShiftTemplateProvider->Delete(pArrayShiftTemplate);
        if (CF_FAILED(hr))
        {
            std::cout << "failed to Delete" << std::endl;
        }
    }
}
```

Copy code

```
    }  
}  
}
```

9.7.17 ISHCShift

Description

The C++ interface "ISHCShift" specifies the methods of a shift.

The interface inherits from the "ICfUnknown" interface.

Members

"GetTimeSlices" method

Supplies an "ISHCTimeSliceEnumerator" instance. The enumerator provides you with access to the "ISHCTimeSlice" instances of the "ISHCShift" instance.

```
CRFESULT GetTimeSlices (ISHCTimeSliceEnumerator**  
ppSHCTimeSliceEnumerator)
```

- ppSHCTimeSliceEnumerator
[out]: The enumerator

"CreateTimeSlice" method

Adds an "ISHCTimeSlice" instance to the "ISHCShiftTemplate" instance.

```
CRFESULT CreateTimeSlice (ISHCTimeSlice* pSlice)
```

- pSlice
[in]: Reference to the new time slice

Example

Copy code

```

void CreateTimeSliceUsingShift()
{
    if (nullptr != pShcDayTemplateProvider && nullptr != pCalendar)
    {
        CFRESULT hr = CF_ERROR;
        ISHCDayTemplateEnumeratorPtr pShcDayTemplates;
        hr = pShcDayTemplateProvider->Read(CF_FALSE, &pShcDayTemplates);
        if (CF_SUCCEEDED(hr) && pShcDayTemplates != nullptr)
        {
            uint32_t nCount = 0;
            pShcDayTemplates->Count(&nCount);
            if (nCount > 0)
            {
                if (CF_SUCCEEDED(pShcDayTemplates->MoveNext()))
                {
                    ISHCDayTemplatePtr pShcDayTemplate;
                    hr = pShcDayTemplates->Current(&pShcDayTemplate);
                    if (CF_SUCCEEDED(hr))
                    {
                        ISHCShiftEnumeratorPtr pShiftEnum;
                        hr = pShcDayTemplate->GetShifts(&pShiftEnum);
                        if (CF_SUCCEEDED(hr) && pShiftEnum != nullptr)
                        {
                            uint32_t nlength = 0;
                            pShiftEnum->Count(&nlength);
                            if (nlength > 0)
                            {
                                if (CF_SUCCEEDED(pShiftEnum->MoveNext()))
                                {
                                    ISHCShiftPtr pShcShift;
                                    hr = pShiftEnum->Current(&pShcShift);
                                    if (CF_SUCCEEDED(hr) && pShcShift != nullptr)
                                    {
                                        ISHCTimeSlicePtr pShcTimeSlice;
                                        ICfUnknownPtr pUnk;
                                        hr = pCalendar->GetObject(ODK_SHC_TIME_SLICE, &pUnk);
                                        pShcTimeSlice = (ISHCTimeSlicePtr)pUnk;
                                        if (pShcTimeSlice != nullptr)
                                        {
                                            pShcTimeSlice->SetCategory(CCfString(L"Break"));
                                            pShcTimeSlice->SetDuration(Get1Hour() * 1);
                                            CCfDateTime64 dt = GetStartoftheDay();
                                            dt.AddTimeSpan(Get1Hour() * 3);
                                            pShcTimeSlice->SetStartTime(dt);
                                        }
                                        hr = pShcShift->CreateTimeSlice(pShcTimeSlice);
                                        if (CF_FAILED(hr))
                                        {
                                            std::cout << "failed to CreateTimeSlice" <<
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}

```

Copy code

```
    }  
    }  
    }  
    }  
}
```

"DeleteTimeSlice" method

Deletes an "ISHCTimeSlice" instance of the "ISHCShiftTemplate" instance.

CRFESULT DeleteTimeSlice(ISHCTimeSlice* pSlice)

- pSlice
[in]: Reference to the time slice to be deleted

Example:

Copy code

```
void DeleteTimeSliceUsingShift()
{
    if (nullptr != pShcDayTemplateProvider && nullptr != pCalendar)
    {
        CFRESULT hr = CF_ERROR;
        ISHCDayTemplateEnumeratorPtr pShcDayTemplates;
        hr = pShcDayTemplateProvider->Read(CF_FALSE, &pShcDayTemplates);
        if (CF_SUCCEEDED(hr) && pShcDayTemplates != nullptr)
        {
            uint32_t nCount = 0;
            pShcDayTemplates->Count(&nCount);
            if (nCount > 0)
            {
                if (CF_SUCCEEDED(pShcDayTemplates->MoveNext()))
                {
                    ISHCDayTemplatePtr pShcDayTemplate;
                    hr = pShcDayTemplates->Current(&pShcDayTemplate);
                    if (CF_SUCCEEDED(hr))
                    {
                        ISHCShiftEnumeratorPtr pShiftEnum;
                        hr = pShcDayTemplate->GetShifts(&pShiftEnum);
                        if (CF_SUCCEEDED(hr) && pShiftEnum != nullptr)
                        {
                            uint32_t nlength = 0;
                            pShiftEnum->Count(&nlength);
                            if (nlength > 0)
                            {
                                if (CF_SUCCEEDED(pShiftEnum->MoveNext()))
                                {
                                    ISHCShiftPtr pShcShift;
                                    hr = pShiftEnum->Current(&pShcShift);
                                    if (CF_SUCCEEDED(hr) && pShcShift != nullptr)
                                    {
                                        ISHCTimeSliceEnumeratorPtr pShcTimeSliceEnum;
                                        hr = pShcShift->GetTimeSlices(&pShcTimeSliceEnum);
                                        if (CF_SUCCEEDED(hr) && pShcTimeSliceEnum != nullptr)
                                        {
                                            uint32_t nCountTimeSlice = 0;
                                            pShcTimeSliceEnum->Count(&nCountTimeSlice);
                                            if (nCountTimeSlice > 0)
                                            {
                                                if (CF_SUCCEEDED(pShcTimeSliceEnum->MoveNext()))
                                                {
                                                    ISHCTimeSlicePtr pShcTimeSlice;
                                                    hr = pShcTimeSliceEnum-
>Current(&pShcTimeSlice);
                                                    = nullptr)

                                                    if (CF_SUCCEEDED(hr) && pShcTimeSlice !
                                                    {
                                                        hr = pShcShift-
                                                        if (CF_FAILED(hr))
                                                        {
                                                            std::cout << "failed to DeleteTimeSlice"
<< std::endl;
```

Supplies a map with the comments of the "ISHCShift" instance and their language code IDs.

- `ppComments`
[out]: A map with int32/string pairs (language code ID for comment).

Adds a new entry to a map with the comments of the "ISHCShift" instance and their language code IDs.

- `languageId`
[in]: The language code ID
- `pComments`
[in]: The comment text

Supplies the duration of the "ISHCShift" instance.

- pDuration
[out]: The duration of the shift

Sets the duration of the "ISHCShift" instance.

- duration
[in]: The duration of the shift

Supplies the "ISHCShiftTemplate" instance on which the "ISHCShift" instance is based.

CRFESULT GetShiftTemplate(CFSTR* pSHCShiftTemplate)

- pSHCShiftTemplate
[out]: The shift template

"CreateAction" method

Adds an "ISHCAction" instance to the "ISHCShift" instance.

CRFESULT CreateAction(ISHCActionTemplate* pActionTemplate,
CFTIMESPAN64 offset, ISHCAction** pShcAction)

- pActionTemplate
[in]: The action template on which the new action is to be based.
- offset
[in]: The offset of the action in relation to the start time of the "ISHCShift" instance. Positive and negative value allowed.
- pShcAction
[out]: The new action

"DeleteAction" method

Deletes an "ISHCAction" instance of the "ISHCShift" instance.

CFRESULT DeleteAction(ISHCAction* pShcAction)

- pShcAction
[in]: Reference to the action to be deleted.

"GetAction" method

Supplies an "ISHCActionEnumerator" instance via which you access the "ISHCAction" instances of the "ISHCShift" instance.

CRFESULT GetActions(ISHCActionEnumerator** ppSHCActionEnumerator)

- ppSHCActionEnumerator
[out]: The enumerator

"GetIsCustomized" method

Supplies the information on whether the "ISHCShift" instance was edited by users.

CRFESULT GetIsCustomized)(CFBOOL* pIsCustomized)

- pIsCustomized
[out]:
 - 0: Was not processed
 - 1: Was processed

"GetDeltaKind" method

Supplies the delta type of the "ISHCShift" instance.

```
CFRESULT GetDeltaKind(CFENUM* pDeltaKind)
```

- `pDeltaKind`
[out]: Points to the enumeration "ShcDeltaType", which can contain the following values:
 - Added (0)
 - Modified (1)
 - Deleted (2)

"GetShiftId" method

Supplies the ID of the "ISHCShift" instance.

```
CFRESULT GetShiftId(uint32_t* pShiftId)
```

- `pShift`
[out]: The ID

"SetShiftId" method

Sets the ID of the "ISHCShift" instance.

```
SetShiftId(uint32_t ShiftId)
```

- `ShiftId`
[in]: The new ID

See also

Locale IDs of the supported languages (Page 21)

9.7.18 ISHCShiftEnumerator

Description

The C++ interface "ISHCShiftEnumerator" specifies methods for handling the enumeration of shifts of an "ISHCDay" instance or "ISHCDayTemplate" instance.

The enumeration is returned by the "GetShifts" method of these instances.

The interface inherits from the "ICfUnknown" interface.

Members

"MoveNext" method

Go to the next element of the enumeration.

```
CFRESULT MoveNext()
```

"Current" method

Output the current element of the enumeration.

CFRESULT Current(ISHCShift** ppItem)

- ppItem
[out]: The current shift

"Reset" method

Reset the current position in the enumeration. The "MoveNext" method moves afterwards to the first element.

CFRESULT Reset()

"Count" method

Output the size of the enumeration or the number of its elements.

CRFRESULT Count(uint32_t* pCount)

- pCount
[out]: Number of shifts

Example**Copy code**

```

void printShift(const ISHCShiftEnumeratorPtr& p_pShcShiftEnum)
{
    std::cout << endl << "*****Print Shift*****" << std::endl << endl; if
(p_pShcShiftEnum != nullptr)
    {
        CFRESULT hr = CF_ERROR;
        uint32_t nCout = 0;
        p_pShcShiftEnum->Count(&nCout);
        for (uint32_t nIndex = 0; nIndex < nCout; nIndex++)
        {
            cout << endl;
            if (CF_SUCCEEDED(p_pShcShiftEnum->MoveNext()))
            {
                ISHCShiftPtr pShcshift;
                p_pShcShiftEnum->Current(&pShcshift);
                if (pShcshift != nullptr)
                {
                    CCfString strshiftTemplateName;
                    hr = pShcshift->GetShiftTemplate(&strshiftTemplateName);
                    if (CF_SUCCEEDED(hr) && (!strCategoryName.IsEmpty()))
                    {
                        std::cout << "ShiftTemplateName= " <<
strshiftTemplateName.ToUTF8().c_str() << std::endl;
                    }
                    CCfTimeSpan64 tsDuration;
                    hr = pShcshift->GetDuration(&tsDuration);
                    if (CF_SUCCEEDED(hr))
                    {
                        CCfString strDuration = tsDuration.GetTimeSpanString();
                        std::cout << "Durations= " << strDuration.ToUTF8().c_str() <<
std::endl;
                    }
                    CFBOOL bIsCustomised;
                    hr = pShcshift->GetIsCustomized(&bIsCustomised);
                    if (CF_SUCCEEDED(hr))
                    {
                        std::cout << "Is Customised= " << (uint32_t)bIsCustomised <<
std::endl;
                    }
                    CFENUM ndeltaKind;
                    hr = pShcshift->GetDeltaKind(&ndeltaKind);
                    if (CF_SUCCEEDED(hr))
                    {
                        std::cout << "deltaKind = " << ndeltaKind << std::endl;
                    }
                    uint32_t nShiftId;
                    hr = pShcshift->GetShiftId(&nShiftId);
                    if (CF_SUCCEEDED(hr))
                    {
                        std::cout << "ShiftId = " << nShiftId << std::endl;
                    }
                    ICfMapIDToVariantPtr pComments;
                    hr = pShcshift->GetComments(&pComments);
                    if (pComments != nullptr && CF_SUCCEEDED(hr))

```

Copy code

```
{
    std::cout << "comments:-" << std::endl << std::endl;
    uint32_t nCount = 0; pComments->Count(&nCount);
    for (uint32_t nIndex1 = 0; nIndex1 < nCount; nIndex1++)
    {
        int32_t nLanguageID;
        pComments->KeyAt(nIndex1, &nLanguageID);
        CCfVariant strComments;
        pComments->ValueAt(nLanguageID, &strComments);
        std::cout << "LangauageID =" << nLanguageID << " Comments=" <<
CCfSmartString(strComments).ToUTF8().c_str() << std::endl;
    }
}
}
}
}
```

9.7.19 ISHCAction

Description

The C++ interface "ISHCAction" specifies the methods of an action.
The interface inherits from the "ICfUnknown" interface.

Members

"GetOffset" method

Returns the offset of the "ISHCAction" instance in relation to the starting point of its "ISHCShift" instance.

CFRESULT GetOffset(CFTIMESPAN64* pOffsetType)

- pOffset
[out]: The offset in 100 nanoseconds. Positive and negative value allowed.

"SetOffset" method

Sets the offset of the "ISHCAction" instance in relation to the starting point of its "ISHCShift" instance.

CFRESULT GetOffset(CFTIMESPAN64 OffsetType)

- OffsetType
[in]: The offset in 100 nanoseconds. Positive and negative value allowed.

"GetActionTemplate" method

Supplies the "ISHCActionTemplate" instance of the "ISHCAction" instance.

```
GetActionTemplate (CFSTR* p_strActionTemplate)
```

- p_strActionTemplate
[out]: The action template

"GetIsCustomized" method

Supplies the information on whether the "ISHCAction" instance was edited by users.

```
CRFESULT GetIsCustomized (CFBOOL* pIsCustomized)
```

- pIsCustomized
[out]:
 - 0: Was not processed
 - 1: Was processed

"GetElements" method

Supplies an "ISHCActionElementEnumerator" instance via which you access the action elements of the "ISHCAction" instance.

```
GetElements (ISHCActionElementEnumerator**  
ppSHCActionElementEnumerator)
```

- ppSHCActionElementEnumerator
[out]: The enumerator

9.7.20 ISHCActionEnumerator

Description

The C++ interface "ISHCActionEnumerator" specifies methods for handling the enumeration of actions of an "ISHCShift" instance or "ISHCShiftTemplate" instance.

The enumeration is returned by the "GetActions" method of these instances.

The interface inherits from the "IcfUnknown" interface.

Members

"MoveNext" method

Go to the next element of the enumeration.

```
CFRESULT MoveNext ()
```

"Current" method

Output the current element of the enumeration.


```
CFRESULT Current(ISHCAction** ppItem)
```

- ppItem
[out]: The current action

"Reset" method

Reset the current position in the enumeration. The "MoveNext" method moves afterwards to the first element.

```
CFRESULT Reset()
```

"Count" method

Output the size of the enumeration or the number of its elements.

```
CRFRESULT Count(uint32_t* pCount)
```

- pCount
[out]: Number of actions

Example**Copy code**

```

void PrintAction(const ISHCActionEnumeratorPtr& pShcActionEnum)
{
    cout << endl << "*****PrintAction*****" << endl << endl; if (pShcActionEnum !=
nullptr)
    {
        CFRESULT hr = CF_ERROR;
        uint32_t nCount = 0;
        pShcActionEnum->Count(&nCount);
        if (nCount > 0)
        {
            for (uint32_t nIndex = 0; nIndex < nCount; nIndex++)
            {
                cout << endl; if (CF_SUCCEEDED(pShcActionEnum->MoveNext()))
                {
                    ISHCActionPtr pShcAction;
                    pShcActionEnum->Current(&pShcAction);
                    if (pShcAction != nullptr)
                    {
                        CCfString strActionTemplateName;
                        hr = pShcAction->GetActionTemplate(&strActionTemplateName);
                        if (CF_SUCCEEDED(hr))
                        {
                            std::cout << "ActionTemplateName = " <<
strActionTemplateName.ToUTF8().c_str() << std::endl;
                        }
                        CFBOOL isCustomize;
                        hr = pShcAction->GetIsCustomized(&isCustomize);
                        if (CF_SUCCEEDED(hr))
                        {
                            cout << "IsCustomize= " << (uint32_t)isCustomize << endl;
                        }
                        CCfTimeSpan64 offset;
                        hr = pShcAction->GetOffset(&offset);
                        if (CF_SUCCEEDED(hr))
                        {
                            cout << "Offset=" << offset.GetTimeSpanString().ToUTF8().c_str()
<< endl;
                        }
                        ISHCActionElementEnumeratorPtr pShcActionElementEnum;
                        hr = pShcAction->GetElements(&pShcActionElementEnum);
                        if (CF_SUCCEEDED(hr))
                        {
                            PrintActionElement(pShcActionElementEnum);
                        }
                    }
                }
            }
        }
    }
}

```

9.7.21 ISHCActionElement

Description

The C++ interface "ISHCActionElement" specifies the methods of the action elements of an "ISHCAction" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"GetElementType" method

Supplies the type of the "ISHCActionElement" instance.

CRFESULT GetElementType(CFENUM* pElementType)

- pElementType
[out]: Points to the enumeration "ShcActionElementType", which can contain the following values:
 - Tag (0)
The action element controls a tag.

"GetEnabled" method

Supplies the information on whether the "ISHCActionElement" instance is activated.

CRFESULT GetEnabled(CFBOOL* pEnabled)

- pEnabled
[out]:
 - 0: Deactivated
 - 1: Activated

"SetEnabled" method

Sets whether the "ISHCActionElement" instance is activated.

CRFESULT SetEnabled(CFBOOL Enabled)

- Enabled
[in]:
 - 0: Deactivated
 - 1: Activated

"GetOffset" method

Returns the offset of the "ISHCActionElement" instance in relation to the anchor point of its "ISHCAction" instance.

CFRESULT GetOffset(CFTIMESPAN64* pOffset)

- pOffset
[out]: The offset in 100 nanoseconds. Positive and negative value allowed.

"SetOffset" method

Sets the offset of the "ISHCActionElement" instance in relation to the anchor point of its "ISHCAction" instance.

```
CFRESULT SetOffset(CFTIMESPAN64 offset)
```

- offset
[in]: The offset in 100 nanoseconds. Positive and negative value allowed.

"GetValue" method

Supplies the value of the tag controlled by the "ISHCActionElement" instance.

```
CFRESULT GetValue(CFVARIANT* pValue)
```

- pValue
[out]: The tag value

"SetValue" method

Sets the value of the tag controlled by the "ISHCActionElement" instance.

```
CRFRESULT SetValue(CFVARIANT value)
```

- value
[in]: The new tag value

"GetElementName" method

Supplies the name of the tag controlled by the "ISHCActionElement" instance.

```
CFRESULT GetElementName(CFSTR* p_strActionElement)
```

- p_strActionElement
[out]: The tag name

"SetElementName" method

Sets the name of the tag controlled by the "ISHCActionElement" instance.

```
CFRESULT SetElementName(CFSTR ActionElement)
```

- ActionElement
[in]: The tag name

9.7.22 ISHCActionElementEnumerator

Description

The C++ interface "ISHCActionElementEnumerator" specifies methods for handling the enumeration of action elements of an "ISHCAction" instance.

The enumeration is returned by the "GetElements" method of an "ISHCAction" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"MoveNext" method

Go to the next element of the enumeration.

```
CFRESULT MoveNext()
```

"Current" method

Output the current element of the enumeration.

```
CFRESULT Current(ISHCActionElement** ppItem)
```

- ppItem
[out]: The current action element

"Reset" method

Reset the current position in enumeration. The "MoveNext" method moves afterwards to the first element.

```
CFRESULT Reset()
```

"Count" method

Output the size of the enumeration or the number of elements.

```
CRFRESULT Count(uint32_t* pCount)
```

- pCount
[out]: Number of action elements

Example**Copy code**

```

void PrintActionElement(const ISHCActionElementEnumeratorPtr& pActionElementEnum)
{
    cout << endl << "*****PrintActionElement*****" << endl << endl;
    if (pActionElementEnum != nullptr)
    {
        CFRESULT hr = CF_ERROR;
        uint32_t nCount = 0;
        pActionElementEnum->Count(&nCount);
        if (nCount > 0)
        {
            for (uint32_t nIndex = 0; nIndex < nCount; nIndex++)
            {
                cout << endl;
                if (CF_SUCCEEDED(pActionElementEnum->MoveNext()))
                {
                    ISHCActionElementPtr pShcActionElement;
                    pActionElementEnum->Current(&pShcActionElement);
                    if (pShcActionElement != nullptr)
                    {
                        CCfString strElementName;
                        hr = pShcActionElement->GetElementName(&strElementName);
                        if (CF_SUCCEEDED(hr))
                        {
                            cout << "ActionElementName= " << strElementName.ToUTF8().c_str()
<< endl;

                                }
                                CFENUM nType;
                                hr = pShcActionElement->GetElementType(&nType);
                                if (CF_SUCCEEDED(hr))
                                {
                                    cout << "Element Type= " << nType << endl;
                                }
                                CFBOOL bIsEnable;
                                hr = pShcActionElement->GetEnabled(&bIsEnable);
                                if (CF_SUCCEEDED(hr))
                                {
                                    cout << "Is Enable = " << (uint32_t)bIsEnable << endl;
                                }
                                CCfTimeSpan64 offset;
                                hr = pShcActionElement->GetOffset(&offset);
                                if (CF_SUCCEEDED(hr))
                                {
                                    cout << "Offset = " << offset.GetTimeSpanString().ToUTF8().c_str()
<< endl;

                                        }
                                        CCfVariant vtValue;
                                        hr = pShcActionElement->GetValue(&vtValue);
                                        if (CF_SUCCEEDED(hr))
                                        {
                                            cout << "value = " << vtValue.uint32 << endl;
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}

```

Copy code

```
}  
}  
}
```

9.7.23 ISHCActionTemplate

Description

The C++ interface "ISHCActionTemplate" specifies the methods of an action template. The interface inherits from the "ICfUnknown" interface.

Members

"GetName" method

Supplies the name of the "ISHCActionTemplate" instance.

CFRESULT GetName(CFSTR * pvarRet)

- pvarRet
[out]: The name of the action template

"SetName" method

Sets the name of the "ISHCActionTemplate" instance

CFRESULT SetName(CFSTR value)

- value
[in]: The name

"GetDisplayNames" method

Supplies a map with the display names of the "ISHCActionTemplate" instance and their language code IDs.

CFRESULT GetDisplayNames(ICfMapIDToVariant** ppDisplayNames)

- ppDisplayNames
[out]: The map with String/String pairs (language code ID for display name).

"SetDisplayNames" method

Adds a display name and its language code ID to the map with the display name of the "ISHCActionTemplate" instance.

CFRESULT SetDisplayNames(CFLCID languageId, CFSTR pDisplayName)

- languageID
[in]: The language code ID of the display name
- pDisplayName
[in]: The display name

"GetDescriptions" method

Supplies a map with the descriptions of the "ISHCActionTemplate" instance and their language code IDs.

```
CFRESULT GetDescriptions(ICfMapIDToVariant** value)
```

- value
[out]: The map with String/String pairs (language code ID for description).

"SetDescriptions" method

Adds a description and its language code ID to the map with the descriptions of the "ISHCActionTemplate" instance.

```
CFRESULT SetDescription(CFLCID languageId, CFSTR pDescriptions)
```

- languageID
[in]: The language code ID of the description
- pDescriptions
[in]: The description text

"GetIsDeleted" method

Supplies the information on whether the "ISHCActionTemplate" instance was deleted by users.

```
CFRESULT GetIsDeleted(CFBOOL* pIsDeleted)
```

- pIsDeleted
[out]:
 - 0: Was not deleted
 - 1: Was deleted

"GetElements" method

Supplies an "ISHCActionTemplateElementEnumerator" instance. The enumerator provides you with access to the "ISHCActionTemplateElement" instances of the "ISHCActionTemplate" instance.

```
CRFRESULT GetElements(ISHCActionTemplateElementEnumerator**  
ppISHCActionTemplateElementEnumerator)
```

- ppISHCActionTemplateElementEnumerator
[out]: The enumerator

"CreateElement" method

Adds an "ISHCActionTemplateElement" instance to the "ISHCActionTemplate" instance.

```
CRFRESULT CreateElement(ISHCActionTemplateElement*  
pSHCActionTemplateElement)
```

- pSHCActionTemplateElement
[in]: Reference to the new action element

"DeleteElement" method

Deletes an action element of the "ISHCActionTemplate" instance.


```
CRFESULT DeleteElement (ISHCActionTemplateElement*  
pSHCActionTemplateElement)
```

- pSHCActionTemplateElement
[in]: Reference to the action element to be deleted

See also

Locale IDs of the supported languages (Page 21)

9.7.24 ISHCActionTemplateEnumerator

Description

The C++ interface "ISHCActionTemplateEnumerator" specifies methods for handling the enumeration of the action templates of an "ISHCCalendar" instance.

The enumeration is returned by the "Read" method of an "ISHCActionTemplatesProvider" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"MoveNext" method

Go to the next element of the enumeration.

```
CFRESULT MoveNext ()
```

"Current" method

Output the current element of the enumeration.

```
CFRESULT Current (ISHCActionTemplate** ppItem)
```

- ppItem
[out]: The current action template

"Reset" method

Reset the current position in the enumeration. The "MoveNext" method moves afterwards to the first element.

```
CFRESULT Reset ()
```

"Count" method

Output the size of the enumeration or the number of elements.

```
CRFESULT Count (uint32_t* pCount)
```

- pCount
[out]: Number of action templates

Example**Copy code**

```

void PrintActionTemplate(const ISHCActionTemplateEnumeratorPtr& p_pShcActionTemplateEnum)
{
    cout << endl << "*****PrintActionTemplate*****" << endl << endl;
    if (p_pShcActionTemplateEnum != nullptr)
    {
        CFRESULT hr = CF_ERROR;
        uint32_t nCount = 0;
        p_pShcActionTemplateEnum->Count(&nCount);
        for (uint32_t nIndex = 0; nIndex < nCount; nIndex++)
        {
            cout << endl;
            if (CF_SUCCEEDED(p_pShcActionTemplateEnum->MoveNext()))
            {
                ISHCActionTemplatePtr pShcActionTemplate;
                p_pShcActionTemplateEnum->Current(&pShcActionTemplate);
                if (pShcActionTemplate != nullptr)
                {
                    CCfString strActionTemplateName;
                    hr = pShcActionTemplate->GetName(&strActionTemplateName);
                    if (CF_SUCCEEDED(hr))
                    {
                        cout << "ActionTemplateName=" <<
strActionTemplateName.ToUTF8().c_str() << endl;
                    }
                    CFBOOL bIsDeleted;
                    hr = pShcActionTemplate->GetIsDeleted(&bIsDeleted);
                    if (CF_SUCCEEDED(hr))
                    {
                        cout << "IsDeleted=" << (uint32_t)bIsDeleted << endl;
                    }
                    ICfMapIDToVariantPtr pDescriptions;
                    hr = pShcActionTemplate->GetDescriptions(&pDescriptions);
                    if (pDescriptions != nullptr && CF_SUCCEEDED(hr))
                    {
                        std::cout << "Descriptions=" << std::endl << std::endl;
                        uint32_t nCount1 = 0;
                        pDescriptions->Count(&nCount1);
                        for (uint32_t nIndex1 = 0; nIndex1 < nCount1; nIndex1++)
                        {
                            int32_t nLanguageID;
                            pDescriptions->KeyAt(nIndex1, &nLanguageID);
                            CCfVariant strDescription;
                            pDescriptions->ValueAt(nLanguageID, &strDescription);
                            std::cout << "LangauageID =" << nLanguageID << " Description="
<< CCfSmartString(strDescription).ToUTF8().c_str() << std::endl;
                        }
                    }
                    ICfMapIDToVariantPtr pDisplayNames;
                    hr = pShcActionTemplate->GetDisplayNames(&pDisplayNames);
                    if (pDisplayNames != nullptr && CF_SUCCEEDED(hr))
                    {
                        std::cout << "DisplayNames:=" << std::endl << std::endl;
                        uint32_t
nCount1 = 0; pDisplayNames->Count(&nCount1);
                        for (uint32_t nIndex1 = 0; nIndex1 < nCount1; nIndex1++)

```

Copy code

```
{
    int32_t nLanguageID;
    pDisplayNames->KeyAt(nIndex1, &nLanguageID);
    CCfVariant strDisplayName;
    pDisplayNames->ValueAt(nLanguageID, &strDisplayName);
    std::cout << "LangauageID =" << nLanguageID << " DisplayName="
<< CCfSmartString(strDisplayName).ToUTF8().c_str() << std::endl;
}

}

ISHCActionTemplateElementEnumeratorPtr pShcActionTemplateElementEnum;
hr = pShcActionTemplate->GetElements(&pShcActionTemplateElementEnum);
if (CF_SUCCEEDED(hr))
PrintActionTemplateElement(pShcActionTemplateElementEnum);
}

}

}

}
```

9.7.25 ISHCActionTemplatesProvider

Description

The C++ interface "ISHCActionTemplatesProvider" provides you with access to an "ISHCActionTemplateEnumerator" instance which contains an enumeration with the action templates of an "ISHCCalendar" instance. With the methods of the provider, you can create, read, update and delete action templates. The provider is returned by the "GetActionTemplatesProvider" method of an "ISHCCalendar" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"Browse" method

Supplies an "ISHCActionTemplateEnumerator" instance which has access to an enumeration with the "ISHCActionTemplate" instances of the calendar.

```
CRFRESULT Browse(CFBOOL includeDeleted, OUT
ISHCActionTemplateEnumerator** ppISHCActionTemplateEnumerator)
```

- includeDeleted
Saves whether the enumerator also has access to the deleted action templates.
- ppISHCActionTemplateEnumerator
[out]: The enumerator

Example:

```
ISHCActionTemplateEnumeratorPtr pShcActionTemplateEnum;
CFRESULT hr = pShcActionTemplateProvider->Browse(CF_FALSE,
&pShcActionTemplateEnum);
```

"Create" method

Adds new action templates to the enumeration with the "ISHCActionTemplate" instances of the calendar.

```
CFRESULT Create(ICfArrayIUnknown* pActionTemplates)
```

- `pActionTemplates`
[in]: An array with the action templates to be added.

Example:

Copy code

```

void CreateActionTemplateWithActionTemplateElement()
{
    if (nullptr != pCalendar && nullptr != pShcActionTemplateProvider)
    {
        CFRESULT hr = CF_ERROR;
        ICfUnknownPtr pUnk;
        hr = pCalendar->GetObject(ODK_SHC_ACTION_TEMPLATE, &pUnk);
        CCfArrayIUnknown ArrayTemplate;
        ICfArrayIUnknownPtr pArrayTemplate;
        ISHCActionTemplatePtr pShcActionTemplate = (ISHCActionTemplatePtr)pUnk;
        if (pShcActionTemplate != nullptr)
        {
            hr = pShcActionTemplate->SetName(CCfString(L"ActionTemplateName"));
            if (CF_FAILED(hr))
            {
                cout << "set name failed" << endl;
            }
            hr = pShcActionTemplate->SetDisplayName(1033,
CCfString(L"ActionTemplateDisplayName"));
            if (CF_FAILED(hr))
            {
                std::cout << "failed to SetDisplayName" << std::endl;
            }
            hr = pShcActionTemplate->SetDescription(1033,
CCfString(L"ActionTemplateDescription"));
            if (CF_FAILED(hr))
            {
                std::cout << "failed to SetDescription" << std::endl;
            }
            ArrayTemplate.Append(pShcActionTemplate);
        }
        ArrayTemplate.DetachEnumerator(&pArrayTemplate);
        hr = pShcActionTemplateProvider->Create(pArrayTemplate);
        if (CF_SUCCEEDED(hr))
        {
            hr = pCalendar->GetObject(ODK_SHC_ACTION_TEMPLATE_ELEMENT, &pUnk);
            ISHCActionTemplateElementPtr pShcActionTemplateElement =
(ISHCActionTemplateElementPtr)pUnk;
            if (nullptr != pShcActionTemplateElement)
            {
                hr = pShcActionTemplateElement-
>SetElementName(CCfString(L"HMI_RT_1::Unit1.Member_1"));
                if (CF_FAILED(hr))
                {
                    std::cout << "failed to SetElementName" << std::endl;
                }
                CFTIMESPAN64 Offset = Get1Hour();
                hr = pShcActionTemplateElement->SetOffset(Offset);
                if (CF_FAILED(hr))
                {
                    std::cout << "failed to SetOffset" << std::endl;
                }
                uint32_t value = 1;
                hr = pShcActionTemplateElement->SetValue(CCfVariant(value));
                if (CF_FAILED(hr))

```

Copy code

```
        {
            std::cout << "failed to SetValue" << std::endl;
        }
    }
    hr = pShcActionTemplate->CreateElement(pShcActionTemplateElement);
    if (CF_FAILED(hr))
    {
        std::cout << "failed to CreateElement" << std::endl;
    }
}
}
```

"Update" method

Updates "ISHCActionTemplate" instances of the enumeration.

CRFESULTUpdate(ICfArrayIUnknown* pActionTemplates)

- pActionTemplates
[in]: An array with the action templates to be updated.

Example:

Copy code

```
void UpdateActionTemplateWithActionTemplateElement()
{
    if (pShcActionTemplateProvider != nullptr)
    {
        CFRESULT hr = CF_ERROR;
        ISHCActionTemplateEnumeratorPtr pShcActionTemplateEnum;
        hr = pShcActionTemplateProvider->Browse(CF_FALSE, &pShcActionTemplateEnum);
        if (CF_SUCCEEDED(hr) && pShcActionTemplateEnum != nullptr)
        {
            hr = pShcActionTemplateEnum->MoveNext();
            if (CF_SUCCEEDED(hr))
            {
                CCfArrayIUnknown ArrayActionTemplate;
                ICfArrayIUnknownPtr pArrayActionTemplate;
                ISHCActionTemplatePtr pShcActionTemplate;
                pShcActionTemplateEnum->Current(&pShcActionTemplate);
                if (pShcActionTemplate != nullptr)
                {
                    hr = pShcActionTemplate->SetName(CCfString(L"UpdatedActionTemplate"));
                    if (CF_FAILED(hr))
                    {
                        std::cout << "failed to SetName" << std::endl;
                    }
                    hr = pShcActionTemplate->SetDisplayName(1033,
CCfString(L"UpdatedActionTemplateDisplayName"));
                    if (CF_FAILED(hr))
                    {
                        std::cout << "failed to SetDisplayName" << std::endl;
                    }
                    ISHCActionTemplateElementEnumeratorPtr pShcActionTemplateElementEnum;
                    pShcActionTemplate->GetElements(&pShcActionTemplateElementEnum);
                    if (pShcActionTemplateElementEnum != nullptr)
                    {
                        hr = pShcActionTemplateElementEnum->MoveNext();
                        if (CF_SUCCEEDED(hr))
                        {
                            ISHCActionTemplateElementPtr pShcActionTemplateElement;
                            pShcActionTemplateElementEnum-
>Current(&pShcActionTemplateElement);
                            if (pShcActionTemplateElement != nullptr)
                            {
                                CCfTimeSpan64 offset = Get1Hour() * 2;
                                hr = pShcActionTemplateElement->SetOffset(offset);
                                if (CF_FAILED(hr))
                                {
                                    std::cout << "failed to SetOffset" << std::endl;
                                }
                                uint32_t nValue = 2;
                                hr = pShcActionTemplateElement->SetValue(CCfVariant(nValue));
                                if (CF_FAILED(hr))
                                {
                                    std::cout << "failed to SetValue" << std::endl;
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

Copy code

```
    }
    ArrayActionTemplate.Append(pShcActionTemplate);
    ArrayActionTemplate.DetachEnumerator(&pArrayActionTemplate);
    hr = pShcActionTemplateProvider->Update(pArrayActionTemplate);
    if (CF_FAILED(hr))
    {
        std::cout << "failed to Update" << std::endl;
    }
}
}
}
}
```

"Delete" method

Deletes "ISHCActionTemplate" instances of the calendar from the enumeration.

CRFESULT Delete(ICfArrayIUnknown* pActionTemplates)

- pActionTemplates
[in]: An array with the action templates to be deleted.

Example:

Copy code

```
void DeleteActionTemplateWithActionTemplateElement()
{
    if (pShcActionTemplateProvider != nullptr)
    {
        CFRESULT hr = CF_ERROR;
        ISHCActionTemplateEnumeratorPtr pShcActionTemplateEnum;
        hr = pShcActionTemplateProvider->Browse(CF_FALSE, &pShcActionTemplateEnum);
        if (CF_SUCCEEDED(hr) && pShcActionTemplateEnum != nullptr)
        {
            hr = pShcActionTemplateEnum->MoveNext();
            if (CF_SUCCEEDED(hr))
            {
                CCfArrayIUnknown ArrayActionTemplate;
                ICfArrayIUnknownPtr pArrayActionTemplate;
                ISHCActionTemplatePtr pShcActionTemplate;
                pShcActionTemplateEnum->Current(&pShcActionTemplate);
                if (pShcActionTemplate != nullptr)
                {
                    ISHCActionTemplateElementEnumeratorPtr pShcActionTemplateElementEnum;
                    pShcActionTemplate->GetElements(&pShcActionTemplateElementEnum);
                    if (pShcActionTemplateElementEnum != nullptr)
                    {
                        hr = pShcActionTemplateElementEnum->MoveNext();
                        if (CF_SUCCEEDED(hr))
                        {
                            ISHCActionTemplateElementPtr pShcActionTemplateElement;
                            pShcActionTemplateElementEnum->Current(&pShcActionTemplateElement);
                            if (pShcActionTemplateElement != nullptr)
                            {
                                hr = pShcActionTemplate->DeleteElement(pShcActionTemplateElement);
                                if (CF_FAILED(hr))
                                {
                                    std::cout << "failed to DeleteElement" << std::endl;
                                }
                            }
                        }
                    }
                    ArrayActionTemplate.Append(pShcActionTemplate);
                    ArrayActionTemplate.DetachEnumerator(&pArrayActionTemplate);
                    hr = pShcActionTemplateProvider->Delete(pArrayActionTemplate);
                    if (CF_FAILED(hr))
                    {
                        std::cout << "failed to Delete" << std::endl;
                    }
                }
            }
        }
    }
}
```

9.7.26 ISHCActionTemplateElement

Description

The C++ interface "ISHCActionTemplateElement" specifies the methods of an action template. The interface inherits from the "ICfUnknown" interface.

Members

"GetElementType" method

Supplies the type of the "ISHCActionTemplateElement" instance.

```
CFRESULT GetElementType(CFENUM* pElementType)
```

- pElementType
[out]: Points to the enumeration "ShcActionElementType", which can contain the following values:
 - Tag (0)
The action element controls a tag.

"GetOffset" method

Returns the offset of the "ISHCActionTemplateElement" instance in relation to the anchor point of its "ISHCActionTemplate" instance.

```
CFRESULT GetOffset(CFTIMESPAN64* pOffset)
```

- pOffset
[out]: The offset in 100 nanoseconds. Positive and negative value allowed.

"SetOffset" method

Sets the offset of the "ISHCActionTemplateElement" instance in relation to the anchor point of its "ISHCActionTemplate" instance.

```
CFRESULT SetOffset(CFTIMESPAN64 offset)
```

- offset
[in]: The offset in 100 nanoseconds. Positive and negative value allowed.

"GetElementName" method

Supplies the name of the tag controlled by the "ISHCActionTemplateElement" instance.

```
CFRESULT GetElementName(CFSTR* pElementName)
```

- pElementName
[out]: The tag name

"SetElementName" method

Sets the name of the tag controlled by the "ISHCActionTemplateElement" instance.

```
CFRESULT SetElementName(CFSTR pElementName)
```

- pElementName
[in]: The tag name

"GetValue" method

Supplies the value of the tag controlled by the "ISHCActionTemplateElement" instance.

```
CFRESULT GetValue(CFVARIANT* pValue)
```

- pValue
[out]: The tag value

"SetValue" method

Sets the value of the tag controlled by the "ISHCActionTemplateElement" instance.

```
CRFRESULT SetValue(CFVARIANT pValue)
```

- pValue
[in]: The new tag value

9.7.27 ISHCActionTemplateElementEnumerator

Description

The C++ interface "ISHCActionTemplateElementEnumerator" specifies methods for handling the enumeration of "ISHCActionTemplateElement" instances of an "ISHCActionTemplate" instance.

The enumeration is returned by the "GetElements" method of an "ISHCActionTemplate" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"MoveNext" method

Go to the next element of the enumeration.

```
CFRESULT MoveNext()
```

"Current" method

Output the current element of the enumeration.

```
CFRESULT Current(ISHCActionTemplateElement** ppItem)
```

- ppItem
[out]: The current action element of the action template

"Reset" method

Reset the current position in the enumeration. The "MoveNext" method moves afterwards to the first element.

```
CFRESULT Reset()
```

"Count" method

Output the size of the enumeration or the number of elements.

```
CRFESULT Count(uint32_t* pCount)
```

- pCount
[out]: Number of "ISHCAActionTemplateElement" instances

Example

Copy code

```
void PrintActionTemplateElement(const ISHCActionTemplateElementEnumeratorPtr&
p_pShcActionTemplateElementEnum)
{
    cout << endl << "****PrintActionTemplateElement*****" << endl << endl;
    if (p_pShcActionTemplateElementEnum != nullptr)
    {
        CFRESULT hr = CF_ERROR;
        uint32_t nCount = 0;
        p_pShcActionTemplateElementEnum->Count(&nCount);
        for (uint32_t nIndex = 0; nIndex < nCount; nIndex++)
        {
            cout << endl; if (CF_SUCCEEDED(p_pShcActionTemplateElementEnum->MoveNext()))
            {
                ISHCActionTemplateElementPtr pShcActionTemplateElement;
                p_pShcActionTemplateElementEnum->Current(&pShcActionTemplateElement);
                if (pShcActionTemplateElement != nullptr)
                {
                    CCfString strElementName;
                    hr = pShcActionTemplateElement->GetElementName(&strElementName);
                    if (CF_SUCCEEDED(hr))
                    {
                        cout << "TemplateElementName= " << strElementName.ToUTF8().c_str()
<< endl;
                    }
                    CFENUM type;
                    hr = pShcActionTemplateElement->GetElementType(&Type);
                    if (CF_SUCCEEDED(hr))
                    {
                        cout << "ElementType=" << Type << endl;
                    }
                    CCfTimeSpan64 tsOffset;
                    hr = pShcActionTemplateElement->GetOffset(&tsOffset);
                    if (CF_SUCCEEDED(hr))
                    {
                        cout << "Element Offset=" <<
tsOffset.GetTimeSpanString().ToUTF8().c_str() << endl;
                    }
                    CCfVariant vtValue;
                    hr = pShcActionTemplateElement->GetValue(&vtValue);
                    if (CF_SUCCEEDED(hr))
                    {
                        cout << "Value=" << vtValue.uint32 << endl;
                    }
                }
            }
        }
    }
}
```

9.8 Interfaces of the contexts

9.8.1 IContextLogging

Description

The C++ interface "IContextLogging" defines events and methods for creating and reading "IContextDefinition" instances as well as for starting, stopping, monitoring and reading their "ILoggedContext" instances. You can use "ILoggedContext" instances to filter runtime data, for example, for alarms that fall within the time period of a particular "ILoggedContext" instance.

The interface implements the methods of the "ICfUnknown" interface.

All methods return CF_SUCCESS after successful execution. In the case of an error, the methods return the corresponding error code.

Members

"CreateContextDefinitions" method

Creates ContextDefinitions in the database.

```
CRFESULT CreateContextDefinitions(ICfArrayIUnknown*
pContextDefinitions, IContextLoggingCallBack *pContextCallBack)
```

- pContextDefinitions:
[in]: Collection with "IContextDefinition" instances
- *pContextCallBack:
[in]: Points to the "IContextLoggingCallBack" object that implements the callback interface.

"ReadCreateContextDefinitions" method

Reads ContextDefinitions from the database. The instances can be filtered by plant object and HMIContextProviderType .

```
CRFESULT ReadContextDefinitions(IContextLoggingCallBack*
pContextCallBack, ICfArrayString* plantViewPaths, ICfArrayVariant*
pProviderTypes, CFENUM sortingMode)
```

- pContextCallBack:
[in]: Points to the "IContextLoggingCallBack" object that implements the callback interface.
- plantViewPaths:
[in/optional]: Limits the read operation to "IContextDefinition" instances from this collection of plant objects.

- `pProviderTypes`:
[in/optional]: Limits the read operation to "IContextDefinition" instances with HmiContextProvider types from this collection.
The enumeration "HmiContextProviderType" can contain the following values:
 - NoContext = 0
 - Calendar = 1
For "IContextDefinition" instances generated by the PI Option Calendar.
 - PerformanceInsightMachineState = 2
For "IContextDefinition" instances generated by the PI Option Performance Insight.
 - LineCoordinator = 3
For "IContextDefinition" instances generated by the PI Option Line Coordinator.
 - UserDefined = 5
It is a user-defined "IContextDefinition" instance, created by ODK, for example.
- `sortingMode`:
[in]: Points to the enumeration HmiContextLoggingSortingMode, which can contain the following values:
 - Ascending = 1
Default setting
 - Descending = 2

"ReadContexts" method

Reads "ILoggedContext" instances of a specific time period. The instances can be filtered using a "IContextFilter" instance.

```
CRFRESULT ReadContexts(CFDATE64 start, CFDATE64 end,
  IContextLoggingCallBack* ContextCallBack, IContextFilter*
  FilterObject, CFENUM type)
```

- `start`:
[in]: The start time of the period within which the "ILoggedContext" instances must lie.
- `end`:
[in]: The end time of the period within which the "ILoggedContext" instances must lie.
- `ContextCallBack`:
[in]: Points to the "IContextLoggingCallBack" object that implements the callback interface.
- `FilterObject`:
[in/optional]: The "IContextFilter" instance whose filter settings are used.
- `type`:
[in]: Points to the enumeration HmiContextLoggingSortingMode, which can contain the following values:
 - Ascending = 1
Default setting
 - Descending = 2

"StartContext" method

Creates a new "ILoggedContext" instance for a "IContextDefinition" instance.

9.8 Interfaces of the contexts

```
CRFESULT StartContext(CFSTR p_strContextName, CFENUM providerType,  
CFVARIANT contextValue, CFDATEETIME64 startTime, uint32_t qualityCode)
```

- `p_strContextName`:
[in]: The name of the "IContextDefinition" instance for which the context log entry is created.
- `providerType`:
[in]: The HmiContextProviderType that the "IContextDefinition" instance of the context log entry must have.
The enumeration "HmiContextProviderType" can contain the following values:
 - NoContext = 0
 - Calendar = 1
For "IContextDefinition" instances generated by the PI Option Calendar.
 - PerformanceInsightMachineState = 2
For "IContextDefinition" instances generated by the PI Option Performance Insight.
 - LineCoordinator = 3
For "IContextDefinition" instances generated by the PI Option Line Coordinator.
 - UserDefined = 5
It is a user-defined "IContextDefinition" instance, created by ODK, for example.
- `contextValue`:
[in]: The context value of the context log entry
- `startTime`:
[in]: Start time of the new context log entry
- `qualityCode`:
[in]: The QualityCode of the context value of the context log entry

"StopContext" method

Stops the currently running "ILoggedContext" instance of an "IContextDefinition" instance.

```
CRFESULT StopContext(CFSTR p_strContextName, CFENUM providerType,
CFDATEETIME64 endtime)
```

- `p_strContextName`:
[in]: The name of the "IContextDefinition" instance whose context log entry is stopped.
- `providerType`:
[in]: The HmiContextProviderType that the "IContextDefinition" instance of the context log entry must have.
The enumeration "HmiContextProviderType" can contain the following values:
 - NoContext = 0
 - Calendar = 1
For "IContextDefinition" instances generated by the PI Option Calendar.
 - PerformanceInsightMachineState = 2
For "IContextDefinition" instances generated by the PI Option Performance Insight.
 - LineCoordinator = 3
For "IContextDefinition" instances generated by the PI Option Line Coordinator.
 - UserDefined = 5
It is a user-defined "IContextDefinition" instance, created by ODK, for example.
- `endtime`:
[in]: End time of the new context log entry

"Add" method

Adds a "IContextDefintion" instance to a vector. The methods Clear(), Subscribe() and CancelSubscription() can be called for the instances of the vector.

```
CRFESULT Add(CFSTR contextName, CFENUM type)
```

- `contextName`:
[in]: The name of the "IContextDefinition" instance
- `type`:
[in]: The HmiContextProviderType of the "IContextDefinition" instance
The enumeration "HmiContextProviderType" can contain the following values:
 - NoContext = 0
 - Calendar = 1
For "IContextDefinition" instances generated by the PI Option Calendar.
 - PerformanceInsightMachineState = 2
For "IContextDefinition" instances generated by the PI Option Performance Insight.
 - LineCoordinator = 3
For "IContextDefinition" instances generated by the PI Option Line Coordinator.
 - UserDefined = 5
It is a user-defined "IContextDefinition" instance, created by ODK, for example.

"Clear" method

Deletes the "IContextDefintion" instances added via Add() from the vector.

```
CRFESULT Clear()
```

"Subscribe" method

Subscribes the "IContextDefinition" instances added to the vector via `Add()` for monitoring.

CRFESULT `Subscribe(IContextLoggingCallBack* pContextLoggingCallback)`

- `pContextLoggingCallback`:
[in]: Points to the "IContextLoggingCallBack" object that implements the callback interface.

"CancelSubscribe" method

Unsubscribes the "IContextDefinition" instances added to the vector with `Add()` from monitoring.

CRFESULT `CancelSubscribe()`

Examples

Copying code

```
void CreateContextDefinitions(IRuntimePtr pRuntime)
{
    if (pRuntime != nullptr)
    {
        std::cout << std::endl << __FUNCTION__ << std::endl << std::endl;
        ICfUnknownPtr pUnk;
        CFRESULT errorCode = pRuntime->GetObject(CCfString(L"ContextLogging"), &pUnk);
        IContextLoggingPtr pContextLoggingPtr(pUnk);
        if (pContextLoggingPtr != nullptr && CF_SUCCEEDED(errorCode))
        {
            std::vector<IContextDefinitionPtr> vecContextDefinitions;
            for (size_t index = 0; index < 10; index++)
            {
                ICfUnknownPtr pUnkCd;
                errorCode = pRuntime->GetObject(CCfString(L"ContextDefinition"), &pUnkCd);
                IContextDefinitionPtr pContextDefinition(pUnkCd);
                if (nullptr != pContextDefinition)
                {
                    if (CF_FAILED(errorCode))
                        std::cout << "SetContextProviderType Failed" << std::endl;
                    CCfSmartString strID = std::to_string(std::rand());
                    CCfSmartString strName(L"CD-");
                    strName.Append(strID);
                    ICfMapIDToVariantPtr pDisplayNames;
                    std::map<int32_t, CCfVariant> DisplayNames;
                    CCfSmartString strDeuName(L"Deutsch-");
                    strDeuName.Append(strName);
                    DisplayNames[1031] = strDeuName;
                    CCfSmartString strEngName(L"English-");
                    strEngName.Append(strName);
                    DisplayNames[1033] = strEngName;
                    CCfMapIDToVariant::CreateEnumerator(DisplayNames, &pDisplayNames);
                    errorCode = pContextDefinition->SetDisplayNames(pDisplayNames);
                    if (CF_FAILED(errorCode))
                        std::cout << "SetDisplayNames Failed" << std::endl;
                    CCfString str_Name = L".hierarchy::Plant/Node1_1";
                    errorCode = pContextDefinition->SetPlantViewPath(str_Name);
                    if (CF_FAILED(errorCode))
                        std::cout << "SetPlantViewPath Failed" << std::endl;
                    CCfString m_strName = L"Contetx_";
                    m_strName.Append(strName.AllocCFSTR());
                    errorCode = pContextDefinition->SetName(m_strName);
                    if (CF_FAILED(errorCode))
                        std::cout << "SetName Failed" << std::endl;
                    errorCode = pContextDefinition->SetDataType(HmiContextDataType::String);
                    if (CF_FAILED(errorCode))
                        std::cout << "SetDataType Failed" << std::endl;
                    vecContextDefinitions.push_back(pContextDefinition);
                }
            }
            ICfArrayIUnknownPtr pContextDefinitionsArray;
            ::CfCreateEnumerator(vecContextDefinitions, &pContextDefinitionsArray);
            CContextLoggingCB * pContextLoggingCB = new CContextLoggingCB();
            pContextLoggingCB->AddRef();
        }
    }
}
```

Copying code

```

        errorCode = pContextLoggingPtr->CreateContextDefintions(pContextDefinitionsArray,
pContextLoggingCB);
        if (CF_FAILED(errorCode))
            std::cout << "CreateContextDefintions Failed" << std::endl;
        else
        {
            if (pContextLoggingCB->WaitForCompletion(MaxContextWaitTime) == CF_SUCCESS)
            {
                std::vector<IContextErrorPtr> vecContextErrors = pContextLoggingCB-
>GetContextErrors();
                DisplayContextError(vecContextErrors);
            }
            else
            {
                std::cout << " CreateContextDefintions TimeOut" << std::endl;
            }
        }
        pContextLoggingCB->Release();
    }
}

void DisplayContextError(const std::vector<IContextErrorPtr>& pVecContext)
{
    for (const auto& pContext : pVecContext)
    {
        if (nullptr != pContext)
        {
            CCfString strContextName;
            pContext->GetContextName(&strContextName);
            std::cout << "ContextName =" << strContextName.ToUTF8().c_str() << endl;
            uint32_t value;
            pContext->GetErrorCode(&value);
            std::cout << "ErrorCode=" << value << endl;
        }
    }
}

```

Copying code

```

void ReadContextDefinitionsWithFilter(IRuntimePtr pRuntime)
{
    if (pRuntime != nullptr)
    {
        std::cout << std::endl << __FUNCTION__ << std::endl << std::endl;
        ICfUnknownPtr pUnk;
        CFRESULT errorCode = pRuntime->GetObject(CcString(L"ContextLogging"), &pUnk);
        if (pUnk != nullptr && CF_SUCCEEDED(errorCode))
        {
            IContextLoggingPtr pContextLoggingPtr(pUnk);
            if (nullptr != pContextLoggingPtr)
            {
                CcString str_Name = L".hierarchy::Plant/Node1_1";
                ICfArrayStringPtr pArrayPlantObjects;
                CcArrayString ArrayPlantobjects;
                ArrayPlantobjects.Append(CcVariant(str_Name));
                ArrayPlantobjects.DetachEnumerator(&pArrayPlantObjects);
                CcVariant
vtProviderTypes(static_cast<CFENUM>(HmiContextProviderType::UserDefined));
                CcArrayVariant arrayProvidertypes;
                arrayProvidertypes.Append(vtProviderTypes);
                ICfArrayVariantPtr pArrayProvidertypes;
                arrayProvidertypes.DetachEnumerator(&pArrayProvidertypes);
                CContextLoggingCB * pContextLoggingCB = new CContextLoggingCB();
                pContextLoggingCB->AddRef();
                errorCode = pContextLoggingPtr->ReadContextDefinitions(pContextLoggingCB,
pArrayPlantObjects, pArrayProvidertypes);
                if (CF_FAILED(errorCode))
                    std::cout << "Read Context Failed" << std::endl;
                else
                {
                    if (pContextLoggingCB->WaitForcompletion(MaxContextWaitTime) ==
CF_SUCCESS)
                    {
                        std::vector<IContextDefinitionPtr> ContextDef = pContextLoggingCB-
>GetContextDefination();
                        DisplayContextDefinition(ContextDef);
                    }
                    pContextLoggingCB->Release();
                }
            }
        }
    }
}

void DisplayContextDefinition(const std::vector<IContextDefinitionPtr>& ContextDef)
{
    CFRESULT errorCode = CF_ERROR;
    for (auto& pValues : ContextDef)
    {
        uint32_t ErrorCode;
        pValues->GetErrorCode(&ErrorCode);
        std::cout << " ErrorCode = " << ErrorCode << std::endl;
    }
}

```

Copying code

```

    if (CF_SUCCEEDED(ErrorCode))
    {
        CCfString strName;
        errorCode = pValues->GetName(&strName);
        if (CF_SUCCEEDED(errorCode))
            std::cout << " GetName = " << strName.ToUTF8().c_str() << std::endl;
        CFENUM pnnum;
        errorCode = pValues->GetProviderType(&pnnum);
        if (CF_SUCCEEDED(errorCode))
            std::cout << " Provider Type= " << GetProviderType(pnnum).ToUTF8().c_str() <<
std::endl;
        HmiContextDataType dataType;
        errorCode = pValues->GetDataType(&dataType);
        if (CF_SUCCEEDED(errorCode))
            std::cout << " DataType Type= " << GetDataType(dataType).ToUTF8().c_str() <<
std::endl;
        ICfMapIDToVariantPtr displayName;
        errorCode = pValues->GetDisplayNames(&displayName);
        if (CF_SUCCEEDED(errorCode))
        {
            uint32_t nCount;
            displayName->Count(&nCount);
            std::cout << "Display Names= " << std::endl;
            for (uint32_t nIndex = 0; nIndex < nCount; nIndex++)
            {
                int32_t langID;
                displayName->KeyAt(index, &langID);
                CCfVariant vtName;
                displayName->ValueAt(langID, &vtName);
                std::cout << " " << "lang ID= " << langID << " DisplayName = " <<
CCfString(vtName).ToUTF8().c_str() << std::endl;
            }
        }
        CCfString strPlantViewPath;
        errorCode = pValues->GetPlantViewPath(&strPlantViewPath);
        if (CF_SUCCEEDED(errorCode))
            std::cout << " PlantViewPath = " << strPlantViewPath.ToUTF8().c_str() <<
std::endl;
    }
}
}

```


Copying code

```

void StartContext(IRuntimePtr pRuntime)
{
    if (pRuntime != nullptr)
    {
        std::cout << std::endl << __FUNCTION__ << std::endl << std::endl;
        ICfUnknownPtr pUnk;
        CFRESULT errorCode = pRuntime->GetObject(CCfString(L"ContextLogging"), &pUnk);
        if (pUnk != nullptr && CF_SUCCEEDED(errorCode))
        {
            IContextLoggingPtr pContextLoggingPtr(pUnk);
            if (nullptr != pContextLoggingPtr)
            {
                CCfDateTime64 dtStartTime = CCfDateTime64::Now();
                CCfVariant vtValue = L"Orange Juice";
                uint32_t nQuality = 192;
                CCfString strNameContextName = L"ContetxName_";
                strNameContextName.Append(strContext_Unique_Num);
                errorCode = pContextLoggingPtr->StartContext(strNameContextName,
static_cast<CFENUM>(HmiContextProviderType::UserDefined), vtValue, dtStartTime, nQuality);
                if (CF_FAILED(errorCode))
                    std::cout << "startContext failed for " <<
strNameContextName.ToUTF8().c_str() << std::endl;
            }
        }
    }
}

```

Copying code

```

void StopContext(IRuntimePtr pRuntime)
{
    if (pRuntime != nullptr)
    {
        std::cout << std::endl << __FUNCTION__ << std::endl << std::endl;
        ICfUnknownPtr pUnk;
        CFRESULT errorCode = pRuntime->GetObject(CCfString(L"ContextLogging"), &pUnk);
        if (pUnk != nullptr && CF_SUCCEEDED(errorCode))
        {
            IContextLoggingPtr pContextLoggingPtr(pUnk);
            if (nullptr != pContextLoggingPtr)
            {
                CCfDateTime64 dtEndTime = CCfDateTime64::Now();
                CCfString strNameContextName = L"ContetxName_";
                strNameContextName.Append(strContext_Unique_Num);
                errorCode = pContextLoggingPtr->StopContext(strNameContextName,
static_cast<CFENUM>(HmiContextProviderType::UserDefined), dtEndTime);
                if (CF_FAILED(errorCode))
                    std::cout << "stopContext failed for " <<
strNameContextName.ToUTF8().c_str() << std::endl;
            }
        }
    }
}

```

Copying code

```

void ReadContextWithFilter(IRuntimePtr pRuntime)
{
    if (pRuntime != nullptr)
    {
        std::cout << std::endl << __FUNCTION__ << std::endl << std::endl;
        ICfUnknownPtr pUnk;
        CFRESULT errorCode = pRuntime->GetObject(CCfString(L"ContextLogging"), &pUnk);
        if (pUnk != nullptr && CF_SUCCEEDED(errorCode))
        {
            IContextLoggingPtr pContextLoggingPtr(pUnk);
            if (nullptr != pContextLoggingPtr)
            {
                CCfDateTime64 dtEndTime = CCfDateTime64::Now();
                CCfDateTime64 dtStartTime;
                IContextFilterPtr pFilter;
                ICfUnknownPtr pUnkFilter;
                errorCode = pRuntime->GetObject(CCfString(L"ContextFilter"), &pUnkFilter);
                pFilter = pUnkFilter;
                if (nullptr != pFilter)
                {
                    {
                        CCfString strNameContextName = L"ContetxName_";
                        strNameContextName.Append(strContext_Unique_Num);
                        pFilter->SetContextName(strNameContextName);
                        pFilter->SetProviderType(static_cast<CFENUM>(HmiContextProviderType::UserDefined));
                        pFilter->SetOperator(CCfString(L"="));
                        pFilter->SetValue(CCfVariant(L"Orange Juice"));
                    }
                    CContextLoggingCB * pContextLoggingCB = new CContextLoggingCB();
                    pContextLoggingCB->AddRef();
                    errorCode = pContextLoggingPtr->ReadContexts(dtStartTime, dtEndTime,
                    pContextLoggingCB, pFilter, HmiContextLoggingSortingMode::Ascending);
                    if (CF_SUCCEEDED(errorCode))
                    {
                        if (CF_SUCCESS == pContextLoggingCB->WaitForcompletion(MaxContextWaitTime))
                        {
                            std::vector<ILoggedContextPtr> Context = pContextLoggingCB->GetContexts();

                            DisplayContext(Context);
                        }
                        else
                        {
                            std::cout << "ReadContexts Timeout " << std::endl;
                        }
                    }
                    else
                    {
                        std::cout << "ReadContexts failed " << std::endl;
                    }
                }
            }
        }
    }
}

```

Copying code

```

void DisplayContext(const std::vector<ILoggedContextPtr>& Context)
{
    CFRESULT errorCode = CF_ERROR;
    for (const auto& item : Context)
    {
        uint32_t pError;
        errorCode = item->GetErrorCode(&pError);
        if (CF_SUCCEEDED(errorCode))
            std::cout << "Error Code= " << pError << std::endl;
        if (CF_SUCCEEDED(pError))
        {
            CCfDateTime64 dtStartTime;
            errorCode = item->GetStartTime(&dtStartTime);
            CCfString strStartTime = dtStartTime.GetDateTimeString();
            if (CF_SUCCEEDED(errorCode))
                std::cout << "Start Time= " << strStartTime.ToUTF8().c_str() << std::endl;
            CCfDateTime64 dtEnd;
            errorCode = item->GetEndTime(&dtEnd);
            CCfString strEndTime = dtEnd.GetDateTimeString();
            if (CF_SUCCEEDED(errorCode))
                std::cout << "End Time =" << strEndTime.ToUTF8().c_str() << std::endl;
            uint32_t pQuality;
            errorCode = item->GetQuality(&pQuality);
            if (CF_SUCCEEDED(errorCode))
                std::cout << "Quality Code= " << pQuality << std::endl;
            CCfVariant vtValue;
            errorCode = item->GetValue(&vtValue);
            if (CF_SUCCEEDED(errorCode))
                PrintVariantType(vtValue);
        }
    }
}

```

Copying code

```

void SubscribeContextLogging(IRuntimePtr pRuntime)
{
    std::cout << std::endl << __FUNCTION__ << std::endl << std::endl;
    ICfUnknownPtr pUnk;
    CFRESULT errorCode = pRuntime->GetObject(CCFString(L"ContextLogging"), &pUnk);
    if (pUnk != nullptr && CF_SUCCEEDED(errorCode))
    {
        IContextLoggingPtr pContextLoggingPtr(pUnk);
        if (nullptr != pContextLoggingPtr)
        {
            CContextLoggingCB * pContextLoggingCB = new CContextLoggingCB();
            pContextLoggingCB->AddRef();
            CreateContextDefinition(pRuntime);
            CCFString strNameContextName = L"ContextName_";
            strNameContextName.Append(strContext_Unique_Num);
            CFENUM providerType = static_cast<CFENUM>(HmiContextProviderType::UserDefined);
            pContextLoggingPtr->Add(strNameContextName, providerType);
            errorCode = pContextLoggingPtr->Subscribe(pContextLoggingCB);
            if (CF_SUCCEEDED(errorCode))
            {
                StartContext(pRuntime);
                if (pContextLoggingCB->GetContexts().size() > 0 || pContextLoggingCB-
>WaitForCompletion(MaxContextWaitTime) == CF_SUCCESS);
                {
                    std::vector<ILoggedContextPtr> Context = pContextLoggingCB->GetContexts();
                    DisplayContext(Context);
                    pContextLoggingCB->clear();
                }
                StopContext(pRuntime);
                if (pContextLoggingCB->GetContexts().size() > 0 || pContextLoggingCB-
>WaitForCompletion(MaxContextWaitTime) == CF_SUCCESS);
                {
                    std::vector<ILoggedContextPtr> Context = pContextLoggingCB->GetContexts();
                    DisplayContext(Context);
                }
                errorCode = pContextLoggingPtr->CancelSubscribe();
                if (CF_FAILED(errorCode))
                    std::cout << "CancelSubscribe failed " << std::endl;
            }
            else
            {
                std::cout << "Subscribe failed " << std::endl;
            }
            pContextLoggingCB->Release();
        }
    }
}

void DisplayContext(const std::vector<ILoggedContextPtr>& Context)
{
    CFRESULT errorCode = CF_ERROR;
    for (const auto& item : Context)
    {
        uint32_t pError;
    }
}

```

Copying code

```

errorCode = item->GetErrorCode(&pError);
if (CF_SUCCEEDED(errorCode))
    std::cout << "Error Code= " << pError << std::endl;
if (CF_SUCCEEDED(pError))
{
    CCfDateTime64 dtStartTime;
    errorCode = item->GetStartTime(&dtStartTime);
    CCfString strStartTime = dtStartTime.GetDateTimeString();
    if (CF_SUCCEEDED(errorCode))
        std::cout << "Start Time= " << strStartTime.ToUTF8().c_str() << std::endl;
    CCfDateTime64 dtEnd;
    errorCode = item->GetEndTime(&dtEnd);
    CCfString strEndTime = dtEnd.GetDateTimeString();
    if (CF_SUCCEEDED(errorCode))
        std::cout << "End Time =" << strEndTime.ToUTF8().c_str() << std::endl;
    uint32_t pQuality;
    errorCode = item->GetQuality(&pQuality);
    if (CF_SUCCEEDED(errorCode))
        std::cout << "Quality Code= " << pQuality << std::endl;
    CCfVariant vtValue;
    errorCode = item->GetValue(&vtValue);
    if (CF_SUCCEEDED(errorCode))
        PrintVariantType(vtValue);
}
}
}

```

9.8.2 IContextLoggingCallBack**Description**

The C++ interface "IContextLoggingCallBack" defines methods for implementing asynchronous operations for monitoring "IContextDefinition" instances. The methods are used by the "IContextLogging" interface.

The interface implements the methods of the "ICfUnknown" interface.

All methods return CF_SUCCESS after successful execution. In the case of an error, the methods return the corresponding error code.

Members**"OnCreate" method**

Callback method is called when a ContextDefintion is created in the database by calling the "IContextLogging.CreateContextDefinitions" method.

9.8 Interfaces of the contexts

```
CFRESULT OnCreate(uint32_t globalError, CFSTR SystemName,  
IContextErrorEnumerator* errors, CFBOOL Completed)
```

- `globalError`:
[in]: Global ErrorCode when a global error occurs during the call. All other parameters are invalid in this case.
- `SystemName`:
[in]: Name of the system on which the ContextDefinitions have been created.
- `errors`:
[in]: Enumerator for the instance-specific errors that were created when the ContextDefinitions were created.
- `Completed`:
[in]: Status of the asynchronous transfer:
 - True: All ContextDefinitions have been notified.
 - False: Additional notifications are expected.

Members "OnRead"

Callback method is called:

- When a ContextDefintion is read from the database by calling the "IContextLogging.ReadContextDefinitions" method.
The method with the following signature is called:

```
CFRESULT OnRead(uint32_t globalError, CFSTR SystemName,
IContextDefinitionEnumerator* contextLoggingResult, CFBOOL
Completed)
```

 - globalError:
[in]: Global ErrorCode when a global error occurs during the call. All other parameters are invalid in this case.
 - SystemName:
[in]: Name of the system on which the ContextDefinitions have been created.
 - contextLoggingResult:
[in]: Enumerator for the read "IContextDefinition" instances
 - Completed:
[in]: Status of the asynchronous transfer:
True: All ContextDefinitions have been notified.
False: Additional notifications are expected.
- When a LoggedContext is read by calling the "IContextLogging.ReadContexts" method.
The method with the following signature is called:

```
CFRESULT OnRead(uint32_t globalError, CFSTR SystemName,
ILoggedContextEnumerator* loggedContexts, CFBOOL Completed)
```

 - globalError:
[in]: Global ErrorCode when a global error occurs during the call. All other parameters are invalid in this case.
 - SystemName:
[in]: Name of the system on which the "ILoggedContext" instances have been created.
 - contextLoggingResult:
[in]: Enumerator for the read "ILoggedContext" instances that were started or stopped.
 - Completed:
[in]: Status of the asynchronous transfer:
True: All ContextDefinitions have been notified.
False: Additional notifications are expected.
 - Completed:
[in]: Status of the asynchronous transfer:
True: All ContextDefinitions have been notified.
False: Additional notifications are expected.

"OnDataChanged" method

Callback method is called when a monitored "ILoggedContext" instance is started or stopped by calling "IContextLogging.StartContext" or "IContextLogging.StopContext".

```
CFRESULT OnDataChanged (ILoggedContextEnumerator* pEnumerator)
```

- `pEnumerator`:
[out]: Points to an "ILoggedContextEnumerator" object which contains an enumeration with "ILoggedContext" instances.

9.8.3 IContextDefinitionEnumerator

Description

The C++ interface "IContextDefinitionEnumerator" specifies methods for handling the enumeration of "IContextDefinition" instances. The enumeration is returned by the "OnRead" method of an "IContextLoggingCallBack" instance.

The interface inherits from the "ICfUnknown" interface.

All methods return `CF_SUCCESS` after successful execution. In the case of an error, the methods return the corresponding error code.

Members

"MoveNext" method

Go to the next element of the enumeration.

```
CFRESULT MoveNext ()
```

"Current" method

Output the current element of the enumeration.

```
CFRESULT Current (IContextDefinition **pItem)
```

- `ppItem`
[out]: The current "IContextDefinition" instance

"Reset" method

Reset the current position in the enumeration. The "MoveNext" method moves afterwards to the first element.

```
CFRESULT Reset ()
```

"Count" method

Output the size of the enumeration or the number of its elements.

```
CFRESULT Count (uint32_t* pCount)
```

- `pCount`
[out]: Number of "IContextDefinition" instances

9.8.4 IContextErrorEnumerator

Description

The C++ interface "IContextErrorEnumerator" specifies methods for handling the enumeration of "IContextError" instances. The enumeration is returned by the "OnCreate" method of an "IContextLoggingCallBack" instance.

The interface inherits from the "ICfUnknown" interface.

All methods return CF_SUCCESS after successful execution. In the case of an error, the methods return the corresponding error code.

Members

"MoveNext" method

Go to the next element of the enumeration.

```
CFRESULT MoveNext()
```

"Current" method

Output the current element of the enumeration.

```
CFRESULT Current(IContextError **pItem)
```

- pItem
[out]: The current "IContextError" instance

"Reset" method

Reset the current position in the enumeration. The "MoveNext" method moves afterwards to the first element.

```
CFRESULT Reset()
```

"Count" method

Output the size of the enumeration or the number of its elements.

```
CFRESULT Count(uint32_t* pCount)
```

- pCount
[out]: Number of "IContextError" instances

9.8.5 ILoggedContextEnumerator

Description

The C++ interface "ILoggedContextEnumerator" specifies methods for handling the enumeration of "ILoggedContext" instances. The enumeration is returned by the "OnRead" method of an "IContextLoggingCallBack" instance.

The interface inherits from the "ICfUnknown" interface.

All methods return CF_SUCCESS after successful execution. In the case of an error, the methods return the corresponding error code.

Members

"MoveNext" method

Go to the next element of the enumeration.

```
CFRESULT MoveNext ()
```

"Current" method

Output the current element of the enumeration.

```
CFRESULT Current (ILoggedContext **pItem)
```

- pItem
[out]: The current "ILoggedContext" instance

"Reset" method

Reset the current position in the enumeration. The "MoveNext" method moves afterwards to the first element.

```
CFRESULT Reset ()
```

"Count" method

Output the size of the enumeration or the number of its elements.

```
CRFRESULT Count (uint32_t* pCount)
```

- pCount
[out]: Number of "ILoggedContext" instances

9.8.6 IContextDefinition

Description

The C++ interface "IContextDefinition" specifies properties for defining "IContextDefinition" instances. "ILoggingContext" instances can be created using the StartContext() and StopContext() methods based on an "IContextDefiniton" instance.

The interface implements the methods of the "ICfUnknown" interface.

All methods return CF_SUCCESS after successful execution. In the case of an error, the methods return the corresponding error code.

Members

"GetPlantViewPath" method

Sets the path to the plant object of the "IContextDefinition" instance.

```
CRFESULT GetPlantViewPath(CFSTR* p_strPlantViewPath)
```

- `p_strPlantViewPath`:
[out]: The path to the plant object
Example: ".hierarchy::Plant/Station"

"SetPlantViewPath" method

Sets the path to the plant object of the "IContextDefinition" instance.

```
CRFESULT SetPlantViewPath(CFSTR p_strPlantViewPath)
```

- `p_strPlantViewPath`:
[in]: The path to the plant object
Example: ".hierarchy::Plant/Station"

"GetProviderType" method

The source that creates the instance.

Is used together with "Name" to uniquely identify a "IContextDefinition" instance.

```
CRFESULT GetProviderType(CFENUM* pContextType)
```

- `pContextType`:
[out]: The enumeration "HmiContextProviderType" can contain the following values:
 - NoContext = 0
 - Calendar = 1
For "IContextDefinition" instances generated by the PI Option Calendar.
 - PerformanceInsightMachineState = 2
For "IContextDefinition" instances generated by the PI Option Performance Insight.
 - LineCoordinator = 3
For "IContextDefinition" instances generated by the PI Option Line Coordinator.
 - UserDefined = 5
It is a user-defined "IContextDefinition" instance, created by ODK, for example.

"GetName" method

Supplies the name of the "IContextDefinition" instance.

```
CRFESULT GetName(CFSTR* pName)
```

- `pName`:
[out]: The name

"SetName" method

Sets the name of the "IContextDefinition" instance.

Sets the name of the "IContextDefinition" instance.

```
CRFESULT SetName(IN CFSTR name)
```

- name:
[in]: The name

"GetDisplayNames" method

Supplies the display name of the "IContextDefinition" instance

```
CRFESULT GetDisplayNames(ICfMapIDToVariant** ppDisplayName)
```

- ppDisplayName:
[out]: The display name

"SetDisplayNames" method

Sets the display name of the "IContextDefinition" instance.

```
CRFESULT SetDisplayNames(ICfMapIDToVariant* pDisplayName)
```

- pDisplayName:
[in]: The display name

"GetDataType" method

Supplies the data type of the "IContextDefinition" instance.

```
CRFESULT GetDataType (HmiContextDataType* pdatatype)
```

- `pdatatype`:
[out]: The data type
The enumeration "HmiContextDataType" can contain the following values:
 - Bool = 0x01
 - SInt = 0x02
 - Int = 0x03
 - DInt = 0x04
 - LInt = 0x05
 - USInt = 0x06
 - UInt = 0x07
 - UDIInt = 0x08
 - ULInt = 0x09
 - Real = 0x0A
 - LReal = 0x0B
 - LTime = 0x0C
 - DateTime = 0x0D
 - Byte = 0x11
 - Word = 0x12
 - DWord = 0x13
 - LWord = 0x14
 - String = 0x32

"SetDataType" method

Sets the data type of the "IContextDefinition" instance.

```
CRFESULT SetDataType(HmiContextDataType datatype)
```

- datatype:
[in]: The data type
The enumeration "HmiContextDataType" can contain the following values:
 - Bool = 0x01
 - SInt = 0x02
 - Int = 0x03
 - DInt = 0x04
 - LInt = 0x05
 - USInt = 0x06
 - UInt = 0x07
 - UDInt = 0x08
 - ULLInt = 0x09
 - Real = 0x0A
 - LReal = 0x0B
 - LTime = 0x0C
 - DateTime = 0x0D
 - Byte = 0x11
 - Word = 0x12
 - DWord = 0x13
 - LWord = 0x14
 - String = 0x32

"GetErrorCode" method

Supplies the error code of the "IContextDefinition" instance

```
CRFESULT GetErrorCode(uint32_t* error)
```

- error:
[out]:

9.8.7 ILoggedContext

Description

The C++ interface "ILoggedContext" defines properties of context log entries of an "IContextDefinition" instance.

The context log entries are started and stopped using methods of the "IContextLogging" interface.

The interface implements the methods of the "ICfUnknown" interface.

All methods return CF_SUCCESS after successful execution. In the case of an error, the methods return the corresponding error code.

Members

"GetStartTime" method

Supplies the start time of the "IContextLogging" instance.

```
CRFESULT GetStartTime(CFDATE64* value)
```

- value:
[out]: The start time

"GetEndTime" method

Supplies the end time of the "IContextLogging" instance.

```
CRFESULT GetEndTime(CFDATE64* value)
```

- value:
[out]: The end time

"GetErrorCode" method

Supplies the error code of the "IContextLogging" instance.

```
CRFESULT GetErrorCode(uint32_t* error)
```

- error:
[out]: The error code

"GetValue" method

Supplies the name that the "IContextDefinition" instance of the "IContextLogging" instance has in the user interface.

Must correspond to the data type from "DataType" of the "IContextDefinition" instance.

```
CRFESULT GetValue(CFVARIANT * value)
```

- value:
[out]: The name of the "IContextDefinition" instance in the user interface

"GetQuality" method

Supplies the QualityCode of the context value.

```
CRFESULT GetQuality(uint32_t* quality)
```

- quality:
[out]: The QualityCode

9.8.8 IContextError

Description

The C++ interface "IContextError" specifies methods for accessing error results that occur when creating ContextDefinitions in the database.

The interface implements the methods of the "ICfUnknown" interface.

All methods return CF_SUCCESS after successful execution. In the case of an error, the methods return the corresponding error code.

Members

"GetContextName" method

Supplies the name of the "IContextDefintion" instance.

```
CRFESULT GetContextName (CFSTR* p_strName)
```

- p_strName:
[out]: The name

"GetContextErrorCode" method

Supplies the error code.

```
CRFESULT GetErrorCode) (OUT uint32_t* p_ErrorCode)
```

- p_ErrorCode:
[out]: The error code

9.8.9 IContextFilter

Description

The C++ interface "ILoggedContext" specifies methods for accessing properties for filtering for "ILoggedContext" instances.

The interface inherits from the "ICfUnknown" interface.

All methods return CF_SUCCESS after successful execution. In the case of an error, the methods return the corresponding error code.

Members

"GetName" method

Supplies the name of the "IContextDefinition" instance according to whose "ILoggedContext" instances the filtering is performed.

```
CFRESULT GetContextName (CFSTR* p_strName)
```

p_strName:

[out]: The name of the "IContextDefinition" instance

"SetName" method

Sets the name of the "IContextDefinition" instance according to whose "ILoggedContext" instances the filtering is performed.

```
CFRESULT SetContextName( CFSTR p_strName)
```

p_strName:

[in]: The name of the "IContextDefinition" instance

"GetProviderType" method

The HmiContextProviderType that the "IContextDefinition" instance of the context log entry must have.

```
CFRESULT GetProviderType(CFENUM* p_ProviderType)
```

- p_ProviderType:

[out]: The provider type.

The enumeration "HmiContextProviderType" can contain the following values:

- NoContext = 0
- Calendar = 1
For "IContextDefinition" instances generated by the PI Option Calendar.
- PerformanceInsightMachineState = 2
For "IContextDefinition" instances generated by the PI Option Performance Insight.
- LineCoordinator = 3
For "IContextDefinition" instances generated by the PI Option Line Coordinator.
- UserDefined = 5
It is a user-defined "IContextDefinition" instance, created by ODK, for example.

"SetProviderType" method

Sets the HmiContextProviderType of an "IContextDefinition" instance for whose "ILoggedContext" instances the filtering is performed.

```
CFRESULT SetProviderType(CFENUM p_ProviderType)
```

- p_ProviderType:

[in]: The provider type.

The enumeration "HmiContextProviderType" can contain the following values: See "GetProviderType".

"GetOperator" method

Supplies the filter operator.

```
CFRESULT GetOperator(CFSTR* p_Operator)
```

- p_Operator:

[out]: The operator.

The operator is applied to the value.

"SetOperator" method

Sets the filter operator.

```
CFRESULT SetOperator(CFSTR p_Operator)
```

- **p_Operator:**
[in]: The operator
The operator is applied to the value. The following operators are allowed:
For values with data type Int and Real:

- =
- !=
- <
- >
- <=
- >=

For values with data type String:

- LIKE
- =

Strings must always be fully specified.

"GetValue" method

Supplies the "Value" of the "ILoggedContext" instance

```
CFRESULT GetValue(CFVARIANT* p_vtValue)
```

- **p_vtValue:**
[out]: The value

"SetValue" method

Sets the "Value" of the "ILoggedContext" instance

```
CFRESULT SetValue(CFVARIANT p_vtValue)
```

- **p_vtValue:**
[in]: The value