


Description of Function

DoF CB Open Interface

SUMMARY

This document describes the “Control Builder Open Interface”. The idea is to open up the Control Builder Professional and Compact Control Builder products to other applications. External applications (EXE’s) can use the methods of the open interface in order to create, change and delete CB-objects such as project, libraries, applications, controllers, data types, function block types, control module types and so on. These methods are primarily used by the Source Code and by the integration with PPA.

The official name of the interface is “Control Builder Open Interface” or “CB Open Interface”. This name is sometimes abbreviated, in this document, to just “Open Interface”.

 ABB AB	Doc. no. 3BSE033313	Lang. en	Rev. ind. Q	Page 1
--	------------------------	-------------	----------------	-----------

CONTENTS

1	INTRODUCTION	4
2	REFERENCES	4
2.1	Related Documents.....	4
3	TERMINOLOGY	5
4	GENERAL DESCRIPTION	6
4.1	Short description of CB Open Interface	6
5	SPECIFIC FUNCTIONS.....	11
5.1	Interface methods.....	12
5.1.1	Object paths	12
5.1.2	Optional parameters	12
5.1.3	Error codes etc.	12
5.1.4	Methods for Project.....	17
5.1.5	Methods for Library.....	24
5.1.6	Methods for Hardware Library	31
5.1.7	Methods for Application	39
5.1.8	Methods for Execution Order	49
5.1.9	Methods for Task Connection	52
5.1.10	Methods for Controller	54
5.1.11	Methods for DataType	66
5.1.12	Methods for FunctionBlockType.....	72
5.1.13	Methods for FunctionBlock	78
5.1.14	Methods for ControlModuleType.....	84
5.1.15	Methods for ControlModule.....	90
5.1.16	Methods for SingleControlModules	98
5.1.17	Using graphical connections in Control Module Types.....	105
5.1.18	Methods for DiagramTypes.....	118
5.1.19	Methods for DiagramInstances	125
5.1.20	Methods for Program	131
5.1.21	Methods for Diagram	137
5.1.22	Methods for HardwareType	145
5.1.23	Methods for Access Variables	150
5.1.24	Methods for Task.....	152
5.1.25	Methods for ConnectedApplications	157
5.1.26	Methods for HWUnits.....	159
5.1.27	Methods for Online	169
5.1.28	Methods for Folder.....	173
5.1.29	Methods for retrieving information from the current project	179
5.1.30	Methods for reservation of File Organization Units.....	190
5.1.31	Methods for settings	192
5.1.32	Methods for Parameters	195
5.1.33	Methods for Variables.....	200
5.1.34	Methods for Signals.....	205
5.1.35	Methods for CodeBlocks.....	210
5.1.36	Methods for CMConnections	215
5.1.37	Methods for FDConnections	217
5.1.38	Methods for Connected Libraries	219
5.1.39	Methods for Connected Hardware Libraries.....	224
5.1.40	Methods for Version and State	229
5.1.41	Methods for Refresh	237
5.1.42	Methods for writing in the CBM Session log.....	240
5.1.43	XML Examples for code blocks written in various languages	242
5.1.44	Some additional remarks	252
5.2	The XML Schema.....	259

5.2.1	The HTML project "CBOpenIFSchema3_0.html"	260
5.3	A Visual Basic 6.0 application sample using the CB Open Interface.....	261
5.4	A Visual C++ 6.0 application sample using the CB Open Interface.....	268
5.4.1	Some advice for Visual C++ programmers using the CB Open Interface	268
5.4.2	A Visual C++ 6.0 application sample	269
5.5	The CB Object model	273
5.6	The Control IT Installation media.....	276

1 INTRODUCTION

The purpose of the “Control Builder Open Interface” is to open up the Control Builder (CB) product to other applications. External applications (EXE's) can use the methods of the open interface in order to create, change and delete CB-objects such as project, libraries, applications, controllers, data types, function block types, control module types and so on.

Possible clients to the open interface are among others:

- Aspect systems. The open interface is the primary way to provide integration with the Operate IT products
- Some external customers.
- External editors.
- Source Code Storage

The interface will enable a new Control Builder architecture that makes engineering tools more independent. The interface is also a step in a direction where the Atlas code base on the Control Builder side can be split up into several components.

2 REFERENCES

2.1 Related Documents

Number	Document Identity	Document Title
[3]	ISBN 1-861005-05-9	Professional XML 2 nd Edition,
[5]	ISBN 1-861001-20-7	Programming Components with Microsoft Visual Basic 6.0
[6]		MSDN magazine, September 2000 and October 2000
[7]	ISBN 1-861004-99-0	Professional C#
[8]	ISBN 1-861001-20-7	Beginning ATL3 COM Programming

3 TERMINOLOGY

Terms as XML, XML Schemas, COM, COM Interface, SOAP and WebServices are described in reference [3], [5], [6] and [7].

Terms

Definitions

GUID

Short for Globally Unique Identifier. A 128 bit number that is globally unique. It is used in COM for identification purposes.

UNC

Stands for *Universal Naming Convention* and is a way to specify a directory path without using a drive letter. It consists of a server and a share specification, followed by any subdirectory path. Example:

\\FileServer1\rootshare\subdirectory

More details on this subject may be found in the Windows 2000 help system

SCM

Single Control Module

GENERAL DESCRIPTION

4.1 Short description of CB Open Interface

CB Open Interface is a COM-interface. The interface contains a lot of methods. The methods expose almost all of the Control Builder's functionality. See Figure 1.

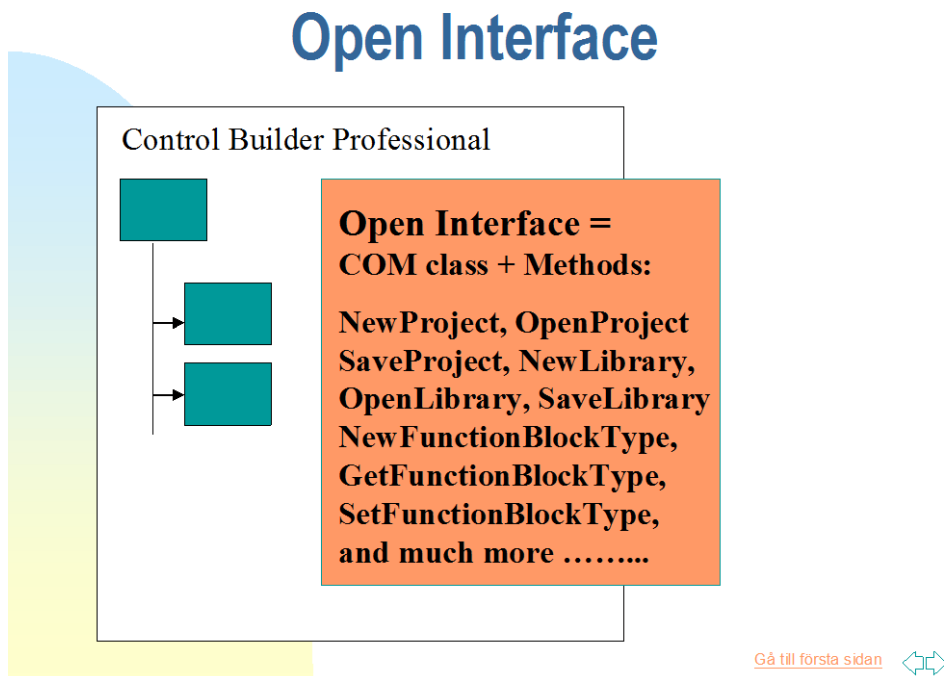


Figure 1

The interface contains a set of methods so that other applications (EXEs) will be able to create, change and delete the following CB-objects:

1. Project
2. Libraries
3. Hardware libraries
4. Applications
5. Controllers
6. Data types
7. Function block types including all sub objects
8. Control module types including all sub objects
9. Diagram types including all sub objects
10. Diagrams
11. Programs including all sub objects
12. Control modules including all sub objects
13. Hardware types
14. Folders
15. Hardware units including all sub objects
16. Tasks
17. Task connections
18. Hardware connections
19. Access variables
20. Project constants

XML in Open Interface

- Some methods use XML in order to describe the content (variables, parameters, codeblocks, ..) for a type, an instance or a singleton. Example:

```
◆ XMLString = GetFunctionBlockType("MyLib.MyFBType")
◆ MsgBucket = SetFunctionBlockType("MyLib.MyFBType",
                                   XMLString)
◆ MsgBucket = NewFunctionBlockType("ANewFB", "MyApp",
                                   XMLString)
```

[Gå till första sidan](#) 

Figure 2

Some interface methods return strings containing an XML-description of (for instance) a type, an instance, a single-program or a single module. Other methods have a string parameter containing an XML-description. Such strings are called "XMLString" in Figure 2.

The content of a type, an instance and so on, can easily be described in XML. Figure 3 and Figure 4 show a Program in the Control Builder described in XML. The root element <Program> has a name attribute containing the value "NewProg", an empty task-connection attribute and a reference to an XML Schema followed by the sub elements: Variables, FunctionBlocks and CodeBlocks. The <Variables> element contains a sequence of <Variable> sub-elements and so on.

Example. A CB-program described in XML.

```
<?xml version="1.0" encoding="UTF-16"?>
<Program Name="NewProg" TaskConnection=""
        xmlns="urn:CBOpenIFSchema2_0">
  <Variables>
    <Variable Name="ProgVar" Type="dint" Attribute="retain"
              InitialValue="">
    </Variable>
    <Variable Name="Kalle" Type="bool" Attribute="retain"
              InitialValue="true">
    </Variable>
  </Variables>
  <FunctionBlocks/>
  <CodeBlocks>
  <!--! See next page -->
```

[Gå till första sidan](#) 

Figure 3

Example. Continuation

```
<!--! Continuation from previous page -->
<CodeBlocks>
  <STCodeBlock Name="Code">
    <ST_Code>
      if ProgVar > 0 and Kalle then
        ProgVar := ProgVar+1;
      end_if;
    </ST_Code>
  </STCodeBlock>
</CodeBlocks>
</Program>
```

[Gå till första sidan](#) 

Figure 4

The internal code in the Control Builder uses Microsoft's XML parser called MSXML. This parser is an XML DOM parser. Input to the parser is an XML document contained in a string. Output from the parser is a tree of XMLDOMNodes i.e. a tree of COM-objects. In addition, this parser is able to validate the XML document (string) according to its XML Schema. The XML Schema contains the grammatical rules for XML elements and attributes defined for the Control Builder. Knowledge about the correctness of the XML document makes the code for traversing the tree much easier to write. See Figure 5.

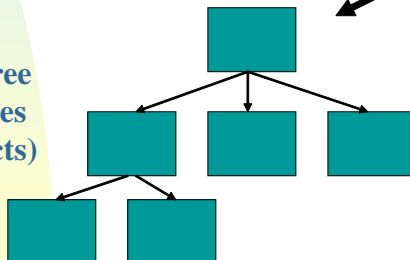
The MSXML Parser

Input: an XML Document

```
<?xml version="1.0" encoding="UTF-16"?>
<Program Name="NewProg" TaskConnection=""
  xmlns="urn:CBOpenIFSchema2_0">
  <Variables>
    <Variable Name="ProgVar" Type="dint"
      Attribute="retain"
      InitialValue="">
    </Variable>
    <Variable Name="Kalle" Type="bool"
      Attribute="retain"
      InitialValue="true">
    </Variable>
  </Variables>
</Program>
```

**MSXML
parser**

Output: a Tree of XML Nodes (COM Objects)



[Gå till första sidan](#) 

Figure 5

We have also developed a Control Builder “OLE automation compatible” object model, named [“CB Open Interface Helper”](#), as a **COM-DLL**. The purpose of the object model is to take care of all XML stuff so that clients (other applications, EXEs) don’t have to have any knowledge about XML. See Figure 6.

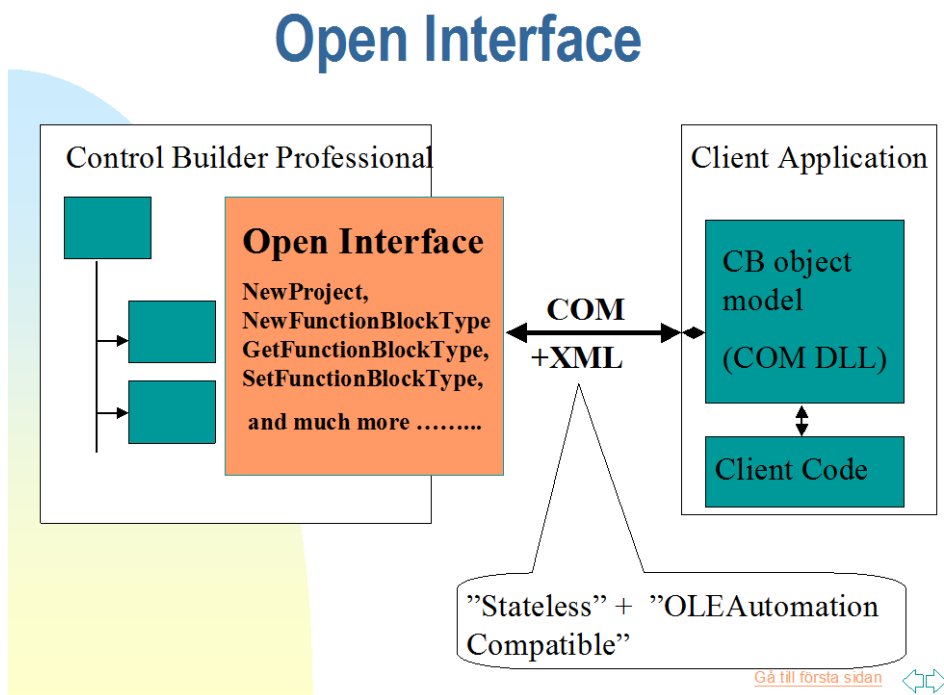


Figure 6

Figure 6 shows the “Control Builder object model” wrapped in a COM-DLL. The object model is easier to use compared with an XML DOM tree. Figure 7 shows some C# code using the object model.

Create a new FunctionBlockType using the object model

```
// Create an object of the type "FunctionBlockType" in the client's
// process memory
FunctionBlockType fbType =
    ObjectFactory.NewFunctionBlockType("MyFBType", "Desc of MyFBType");
// Add some Variable objects
fbType.Variables.Add1("MyVariable", "bool");
fbType.Variables.Add2("X", "dint", "retain", "8", "", "", "Desc");
// Add a ST CodeBlock
string stCode = "if MyVariable then\n" +
    "    X := X + 1;\n" +
    "end_if;";
fbType.CodeBlocks.AddSTCodeBlock2("MySTCodeBlock", ref stCode);
// Finally, serialize the object model into an XML String and
// call the OpenIF method "NewFunctionBlockType" in order to create
// the type in the Control Builder EXE.
string bucket = cb.NewFunctionBlockType(fbType.Name, "MyLib",
    fbType.Serialize());
```

[Gå till första sidan](#)



Figure 7

Figure 8 shows some characteristics for the “Open Interface”.

Open Interface characteristics

- Open Interface is "Stateless" =>
 - ◆ Remote (**non windows**) clients can use SOAP (WEBServices)
 - ◆ Remote **Windows** clients can use DCOM or SOAP (WEBServices)
- Open Interface is "OLEAutomation Compatible" =>
 - ◆ VC++ version 6 clients can use the "#import" directive in order to create wrapper classes
 - ◆ .NET clients can create managed wrapper classes
 - ◆ Script clients, as old ASP-pages, can use the interface because it is "dual" as well.
- Windows clients can use the CB object model (COM DLL) to simplify the client code

[Gå till första sidan](#) 

Figure 8

5 SPECIFIC FUNCTIONS

The Open Interface consists of two different parts:

1. A COM Interface with a lot of methods. Some interface methods return strings, or have strings as parameters, where the strings contain an XML-description of (for instance) a type, an instance, and so on. This part is described in the chapter [Interface Methods](#).
2. An “OLE automation compatible” object model as a **COM-DLL**. The purpose of the object model is to take care of all XML stuff so that clients (other applications, EXEs) don’t have to have any knowledge about XML. The object model is easier to use compared with an XML DOM tree. This part is described in the chapter [CB Object model](#). Also see ref [9]

A client on non-windows platforms, for instance a Java application running on Linux, has to use the interface methods directly. A non-windows client also has to investigate the SOAP-support for the particular target and to choose an appropriate XML parser.

5.1 Interface methods

5.1.1 Object paths

When accessing objects in the CB using OpenIF methods, the path or just the name to the object is used to identify which object to operate on.

If the object is a library, application or controller, it is enough with just the name of the object.

Example 1: To access Library1, the path to use is just "Library1".

To access objects that lies beneath another object, a peroid (".") is used to separate each level.

Example 2: To access the DataType DT1 in Library1, the path is "Library1.DT1"

Example 3: To access the ControlModule CM1 in the Control Module Type CMT1 in Application1, The path is: Application1.CMT1.CM1

Handling paths with versions

It is possible to include the version of a library, application and controller in the object paths. If no version is specified in the path, the library, application or controller with the highest version number will be pointed out.

The version number is included in the name of the library, application or controller name part of the path. It is appended to the name with a space separating the name from the version. The version follows the following syntax: A.B-C where A is the Major version number, B is the Minor version number and C is the Revision number. A is a number between 1 and 32767 and B and C are numbers between 0 and 255.

Example 1: To access Library1 version 1.0-0, the path to use is just "Library1 1.0-0".

Example 2: If there are two versions of Library1 in the CB, e.g. version 1.0-0 and 2.0-0 and the path given to the OpenIF method is just "Library1", Library1 with the version 2.0-0 will be pointed out.

Example 3: To access the DataType DT1 in Library1 version 2.0-3, the path is "Library1 2.0-3.DT1"

Example 4: To access the ControlModule CM1 in the Control Module Type CMT1 in Application1 version 3.4-5, the path is: Application1 3.4-5.CMT1.CM1

5.1.2 Optional parameters

There are several methods in the CB Open Interface that have optional parameters. The optional parameters can NOT be left out, but if there is no valid value to give the parameter, a blank string functions as an empty value for all optional parameters of the type String.

5.1.3 Error codes etc.

The Open Interface methods can report errors in different ways:

1. As HRESULTs combined with "rich error information". The "rich error information" provides a textual description of the error.
2. As a "Message bucket"

5.1.3.1 HRESULTS and “rich error information”

All methods return HRESULT. The HRESULT tells whether the method call succeeded or not. In addition a textual description, sometimes called “rich error information”, is returned. Table 1 shows the HRESULT codes used by Open Interface.

Symbolic constant	Value	Description
OI_E_NOTSUPPORTED	0x80040BC2	CB Open Interface is not supported in other products than Control Builder Professional
OI_E_MODE	0x80040BC3	Can not be called when Control Builder is in Online or TestMode mode
OI_E_XML	0x80040BC4	Error when parsing XML document
OI_E_NAME	0x80040BC5	An invalid name was used
OI_E_PATH	0x80040BC6	An invalid path was used
OI_E_CREATEDIRECTORY	0x80040BC7	Failed to create file directory
OI_E_ALREADYEXIST	0x80040BC8	The item already exists
OI_E_MAXNOEXCEEDED	0x80040BC9	Maximum number of items exceeded
OI_E_OPEN	0x80040BCA	The item could not be opened
OI_E_SAVE	0x80040BCB	The item could not be saved
OI_E_DOESNTEXIST	0x80040BCC	The item doesn't exist
OI_E_DELETE	0x80040BCD	The item could not be deleted
OI_E_TYPE	0x80040BCE	Type is unknown or doesn't exist
OI_E_STATE	0x80040BCF	The state is invalid
OI_E_VERSION	0x80040BD0	The version is missing or invalid
OI_E_MODALDIALOG	0x80040BD1	The operation could not be performed due to an open modal dialog
OI_E_NAME_CONFLICT	0x80040BD2	An object with the same name but a different GUID already exists
OI_E_GUID	0x80040BD3	Incorrect GUID
OI_E_DECRYPTION	0x80040BD4	Source code decryption has failed
OI_E_CHECKSUM	0x80040BD5	Source code check sum is incorrect
OI_E_CHECKSUMCODEPAGE	0x80040BD6	Source code check sum is incorrect due to differing language setting in the system the saved the source code and the one reading it
OI_E_INUSE	0x80040BD7	The item is in use
OI_E_RESERVATION	0x80040BD8	The necessary entities are not reserved for this operation

Table 1. HRESULT codes.

In addition, the following standard constants are used:

Symbolic constant	Value	Description
S_OK	Standard	Success
E_FAIL	Standard	Unspecified failure
E_ACCESSDENIED	Standard	General access denied error
E_POINTER	Standard	Invalid pointer passed for output parameter
E_UNEXPECTED	Standard	Unexpected error in you application
E_INVALIDARG	Standard	One or more arguments are invalid
E_NOTIMPL	Standard	This method is not implemented

Table 2. Additional HRESULT codes.

The Open Interface methods support “rich error information” in addition to HRESULT error codes. The rich error information provides a detailed textual description of an error. The mapping between HRESULT error codes and “rich error information” texts is not a one to one relation. The error text often describes the error in detail. A Visual Basic application can retrieve the textual description of an error through the built in “Err” object’s description property (“Err.Description”).

Note! **The COM runtime can issue other HRESULT codes than the ones presented above.** A C-client should use “if (SUCCEEDED(hr))” instead of “if (hr == S_OK)” when testing if a method call succeeded or failed.

5.1.3.2 Message buckets

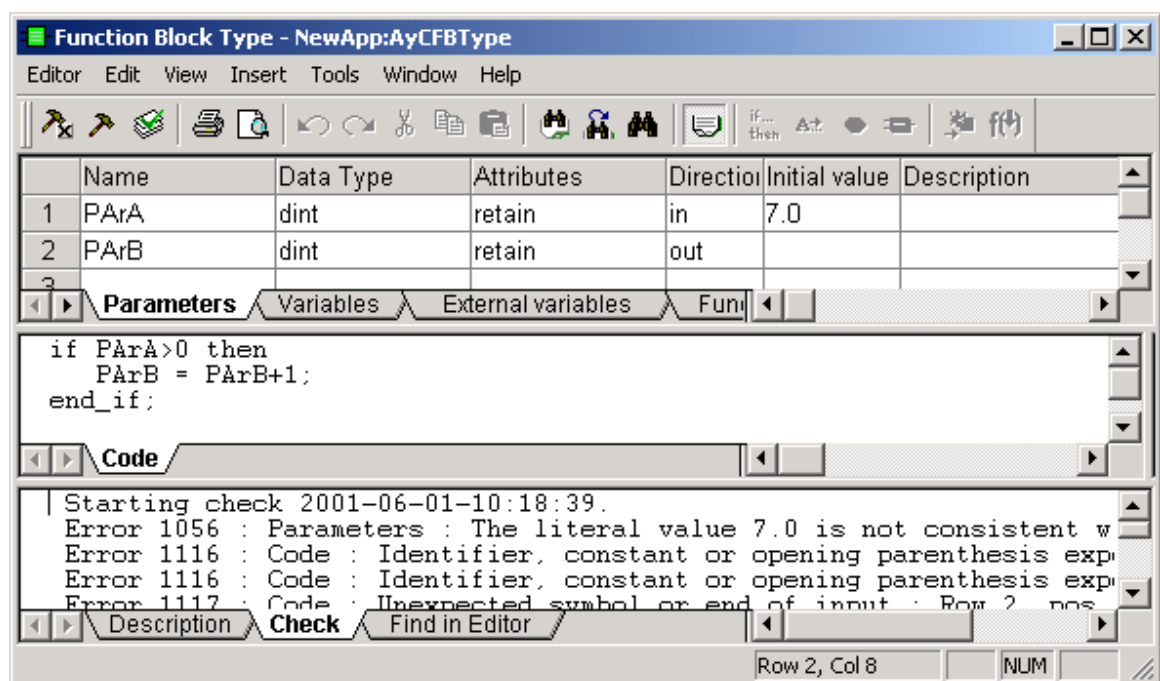


Figure 9

What is a message bucket? Figure 9 tries to explain the concept from the Control Builder’s point of view. Consider the following case:

1. A user edits a function block type using a POU editor
2. Some of the changes are erroneous. For instance (see Figure 9), an integer parameter named "PArA" has an initial value "7.0" that isn't consistent with the type. Another example is the structure text code block which contains an illegal "assignment" operator "=" instead of "[:="
3. The user tries to save the changes.

The internal structure of the Control Builder (the internal object model) is able to store erroneous "stuff". The result of the user's save attempt will be:

1. The POU Editor "saves" the changes. That is, the editor stores the content of the editor to storage.
2. The POU Editor displays some error texts into the editor's third pane. These error texts is contained in a so called "**Message bucket**"

A "Message bucket" is a container for information, warning and error texts. The Open Interface equivalence of saving changes (in a function block type etc) from a POU Editor is the **SetFunctionBlockType** or **NewFunctionBlockType** methods.

Conclusions:

1. Some New-methods and Set-methods returns a message bucket, **but only if the returned HRESULT is S_OK**. The method call is regarded as successful but some minor warnings, errors or information text are returned in the "bucket"
2. A "Message bucket" is an XML coded container for information, warning and error texts.
3. The "Message bucket" functionality is needed in order to let remote editors save incorrect information to storage of the Control Builder and in order to let the editors display this information.

5.1.3.2.1 Example of a returned message bucket coded in XML:

```
<?xml version="1.0" encoding="UTF-16"?>
<MessageBucket xmlns="CBOpenIFSschema3_0" NoOfErrors="5" NoOfWarnings="0">
  <Error ErrorNo="1056" Message="The literal value 7.0 is not consistent with data
type dint.">
    <PosInfo FOUName="NewApp" POUName="AyCFBType" TabName="Parameters" Row="1"
Col="5" StartPos="-1" EndPos="-1" MessageType="FunctionBlock"/>
    <ExtraInfo/>
  </Error>
  <Error ErrorNo="1116" Message="Identifier, constant or opening parenthesis
expected">
    <PosInfo FOUName="NewApp" POUName="AyCFBType" TabName="Code" Row="2" Col="1"
StartPos="9" EndPos="9" MessageType="FunctionBlock"/>
    <ExtraInfo/>
  </Error>
  <Error ErrorNo="1116" Message="Identifier, constant or opening parenthesis
expected">
    <PosInfo FOUName="NewApp" POUName="AyCFBType" TabName="Code" Row="2" Col="1"
StartPos="14" EndPos="14" MessageType="FunctionBlock"/>
    <ExtraInfo/>
  </Error>
  <Error ErrorNo="1117" Message="Unexpected symbol or end of input">
    <PosInfo FOUName="NewApp" POUName="AyCFBType" TabName="Code" Row="2" Col="1"
StartPos="16" EndPos="16" MessageType="FunctionBlock"/>
    <ExtraInfo/>
  </Error>
  <Error ErrorNo="1117" Message="Unexpected symbol or end of input">
    <PosInfo FOUName="NewApp" POUName="AyCFBType" TabName="Code" Row="3" Col="1"
StartPos="1" EndPos="6" MessageType="FunctionBlock"/>
    <ExtraInfo/>
  </Error>
</MessageBucket>
```

For further information see the [XML Schema](#)

5.1.4 Methods for Project

5.1.4.1 NewProject

```
HRESULT NewProject([in] BSTR projectName,  
                  [in] BSTR directoryPath,  
                  [in] BSTR guid,  
                  [in] BSTR templateName);
```

```
Sub NewProject(projectName As String, _  
              directoryPath As String, _  
              guid As String, _  
              templateName As String)
```

Description

Creates a new project according to the template given. If no template is entered, an empty project containing only the standard library System will be created. An already loaded project is closed.

Parameters

Name	Type	Description
projectName	String	Name of the new project
directoryPath	String	An optional parameter specifying the file location where the project will be saved. If this parameter is an empty string the project will be created in the default directory, i.e. the directory that is proposed when manually creating a project. It is possible to use an UNC path. Note however the restriction described in chapter “Some additional remarks, File Organization Units ”.
guid	String	An optional parameter specifying a Globally Unique Identifier. If this parameter is an empty string the Control Builder will assign a GUID to the object. If the parameter is present the syntax must be as follows: "12345678-1234-1234-1234-123456789ABC"
templateName	String	An optional parameter specifying the template to use for the new project. If an empty string is entered, an empty project will be created.

5.1.4.2 NewProjectInEnvironment

```
HRESULT NewProjectInEnvironment([in] BSTR projectName,  
                                [in] BSTR guid,  
                                [in] BSTR templateName,  
                                [in] BSTR environmentGuidOrName);
```

```
Sub NewProjectInEnvironment(projectName As String, _  
                            guid As String, _  
                            templateName As String, _  
                            environmentGuidOrName As String)
```

Description

Creates a new project according to the template given in a specific environment. If no template is entered, an empty project containing only the standard library System will be created. An already loaded project is closed.

This method is only functional in Control Builder Professional.

Parameters

Name	Type	Description
projectName	String	Name of the new project
guid	String	An optional parameter specifying a Globally Unique Identifier. If this parameter is an empty string the Control Builder will assign a GUID to the object. If the parameter is present the syntax must be as follows: "12345678-1234-1234-1234-123456789ABC"
templateName	String	An optional parameter specifying the template to use for the new project. If an empty string is entered, an empty project will be created.
environmentGuidOrName	String	Specifies the name or the GUID of the environment in which to create the project.

5.1.4.3 OpenProject

```
HRESULT OpenProject([in] BSTR filePath);
```

```
Sub OpenProject(filePath As String)
```

Description

Opens an existing project. An already loaded project is closed.

Parameters

Name	Type	Description
filePath	String	<p>CB Professional:</p> <p>The GUID of the project to open.</p> <p>CB Standad:</p> <p>File specification for an existing project. The file specification can be a full path. Example: “C:\MyDir\MyProject”</p> <p>If the directory path is empty then the logical “Projects” folder is assumed. Thus, if the file path is “aProject” the project “Projects\aProject\aProject” is opened.</p> <p>The environment variable “Examples” can be used in order to specify the examples folder where some template projects reside. Example: “Examples\AlarmSimple\AlarmSimple”. The syntax “%Examples%\ AlarmSimple\AlarmSimple” is also allowed.</p>

5.1.4.4 OpenProjectInEnvironment

```
HRESULT OpenProjectInEnvironment([in] BSTR projectGuidOrName,  
                                [in] BSTR environmentGuidOrName);
```

```
Sub OpenProjectInEnvironment(filePath As String, _  
                             environmentGuidOrName As String)
```

Description

Opens an existing project in a specific environment. An already loaded project is closed.

This method is only functional in Control Builder Professional.

Parameters

Name	Type	Description
projectGuidOrName	String	The name or the GUID of the project to open.
environmentGuidOrName	String	Specifies the name or the GUID of the environment from which to open the project.

5.1.4.5 CloseProject

```
HRESULT CloseProject();
```

```
Sub CloseProject()
```

Description

Closes an existing project.

Parameters

None.

5.1.4.6 Project Constants XML example

XML example:

```
<?xml version="1.0" encoding="UTF-16"?>
<ProjectConstants xmlns="CBOpenIFSchema3_0">
  <ProjectConstant Name="Pi">
    <Type>real</Type>
    <Value>3.14159</Value>
  </ProjectConstant>
  <ProjectConstant Name="MaxNoOfBatches">
    <Type>dint</Type>
    <Value>60</Value>
  </ProjectConstant>
</ProjectConstants>
```

For further information see the [XML Schema](#) and [semantics for omitted attributes and elements](#).

5.1.4.7 GetProjectConstants

HRESULT GetProjectConstants([out, retval] BSTR* constants);

Function GetProjectConstants() As String

Description

Returns the Project Constants as an XML-string.

Parameters

None.

Return value

A string containing an XML description of the Project Constants is returned. See [project constants XML example](#).

5.1.4.8 SetProjectConstants

```
HRESULT SetProjectConstants([in] BSTR constants,  
                             [out, retval] BSTR* messages);
```

```
Function SetProjectConstants(constants As String) As String
```

Description

Replaces current Project Constants, if any, with new ones described in the **constants** parameter.

Parameters

Name	Type	Description
constants	String	A string containing an XML description of the new Project Constants. See project constants XML example .

Return value

A [message bucket](#) is returned.

5.1.5 Methods for Library

5.1.5.1 NewLibrary

```
HRESULT NewLibrary([in] BSTR libraryName,  
                   [in] BSTR directoryPath,  
                   [in] BSTR guid);
```

```
Sub NewLibrary(libraryName As String, _  
               directoryPath As String, _  
               guid As String)
```

Description

Creates a new empty library.

Parameters

Name	Type	Description
libraryName	String	Name of the new library
directoryPath	String	An optional parameter specifying the file location where the library will be saved. If this parameter is an empty string the library will be created in the “Libraries” folder (the User Libraries folder) which is located in parallel to the project’s folder. If the user wants the library to be saved in the same directory as the project, the project’s path must be specified. It is possible to use an UNC path. Note however the restriction described in chapter “Some additional remarks, File Organization Units ”.
guid	String	An optional parameter specifying a Globally Unique Identifier. If this parameter is an empty string the Control Builder will assign a GUID to the object. If the parameter is present the syntax must be as follows: "12345678-1234-1234-1234-123456789ABC"

5.1.5.2 InsertLibrary

```
HRESULT InsertLibrary([in] BSTR filePath);
```

```
Sub InsertLibrary(filePath As String)
```

Description

Inserts an existing library into the project.

Parameters

Name	Type	Description
filePath	String	<p>The file path, for an existing library, can be specified as follows:</p> <p>CB Professional:</p> <ul style="list-style-type: none">• The GUID of the library to insert.• The name, with or without version, of the library to insert. If version is omitted the library with the highest version will be inserted. Example: "Library1 1.2-3" or "Library1" <p>Compact CB:</p> <ul style="list-style-type: none">• A full path. Example: "C:\MyLibs\MyLib-1-0-0"• As a file name without directory path. In this case the "Libraries" folder (the User Libraries folder) is assumed. Example: "aLib-2-3-0".• As a file name with a relative path. The "Projects" folder (the folder where all projects are stored) is used as base folder. Example: "MyProject/aLib-2-3-0".• The environment variable "Libraries" can be used in order to specify the system library folder. Example: "Libraries:\AlarmEventLib-1-0-0". The syntax "%Libraries%\AlarmEventLib-1-0-0" is also allowed. <p>Please note that in all of the Compact CB examples above the library name can be specified as either a file name, for example "aLib-2-3-0", or as a name with or without version, for example "Library1 1.2-3" or "Library1". If version is omitted the library with the highest version will be used.</p>

5.1.5.3 DeleteLibrary

`HRESULT DeleteLibrary([in] BSTR libraryName);`

`Sub DeleteLibrary(libraryName As String)`

Description

Deletes the library from the project. All connections to the deleted library are removed. If not all connections can be removed (e.g. connection in released library), the library will not be deleted from the CB.

The library file on disk is not deleted.

Parameters

Name	Type	Description
libraryName	String	Name of the library to delete

5.1.5.4 RenameLibrary

```
HRESULT RenameLibrary([in] BSTR libraryName,  
                      [in] BSTR newLibraryName);
```

```
Sub RenameLibrary(libraryName As String, _  
                  newLibraryName As String)
```

Description

Renames the library. If the library is protected by a password, an error is returned. Any connections to this library from any other library or application will also be renamed.

Parameters

Name	Type	Description
libraryName	String	Name and version of the library to rename.
newLibraryName	String	The new name of the library. This name can not contain a version number.

5.1.5.5 GetLibraryProjectConstants

```
HRESULT GetLibraryProjectConstants([in] BSTR libraryName,  
                                   [out, retval] BSTR* constants);
```

```
Function GetLibraryProjectConstants(libraryName As String) As String
```

Description

Returns all Project Constants in a specific library as an XML-string.

Parameters

Name	Type	Description
libraryName	String	Name of the library

Return value

A string containing an XML description of the Project Constants is returned. See [project constants XML example](#).

5.1.5.6 **SetLibraryProjectConstants**

```
HRESULT SetLibraryProjectConstants([in] BSTR libraryName,  
                                   [in] BSTR constants,  
                                   [out, retval] BSTR* messages);  
  
Function SetLibraryProjectConstants(libraryName As String, _  
                                   constants As String) As String
```

Description

Replaces current Project Constants, if any, with new ones described in the **constants** parameter.

Parameters

Name	Type	Description
libraryName	String	Name of the library
constants	String	A string containing an XML description of the new Project Constants. See project constants XML example .

Return value

A [message bucket](#) is returned.

5.1.5.7 ListAvailableLibraries

```
HRESULT ListAvailableLibraries([out, retval] BSTR* libraries);
```

```
Function ListAvailableLibraries() As String
```

Description

Returns a list of available libraries in the system as an XML-string. In CB Professional the list contains all libraries in the system. In Compact CB the list contains all standard libraries aswell as all user libraries stored in the user library folder and the current projects folder.

Note: To list the libraries in a project opened in Control Builder please use the GetProjectTree method. See [GetProjectTree](#).

Return value

A string containing an XML description of all available libraries in the system is returned. See example below.

CB Professional

```
<?xml version="1.0" encoding="UTF-16"?>
<Libraries xmlns="CBOpenIFSSchema3_0">
  <Library Name="AlarmEventLib" MajorVersion="1" MinorVersion="8" Revision="0"
  Guid="344277F1-3174-4094-907F-640BF713A78D"/>
  <Library Name="BasicGraphicLib" MajorVersion="1" MinorVersion="5" Revision="0"
  Guid="C896B275-E0BC-4780-9713-6CE417E3FE97"/>
  <Library Name="BasicLib" MajorVersion="1" MinorVersion="9" Revision="2"
  Guid="B19AFC97-D36E-4262-9B2F-87EAB843CC58"/>
  ...
  <Library Name="MyLib" MajorVersion="1" MinorVersion="0" Revision="0"
  Guid="2AE8A39E-84F1-4F2B-9F6D-361B7E3B8684"/>
  ...
  <Library Name="UDPCCommLib" MajorVersion="1" MinorVersion="3" Revision="0"
  Guid="C7F0A15E-BB73-4F8C-9493-D75BE8476D3C"/>
</Libraries>
```

Compact CB

```
<?xml version="1.0" encoding="UTF-16"?>
<Libraries xmlns="CBOpenIFSSchema3_0">
  <Library Name="AlarmEventLib" MajorVersion="1" MinorVersion="8" Revision="0"
  FilePath="Libraries:\"/>
  <Library Name="BasicGraphicLib" MajorVersion="1" MinorVersion="5" Revision="0"
  FilePath="Libraries:\"/>
  <Library Name="BasicLib" MajorVersion="1" MinorVersion="9" Revision="2"
  FilePath="Libraries:\"/>
  ...
  <Library Name="MyLib" MajorVersion="1" MinorVersion="0" Revision="0"
  FilePath="C:\ABB Industrial IT Data\Engineer IT Data\Compact Control Builder AC
  800M\Projects\Libraries\"/>
  ...
  <Library Name="UDPCCommLib" MajorVersion="1" MinorVersion="3" Revision="0"
  FilePath="Libraries:\"/>
</Libraries>
```

5.1.6 Methods for Hardware Library

Specific files can be added to a hardware library. To select what kind of file to add, the parameter named **typeOfFile** of the type **enum CBOpenIFHardwareLibraryFileType** can have one of the following values:

- OI_HELPFILE
- OI_ICONFILE

5.1.6.1 NewHardwareLibrary

```
HRESULT NewHardwareLibrary([in] BSTR hardwareLibraryName,  
                            [in] BSTR directoryPath,  
                            [in] BSTR guid);
```

```
Sub NewHardwareLibrary(hardwareLibraryName As String, _  
                      directoryPath As String, _  
                      guid As String)
```

Description

Creates a new empty hardware library.

Parameters

Name	Type	Description
hardwareLibraryName	String	Name of the new hardware library
directoryPath	String	An optional parameter specifying the file location where the hardware library will be saved. If this parameter is an empty string the hardware library will be created in the “Libraries\Hardware” folder (the User Hardware Libraries folder) which is located in parallel to the project’s folder. If the user wants the hardware library to be saved in the same directory as the project, the project’s path must be specified. It is possible to use an UNC path. Note however the restriction described in chapter “Some additional remarks, File Organization Units ”.
guid	String	An optional parameter specifying a Globally Unique Identifier. If this parameter is an empty string the Control Builder will assign a GUID to the object. If the parameter is present the syntax must be as follows: "12345678-1234-1234-1234-123456789ABC"

5.1.6.2 InsertHardwareLibrary

```
HRESULT InsertHardwareLibrary([in] BSTR filePath);
```

```
Sub InsertHardwareLibrary(filePath As String)
```

Description

Inserts an existing hardware library into the project.

Parameters

Name	Type	Description
filePath	String	<p>The file path, for an existing hardware library, can be specified as follows:</p> <p>CB Professional:</p> <ul style="list-style-type: none">• The GUID of the hardware library to open.• The name, with or without version, of the hardware library to insert. If version is omitted the library with the highest version will be inserted. Example: “HWLibrary1 1.2-3” or “HWLibrary1” <p>Compact CB:</p> <ul style="list-style-type: none">• A full path. Example: “C:\MyHWLibs\MyHWLib-1-0-0”• As a file name without directory path. In this case the “Libraries\Hardware” folder (the User Hardware Libraries folder) is assumed. Example: “aHWLib-2-3-0”.• As a file name with a relative path. The “Projects” folder (the folder where all projects are stored) is used as base folder. Example: “MyProject/aHWLib-2-3-0”.• The environment variable “HWLibraries” can be used in order to specify the system library folder. Example: “HWLibraries\BasicHWLib-1-0-0”. The syntax “%HWLibraries%\BasicHWLib-1-0-0” is also allowed. <p>Please note that in all of the Compact CB examples above the hardware library name can be specified as either a file name, for example “aHWLib-2-3-0”, or as a name with or without version, for example “HWLibrary1 1.2-3” or “HWLibrary1”. If version is omitted the hardware library with the highest version will be used.</p>

5.1.6.3 DeleteHardwareLibrary

```
HRESULT DeleteHardwareLibrary([in] BSTR hardwareLibraryName);
```

```
Sub DeleteHardwareLibrary(hardwareLibraryName As String)
```

Description

Removes the hardware library from the project. If the hardware library is in use, e.g. is connected to a controller, the library will not be removed from the CB.

The hardware library file(s) on disk will not be deleted.

Parameters

Name	Type	Description
hardwareLibraryName	String	Name of the hardware library to remove.

5.1.6.4 RenameHardwareLibrary

```
HRESULT RenameHardwareLibrary([in] BSTR hardwareLibraryName,  
                               [in] BSTR newHardwareLibraryName);
```

```
Sub RenameHardwareLibrary(hardwareLibraryName As String, _  
                           newHardwareLibraryName As String)
```

Description

Renames the hardware library. If the hardware library is protected by a password, an error is returned. Any connections to this hardware library from any controller will also be renamed.

Parameters

Name	Type	Description
hardwareLibraryName	String	Name and version of the hardware library to rename.
newHardwareLibraryName	String	The new name of the hardware library. This name can not contain a version number.

5.1.6.5 AddHardwareLibraryFile

```
HRESULT AddHardwareLibraryFile([in] BSTR hardwareLibraryName,  
                                [in] enum COpenIFHardwareLibraryFileType typeOfFile,  
                                [in] BSTR filePath,  
                                [in] BSTR version);
```

```
Sub AddHardwareLibraryFile(hardwareLibraryName As String, _  
                            typeOfFile As COpenIFHardwareLibraryFileType, _  
                            filePath As String, _  
                            version As String)
```

Description

Adds a hardware file to a hardware library.

Parameters

Name	Type	Description
hardwareLibraryName	String	The GUID or the name, including version, of the hardware library in which to add the hardware library file.
typeOfFile	enum COpenIFHardwareLibraryFileType	Decides what kind of file to add.
filePath	String	The full file path to the hardware library file to add.
version	String	Version of the added file.

5.1.6.6 GetHardwareLibraryFiles

```
HRESULT GetHardwareLibraryFiles([in] BSTR hardwareLibraryName,  
                                [out, retval] BSTR* hardwareLibraryFiles);
```

```
Function GetHardwareLibraryFiles(hardwareLibraryName As String _  
                                ) As String
```

Description

Returns the hardware library files in the hardware library specified by **hardwareLibraryName**, as an XML-string.

Parameters

Name	Type	Description
hardwareLibraryName	String	The GUID or the name, including version, of the hardware library from which to get the hardware library files.

Return value

A string containing an XML description of the hardware library files is returned. See example below.

```
<?xml version="1.0" encoding="utf-16"?>  
<HardwareLibraryFiles xmlns="CBOpenIFSschema3_0">  
  <DIWImportFiles/>  
  <HardwareHelpFiles>  
    <HWHelpFile Name="HelpFile" Version="" Timestamp="2015-04-15-06:45:09.273"  
Extension="chm"/>  
  </HardwareHelpFiles>  
  <HardwareIconFiles>  
    <HWIconFile Name="CustomIcon1" Version="" Timestamp="2015-04-15-06:45:12.293"  
Extension="ico"/>  
    <HWIconFile Name="CustomIcon2" Version="" Timestamp="2015-04-15-06:45:15.366"  
Extension="ico"/>  
  </HardwareIconFiles>  
</HardwareLibraryFiles>
```

5.1.6.7 DeleteHardwareLibraryFile

```
HRESULT DeleteHardwareLibraryFile([in] BSTR hardwareLibraryName,  
                                  [in] enum COpenIFHardwareLibraryFileType typeOfFile,  
                                  [in] BSTR fileName);  
  
Sub DeleteHardwareLibraryFile(hardwareLibraryName As String, _  
                              typeOfFile As COpenIFHardwareLibraryFileType, _  
                              fileName As String)
```

Description

Deletes a hardware file from a hardware library.

Parameters

Name	Type	Description
hardwareLibraryName	String	The GUID or the name, including version, of the hardware library in which to delete the hardware library file.
typeOfFile	enum COpenIFHardwareLibraryFileType	Decides what kind of file to delete.
fileName	String	The name of the hardware library file to delete.

5.1.6.8 ListAvailableHardwareLibraries

```
HRESULT ListAvailableHardwareLibraries (  
    [out, retval] BSTR* hardwareLibraries);
```

```
Function ListAvailableHardwareLibraries() As String
```

Description

Returns a list of available hardware libraries in the system as an XML-string. In CB Professional the list contains all hardware libraries in the system. In Compact CB the list contains all standard hardware libraries as well as all user hardware libraries stored in the user hardware library folder and the current projects folder.

Note: To list the hardware libraries in a project opened in Control Builder please use the `GetProjectTree` method. See [GetProjectTree](#).

Return value

A string containing an XML description of all available hardware libraries in the system is returned. See example below.

CB Professional

```
<?xml version="1.0" encoding="UTF-16"?>  
<HardwareLibraries xmlns="CBOpenIFSschema3_0">  
  <HardwareLibrary Name="ABBDrvFenaCI871HwLib" MajorVersion="1" MinorVersion="0"  
Revision="3" Guid="17839F4B-9E45-4B20-8E82-D6A30C599B5F"/>  
  <HardwareLibrary Name="ABBDrvFenaCI871HwLib" MajorVersion="2" MinorVersion="0"  
Revision="0" Guid="886B1ECC-6B73-470A-B8FF-F0BA1C2BFDF2"/>  
  ...  
  <HardwareLibrary Name="BasicHwLib" MajorVersion="6" MinorVersion="1" Revision="0"  
Guid="93F29730-592F-4526-8274-919C6660EDF9"/>  
  ...  
  <HardwareLibrary Name="MyHwLib" MajorVersion="1" MinorVersion="0" Revision="0"  
Guid="0E97A715-3163-4482-9D57-392A5A494E52"/>  
  ...  
  <HardwareLibrary Name="UDPHwLib" MajorVersion="1" MinorVersion="3" Revision="0"  
Guid="AC734726-FD54-4679-A2B6-0FE53D28CDD1"/>  
</HardwareLibraries>
```

Compact CB

```
<?xml version="1.0" encoding="UTF-16"?>  
<HardwareLibraries xmlns="CBOpenIFSschema3_0">  
  <HardwareLibrary Name="ABBDrvFenaCI871HwLib" MajorVersion="1" MinorVersion="0"  
Revision="3" FilePath="HWLibraries:\"/>  
  <HardwareLibrary Name="ABBDrvFenaCI871HwLib" MajorVersion="2" MinorVersion="0"  
Revision="0" FilePath="HWLibraries:\"/>  
  ...  
  <HardwareLibrary Name="BasicHwLib" MajorVersion="6" MinorVersion="1" Revision="0"  
FilePath="HWLibraries:\"/>  
  ...  
  <HardwareLibrary Name="MyHwLib" MajorVersion="1" MinorVersion="0" Revision="0"  
FilePath="C:\ABB Industrial IT Data\Engineer IT Data\Compact Control Builder AC  
800M\Projects\Libraries\Hardware\"/>  
  ...  
  <HardwareLibrary Name="UDPHwLib" MajorVersion="1" MinorVersion="10" Revision="0"  
FilePath="HWLibraries:\"/>  
</HardwareLibraries>
```

5.1.7 Methods for Application

5.1.7.1 NewApplication

```
HRESULT NewApplication([in] BSTR applicationName,  
                        [in] BSTR directoryPath,  
                        [in] BSTR guid,  
                        [in] BSTR templateName);
```

```
Sub NewApplication(applicationName As String, _  
                  directoryPath As String, _  
                  guid As String, _  
                  templateName As String)
```

Description

Creates a new application using the template **templateName**. If no template is specified, an empty application is created.

Parameters

Name	Type	Description
applicationName	String	Name of the new application. To create the application in a specific folder, the folder path shall be provided as part of the application name. Example: "MyFolder.MyApp". To create the application directly under the "Applications" leaf in Project Explorer the folder path shall be omitted (application name only shall be provided).
directoryPath	String	Remark: This method will fail if the provided folder path points to a non-existing folder. An optional parameter specifying the file location where the application will be saved. If this parameter is an empty string the application will be created in the same directory as the project. It is possible to use an UNC path. Note however the restriction described in chapter "Some additional remarks, File Organization Units ".
Guid	String	An optional parameter specifying a Globally Unique Identifier. If this parameter is an empty string the Control Builder will assign a GUID to the object. If the parameter is present the syntax must be as follows: "12345678-1234-1234-1234-123456789ABC"

templateName	String	An optional parameter. The name of the application template to use, when creating the new application. If it is left emty, an empty application will be created.
--------------	--------	--

5.1.7.2 InsertApplication

```
HRESULT InsertApplication([in] BSTR filePath);
```

```
Sub InsertApplication(filePath As String)
```

Description

Inserts an existing application into the project.

Parameters

Name	Type	Description
filePath	String	<p>The file path, for an existing application, can be specified as follows:</p> <ul style="list-style-type: none">• A full path. Example: “C:\MyApps\MyApp-1-0-0”• As a file name without directory path. In this case the folder of the current project is assumed. Example: “aApp”.• The environment variable “Examples” can be used in order to specify the examples folder where some template applications reside. Example: “Examples:\ControlExamples\TankControl1-1-0-0”. <p>The syntax “%Examples%\ ControlExamples\TankControl1-1-0-0” is also allowed.</p> <p>Remark: The application is always inserted under the “Applications” leaf in Project Explorer. Use “MoveFolderObject” to move the application to a desired folder.</p>

5.1.7.3 RenameApplication

```
HRESULT RenameApplication([in] BSTR applicationName,  
                           [in] BSTR newApplicationName);
```

```
Sub RenameApplication(applicationName As String, _  
                      newApplicationName As String)
```

Description

Renames the application. If the application is protected by a password, an error is returned. Any connections to this application from any controller will also be renamed.

Parameters

Name	Type	Description
applicationName	String	Name of the application to rename.
newApplicationName	String	<p>The new name of the application. This name can not contain a version number.</p> <p>Remark: The new application name must not contain a folder path.</p>

5.1.7.4 DeleteApplication

```
HRESULT DeleteApplication([in] BSTR applicationName);
```

```
Sub DeleteApplication(applicationName As String)
```

Description

Deletes the application from the project and, when running Compact CB, the application file is deleted from disk.

Parameters

Name	Type	Description
applicationName	String	Name of the application to delete. Remark: Is's support to provid a folder path as part of the application name. However, this path is disregarded.

5.1.7.5 Application Variables XML example

XML example:

```
<?xml version="1.0" encoding="UTF-16"?>
<ApplicationVariables xmlns="CBOpenIFSchema3_0">
  <GlobalVariables>
    <GlobalVariable Name="AglobalVar" Type="CalendarStruct" Attribute="retain"
    InitialValue="" ReadPermission="" WritePermission="" AuthenticationLevel="None"
    TypePath="System.CalendarStruct"/>
    <GlobalVariable Name="GlobVar2" Type="real" Attribute="retain" InitialValue=""
    ReadPermission="" WritePermission="" AuthenticationLevel="None"
    TypePath="System.real">
      <Description>Description of GlobVar2</Description>
    </GlobalVariable>
  </GlobalVariables>
  <Variables>
    <Variable Name="Var1" Type="dint" Attribute="retain" InitialValue=""
    ReadPermission="" WritePermission="" AuthenticationLevel="None" BatchProperty=""
    TypePath="System.dint">
      <Description>Description of Var1</Description>
    </Variable>
  </Variables>
  <Signals>
    <Signal Name="Signal1" Path="MyApp.Var1" Direction="in" AcknowledgeGroup="auto">
      <Description>Description of Signal1</Description>
    </Signal>
  </Signals>
</ApplicationVariables>
```

For further information see the [XML Schema](#) and [semantics for omitted attributes and elements](#).

5.1.7.6 **GetApplicationVariables**

```
HRESULT GetApplicationVariables([in] BSTR applicationName,  
                               [out, retval] BSTR* variables);
```

```
Function GetApplicationVariables(applicationName As String) As String
```

Description

Returns variables, global variables, and signals from the selected application as an XML-string.

Parameters

Name	Type	Description
applicationName	String	Name of the application containing the variables to retrieve.

Return value

A string containing an XML description of the Application Variables is returned. See [application variables XML example](#).

5.1.7.7 SetApplicationVariables

```
HRESULT SetApplicationVariables([in] BSTR applicationName,  
                                [in] BSTR variables,  
                                [out, retval] BSTR* messages);
```

```
Function SetApplicationVariables(applicationName As String, _  
                                variables As String) As String
```

Description

Sets the variables, global variables, and signals using an XML-string as the parameter variables. The existing variables and global variables in the selected application are discarded. New variables are added according to the XML string.

Parameters

Name	Type	Description
applicationName	String	Name of the application containing the variables to set.
Variables	String	A string containing an XML description of the new Application Variables. See application variables XML example .

Return value

A [Message bucket](#) is returned.

5.1.7.8 Application Properties XML example

XML example:

```
<?xml version="1.0" encoding="UTF-16"?>
<ApplicationProperties xmlns="CBOpenIFSchema3_0" SimulationMark="0"
SILLevel="NonSIL"/>
```

For further information see the [XML Schema](#) and [semantics for omitted attributes and elements](#).

See also [What XML elements and attributes are needed by Set and New – methods?](#)

5.1.7.9 GetApplicationProperties

```
HRESULT GetApplicationProperties([in] BSTR applicationName,
                                [out, retval] BSTR*
                                propertiesContent);
```

```
Function GetApplicationProperties(applicationName As String) As
String
```

Description

Gets the properties of the specified application as a string.

Parameters

Name	Type	Description
applicationName	String	Name of the application. Example: "MyApplication"

Return value

An XML string containing the properties for the application is returned. See [application properties XML example](#).

5.1.7.10 SetApplicationProperties

```
HRESULT SetApplicationProperties([in] BSTR applicationName,  
                                [in] BSTR propertiesContent);
```

```
Sub SetApplicationProperties(applicationName As String, _  
                            propertiesContent As String)
```

Description

Sets the properties of the specified application.

Parameters

Name	Type	Description
applicationName	String	Name of the application. Example: "MyApplication"
propertiesContent	String	An XML containing the properties for the application. See the application properties XML example .

5.1.8 Methods for Execution Order

In an application, runtime instances are grouped in different structures, for example the Diagrams structure. To select what structure to retrieve or alter the execution order, the first parameter named **typeOfExecutionInstance** of the type **enum CBOpenIFExecutionInstanceType** can have one of the following values:

- OI_DIAGRAMS

5.1.8.1 Execution Order XML example

XML example:

```
<?xml version="1.0" encoding="UTF-16"?>
<ExecutionOrder xmlns="CBOpenIFSchema3_0">
  <ExecutionGroup TaskName="Controller_1.Fast">
    <ExecutionInstance Name="Diagram1"/>
    <ExecutionInstance Name="Diagram3"/>
  </ExecutionGroup>
  <ExecutionGroup TaskName="Controller_1.Normal">
    <ExecutionInstance Name="Diagram4"/>
    <ExecutionInstance Name="Diagram2"/>
  </ExecutionGroup>
</ExecutionOrder>
```

For further information see the [XML Schema](#) and [semantics for omitted attributes and elements](#).

5.1.8.2 GetExecutionOrder

```
HRESULT GetExecutionOrder(  
    [in] enum CBOpenIFExecutionInstanceType  
    typeOfExecutionInstance,  
    [in] BSTR applicationName,  
    [out, retval] BSTR* executionOrderContent);
```

```
Function GetExecutionOrder( _  
    typeOfExecutionInstance As CBOpenIFExecutionInstanceType,  
    _  
    applicationName As String) As String
```

Description

Returns an XML-string containing the execution order of the runtime instances in the selected structure of the specified application grouped by their task connection.

Parameters

Name	Type	Description
typeOfExecutionInstance	enum CBOpenIFExecutionInstanceType	Determines in which structure to get the execution order from.
applicationName	String	Name of the application containing the runtime instances.

Return value

A string containing an XML description of the Execution Order is returned. See [execution order XML example](#).

5.1.8.3 SetExecutionOrder

```
HRESULT SetExecutionOrder(  
    [in] enum COpenIFExecutionInstanceType  
    typeOfExecutionInstance,  
        [in] BSTR applicationName,  
        [in] BSTR executionOrderContent,  
        [out, retval] BSTR* messages);  
  
Function SetExecutionOrder( _  
    typeOfExecutionInstance As COpenIFExecutionInstanceType,  
    _  
        applicationName As String, _  
        executionOrderContent As String) As String
```

Description

Sets the execution order of the runtime instances in the selected structure of the specified application grouped by their task connection.

Parameters

Name	Type	Description
typeOfExecutionInstance	enum COpenIFExecutionInstanceType	Determines in which structure to set the execution order of.
applicationName	String	Name of the application containing the runtime instances.
executionOrderContent	String	An XML string containing the execution order of the runtime instances in the selected application. See the execution order XML example .

Return value

A [message bucket](#) is returned.

5.1.9 Methods for Task Connection

5.1.9.1 GetTaskConnection

```
HRESULT GetTaskConnection([in] BSTR objectPath,  
                           [out, retval] BSTR* taskConnection);
```

```
Function GetTaskConnection(objectPath As String) As String
```

Description

Returns the task connection as a string. The parameter **objectPath** is the path to the object (instance) the method retrieves the task connection information from.

Parameters

Name	Type	Description
objectPath	String	The path to the object (instance) the method retrieves the task connection information from. Example: "MyApplication.MyProgram"

Return value

A string containing the task connection is returned. A task connection can not contain the version of the controller. Example: "MyController.Fast"

5.1.9.2 SetTaskConnection

```
HRESULT SetTaskConnection([in] BSTR objectPath,  
                           [in] BSTR taskConnection);
```

```
Sub SetTaskConnection(objectPath As String, taskConnection As String)
```

Description

Sets the task connection for an object (formal instance). The parameter **objectPath** is the path to the object (formal instance).

Parameters

Name	Type	Description
objectPath	String	The path to the object (formal instance) to set the task connection for. Example: "MyApplication.MyProgram" Example: "MyLib.MyCMType.MyCM"
taskConnection	String	The new task connection value (without version on the controller name). Example: "MyController.Fast"

5.1.10 Methods for Controller

5.1.10.1 NewController

```
HRESULT NewController([in] BSTR controllerName,  
                      [in] BSTR controllerType,  
                      [in] BSTR directoryPath,  
                      [in] BSTR guid,  
                      [in] BSTR templateName);
```

```
Sub NewController(controllerName As String, _  
                  controllerType As String, _  
                  DirectoryPath As String, _  
                  guid As String, _  
                  templateName As String);
```

Description

Creates a new controller of the type specified by controllerType. If an “AC 800M HI” is specified, a PM867/TP830 CPU and an appropriate safety module (SM) are created automatically. Otherwise, an empty controller is created.

Optionally, a CPU type can be specified by extending the controllerType value with the exact name of the CPU type. The format shall be <Controller type name>.<CPU type name>. After a successful call a controller of the specified type with the specified CPU is created.

Parameters

Name	Type	Description
controllerName	String	Name of the new controller
controllerType	String	Type of controller. Allowed values are “Soft Controller”, “Soft Controller HI”, “AC 800M” and “AC 800M HI”. If a CPU is specified the exact name of the CPU type must be provided, for example “Soft Controller.CPU PA”, “Soft Controller HI.CPU HI”, “AC 800M.PM866 / TP830”, and “AC 800M HI.PM867 HI/ TP830”.
directoryPath	String	An optional parameter specifying the file location where the controller will be saved. If this parameter is an empty string the controller will be created in the same directory as the project. It is possible to use an UNC path. Note however the restriction described in chapter “Some additional remarks, File Organization Units ”.
Guid	String	An optional parameter specifying a Globally Unique Identifier. If this parameter is an empty string the Control Builder will assign a GUID to the object. If the parameter is present the syntax must be as follows: “12345678-1234-1234-1234-123456789ABC”

templateName	String	Optional parameter. The name, with or without version, of the hardware library containing the controller type. If the parameter is an empty string, the latest version of BasicHwLib loaded in the project will be used.
--------------	--------	--

5.1.10.2 InsertController

```
HRESULT InsertController([in] BSTR filePath);
```

```
Sub InsertController(filePath As String)
```

Description

Inserts an existing Controller into the project.

Parameters

Name	Type	Description
filePath	String	<p>The file path, for an existing controller, can be specified as follows:</p> <ul style="list-style-type: none">• A full path. Example: “C:\MyControllers\MyController-1-0-0”• As a file name without directory path. In this case the folder of the current project is assumed. Example: “aController”.• The environment variable “Examples” can be used in order to specify the examples folder where some template controllers reside. Example: “Examples\ControlExamples\TankController-1-0-0”. <p>The syntax “%Examples%\ControlExamples\TankController-1-0-0” is also allowed.</p>

5.1.10.3 RenameController

```
HRESULT RenameController([in] BSTR controllerName,  
                          [in] BSTR newControllerName);  
  
Sub RenameController(controllerName As String, _  
                      newControllerName As String)
```

Description

Renames the controller.

Parameters

Name	Type	Description
controllerName	String	Name of the controller to rename
newControllerName	String	The new name of the controller. This name can not contain a version number.

5.1.10.4 DeleteController

```
HRESULT DeleteController([in] BSTR controllerName);
```

```
Sub DeleteController(controllerName As String)
```

Description

Deletes the controller from the project and, when running Compact CB, the controller file is deleted from disk.

Parameters

Name	Type	Description
controllerName	String	Name of the controller to delete

5.1.10.5 **GetSystemIdentity**

```
HRESULT GetSystemIdentity([in] BSTR controllerName,  
                           [out,retval] BSTR* systemIdentity);
```

```
Function GetSystemIdentity(controllerName As String) As String
```

Description

Gets the System Identity of the controller as a string.

Parameters

Name	Type	Description
controllerName	String	Name of the controller. Example: “MyController”

Return value

A string containing the System Identity is returned. Example: “172.16.0.0”

5.1.10.6 SetSystemIdentity

```
HRESULT SetSystemIdentity([in] BSTR controllerName,  
                           [in] BSTR systemIdentity);
```

```
Sub SetSystemIdentity(controllerName As String, _  
                      systemIdentity As String)
```

Description

Sets the System Identity of the controller.

Parameters

Name	Type	Description
controllerName	String	Name of the controller. Example: “MyController”
systemIdentity	String	The new System Identity of the controller. Example: “172.16.0.0”

5.1.10.7 Controller Properties XML example

XML example:

```
<?xml version="1.0" encoding="UTF-16"?>
<ControllerProperties xmlns="CBOpenIFSschema3_0" HWSimulation="0"/>
```

For further information see the [XML Schema](#) and [semantics for omitted attributes and elements](#).

See also [What XML elements and attributes are needed by Set and New – methods?](#)

5.1.10.8 GetControllerProperties

```
HRESULT GetControllerProperties([in] BSTR controllerName,
                               [out, retval] BSTR*
                               propertiesContent);
```

```
Function GetControllerProperties(controllerName As String) As String
```

Description

Gets the properties of the specified controller as a string.

Parameters

Name	Type	Description
controllerName	String	Name of the controller. Example: "MyController"

Return value

An XML string containing the properties for the controller is returned. See [controller properties XML example](#).

5.1.10.9 SetControllerProperties

```
HRESULT SetControllerProperties([in] BSTR controllerName,  
                               [in] BSTR propertiesContent);
```

```
Sub SetControllerProperties(controllerName As String, _  
                           propertiesContent As String)
```

Description

Sets the properties of the specified controller.

Parameters

Name	Type	Description
controllerName	String	Name of the controller. Example: "MyController"
propertiesContent	String	An XML string containing the properties for the controller. See the controller properties XML example .

5.1.10.10 Controller Settings XML example

XML example for PA controller:

```
<?xml version="1.0" encoding="UTF-16"?>
<CommonControllerSettings xmlns="CBOpenIFSchema3_0" ChangeTime="1979-12-31-
00:00:00.000" OLUStopTime="3000">
  <FatalOverrun Limit="10" Reaction="1"/>
  <LoadBalancing OverLoadCompensation="1"/>
  <ErrorHandler>
    <SystemDiagnostics>
      <Log Low="0" Medium="1" High="1" Critical="1" Fatal="1"/>
      <Event Low="0" Medium="0" High="1" Critical="1" Fatal="1"/>
      <Reset Low="0" Medium="0" High="0" Critical="1" Fatal="1"/>
    </SystemDiagnostics>
    <Execution>
      <Log Low="0" Medium="1" High="1" Critical="1" Fatal="1"/>
      <Event Low="0" Medium="0" High="1" Critical="1" Fatal="1"/>
      <Reset Low="0" Medium="0" High="0" Critical="1" Fatal="1"/>
    </Execution>
    <IO>
      <Log Low="0" Medium="0" High="0" Critical="1" Fatal="1"/>
      <Event Low="0" Medium="0" High="0" Critical="1" Fatal="1"/>
      <Reset Low="0" Medium="0" High="0" Critical="1" Fatal="1"/>
    </IO>
  </ErrorHandler>
</CommonControllerSettings>
```

XML example for HI controller:

```
<?xml version="1.0" encoding="UTF-16"?>
<CommonControllerSettings xmlns="CBOpenIFSchema3_0" ChangeTime="1979-12-31-
00:00:00.000" OLUStopTime="3000">
  <FatalOverrun Limit="10" Reaction="2"/>
  <Safety DemandMode="2" FDRT="3000" SIL3AppStartValUpdInterval="86400"/>
  <ErrorHandler>
    <SystemDiagnostics>
      <Log Low="0" Medium="1" High="1" Critical="1" Fatal="1"/>
      <Event Low="0" Medium="0" High="1" Critical="1" Fatal="1"/>
      <Reset Low="0" Medium="0" High="1" Critical="1" Fatal="1"/>
      <SystemAlarmOutput Low="0" Medium="0" High="1" Critical="1" Fatal="1"/>
    </SystemDiagnostics>
    <Execution>
      <Log Low="0" Medium="1" High="1" Critical="1" Fatal="1"/>
      <Event Low="0" Medium="0" High="1" Critical="1" Fatal="1"/>
      <Reset Low="0" Medium="0" High="1" Critical="1" Fatal="1"/>
      <SystemAlarmOutput Low="0" Medium="0" High="1" Critical="1" Fatal="1"/>
    </Execution>
    <IO>
      <Log Low="0" Medium="0" High="1" Critical="1" Fatal="1"/>
      <Event Low="0" Medium="0" High="1" Critical="1" Fatal="1"/>
      <Reset Low="0" Medium="0" High="1" Critical="1" Fatal="1"/>
      <SystemAlarmOutput Low="0" Medium="0" High="1" Critical="1" Fatal="1"/>
    </IO>
  </ErrorHandler>
</CommonControllerSettings>
```

For further information see the [XML Schema](#) and [semantics for omitted attributes and elements](#).

See also [What XML elements and attributes are needed by Set and New – methods?](#)

5.1.10.11 GetControllerSettings

```
HRESULT GetControllerSettings([in] BSTR controllerName,  
                             [out, retval] BSTR*  
                             controllerSettings);
```

Function GetControllerSettings(controllerName As String) As String

Description

Gets the settings of the specified controller as a string.

Parameters

Name	Type	Description
controllerName	String	Name of the controller. Example: "MyController"

Return value

An XML string containing the properties for the controller is returned. See [controller settings XML example](#).

5.1.10.12 SetControllerSettings

```
HRESULT SetControllerSettings([in] BSTR controllerName,  
                             [in] BSTR controllerSettings);
```

```
Sub SetControllerSettings(controllerName As String, _  
                        controllerSettings As String)
```

Description

Sets the settings of the specified controller.

Parameters

Name	Type	Description
controllerName	String	Name of the controller. Example: "MyController"
controllerSettings	String	An XML string containing the settings for the controller. See the controller settings XML example .

5.1.11 Methods for DataType

5.1.11.1 DataType XML example

XML example:

```
<?xml version="1.0" encoding="UTF-16"?>
<DataType xmlns="CBOpenIFSchema3_0" Name="MyDataType" Protected="0" Hidden="0"
Scope="public">
  <Components>
    <Component Name="Comp1" Type="dint" Attribute="retain" InitialValue="7"
ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.dint">
      <Description>Description of Comp1</Description>
    </Component>
    <Component Name="Comp2" Type="real" Attribute="retain" InitialValue="1.0"
ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.real"/>
  </Components>
  <Description>General description of the datatype "MyDataType"</Description>
</DataType>
```

For further information see the [XML Schema](#) and [semantics for omitted attributes and elements](#).

5.1.11.2 NewDataType

```
HRESULT NewDataType([in] BSTR dataTypeName,  
                    [in] BSTR appOrLibName,  
                    [in] BSTR dataTypeContent,  
                    [out, retval] BSTR* messages);
```

```
Function NewDataType(dataTypeName As String, _  
                    appOrLibName As String, _  
                    dataTypeContent As String) As String
```

Description

Creates a new data type with the name **dataTypeName**. The parameter **dataTypeContent** is optional and sets the content of the data type using an XML-string. If it is empty, an empty data type will be created. If content is provided the “Name” attribute in the XML-string is ignored since the parameter **dataTypeName** is used as the data type name.

Parameters

Name	Type	Description
dataTypeName	String	The name of the data type to create
appOrLibName	String	The name of the application or library the data type will be created in.
dataTypeContent	String	Optional parameter. A string containing an XML description of the new data type. See data type XML example . If this string is empty, an empty data type will be created.

Return value

A [message bucket](#) is returned.

5.1.11.3 GetDataType

```
HRESULT GetDataType([in] BSTR dataTypePath,  
                    [out, retval] BSTR* dataTypeContent);
```

```
Function GetDataType(dataTypePath As String) As String
```

Description

Returns the content of the data type, specified by the parameter **dataTypePath**, as an XML-string.

Parameters

Name	Type	Description
dataTypePath	String	Path to the data type to retrieve information from. Example: "MyLibrary.MyDataType"

Return value

A string containing an XML description of the data type is returned. See [data type XML example](#)

5.1.11.4 SetDataType

```
HRESULT SetDataType([in] BSTR dataTypePath,  
                    [in] BSTR dataTypeContent,  
                    [out, retval] BSTR* messages);
```

```
Function SetDataType(dataTypePath As String, _  
                    dataTypeContent As String) As String
```

Description

Sets the content of the data type specified by **dataTypePath** using an XML-string as the parameter **dataTypeContent**. The “Name” attribute in the XML-string is ignored, i.e. you can’t change the name of the data type using the XML.

Parameters

Name	Type	Description
dataTypePath	String	Path to the data type to set information for. Example: “MyLibrary.MyDataType”
dataTypeContent	String	The information to set for the selected data type as an XML-string. See data type XML example

Return value

A [message bucket](#) is returned.

5.1.11.5 DeleteDataType

```
HRESULT DeleteDataType([in] BSTR dataTypePath);
```

```
Sub DeleteDataType(dataTypePath As String)
```

Description

Deletes the data type specified by **dataTypePath** from the project. If there are instances of the type, the type can not be removed.

Parameters

Name	Type	Description
dataTypePath	String	Path to the data type to delete. Example: "MyLibrary.MyDataType"

5.1.11.6 RenameDataType

```
HRESULT RenameDataType([in] BSTR dataTypePath,  
                        [in] BSTR newDataTypeName);
```

```
Sub RenameDataType(dataTypePath As String, newDataTypeName As String)
```

Description

Renames the data type specified by **dataTypePath** to **newDataTypeName**. The type attribute of variables of the renamed type will also be changed.

Parameters

Name	Type	Description
dataTypePath	String	Path to the data type to rename. Example: "MyLibrary.MyDataType"
newDataTypeName	String	The new name of the selected data type. (Only name, not complete path)

5.1.12 Methods for FunctionBlockType

5.1.12.1 FunctionBlockType XML example

XML example:

```
<?xml version="1.0" encoding="UTF-16"?>
<FunctionBlockType xmlns="CBOpenIFSchema3_0" Name="MyFBType" Protected="0" Hidden="0"
Scope="public" InteractionWindow="" AlarmOwner="0" SILLevel="NonSIL"
SimulationMark="0">
  <Parameters>
    <Parameter Name="Par1" Type="dint" Attribute="retain" Direction="in"
InitialValue="7" ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.dint">
      <Description>Description of Par1</Description>
    </Parameter>
    <Parameter Name="Par2" Type="CalendarStruct" Attribute="" Direction="in_out"
InitialValue="" ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.CalendarStruct"/>
  </Parameters>
  <ExtensibleParameters/>
  <Variables>
    <Variable Name="Var1" Type="real" Attribute="retain" InitialValue="7.0"
ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.real">
      <Description>Description of Var1</Description>
    </Variable>
  </Variables>
  <ExternalVariables>
    <ExternalVariable Name="ExtVar1" Type="dint" Attribute="constant"
ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.dint"/>
  </ExternalVariables>
  <FunctionBlocks>
    <FunctionBlock Name="FBInst1" Type="AnotherFBType" TaskConnection="NewCont.Fast"
TypePath="MyApplication 1.0-0.AnotherFBType">
      <Description>Description of the instance</Description>
    </FunctionBlock>
  </FunctionBlocks>
  <ControlModules/>
  <CodeBlocks>
    <STCodeBlock Name="MySTCodeBlock">
      <ST_Code>
        if Par1>0 then FBInst1( ParA := ExtVar1 );
        end_if;
      </ST_Code>
    </STCodeBlock>
  </CodeBlocks>
  <Description>Description of the FBType: "MyFBType"</Description>
</FunctionBlockType>
```

For further information see the [XML Schema](#) and [semantics for omitted attributes and elements](#).

See also [What XML elements and attributes are needed by Set and New – methods?](#)

5.1.12.2 NewFunctionBlockType

```
HRESULT NewFunctionBlockType([in] BSTR functionBlockTypeName,
                             [in] BSTR appOrLibName,
                             [in] BSTR functionBlockTypeContent,
                             [out, retval] BSTR* messages);

Function NewFunctionBlockType(functionBlockTypeName As String, _
                             appOrLibName As String, _
                             functionBlockTypeContent As String _
                             ) As String
```

Description

Creates a new function block type with the name **functionBlockTypeName**. The parameter **functionBlockTypeContent** is an optional parameter. It sets the content of the function block type using an XML-string. If it is empty, an empty function block type will be created. If content is provided the “Name” attribute in the XML-string is ignored since the parameter **functionBlockTypeName** is used as the function block type name.

Parameters

Name	Type	Description
functionBlockTypeName	String	The name of the function block type to create.
appOrLibName	String	The name of the application or library the function block type will be created in.
functionBlockTypeContent	String	Optional parameter. A string containing an XML description of the new function block type. See function block type XML example . If this string is empty, an empty function block type will be created.

Return value

A [message bucket](#) is returned.

5.1.12.3 GetFunctionBlockType

```
HRESULT GetFunctionBlockType([in] BSTR functionBlockTypePath,  
                             [out, retval] BSTR*  
                             functionBlockTypeContent);
```

```
Function GetFunctionBlockType(functionBlockTypePath As String _  
                             ) As String
```

Description

Returns the content of the function block type, specified by **functionBlockTypePath**, as an XML-string.

Parameters

Name	Type	Description
functionBlockTypePath	String	Path to the function block type to retrieve information from. Example: “MyLibrary.MyFBType”

Return value

A string containing an XML description of the data type is returned. See [function block type XML example](#).

5.1.12.4 SetFunctionBlockType

```
HRESULT SetFunctionBlockType([in] BSTR functionBlockTypePath,  
                             [in] BSTR functionBlockTypeContent,  
                             [out, retval] BSTR* messages);
```

```
Function SetFunctionBlockType(functionBlockTypePath As String, _  
                             functionBlockTypeContent As String _  
                             ) As String
```

Description

Sets the content of the function block type specified by **functionBlockTypePath** using an XML-string as the parameter **functionBlockTypeContent**. The “Name” attribute in the XML-string is ignored, i.e. you can’t change the name of the function block type using the XML.

Parameters

Name	Type	Description
functionBlockTypePath	String	Path to the function block type to set information for. Example: “MyLibrary.MyFBType”
functionBlockTypeContent	String	The information to set for the selected function block type as an XML-string. See function block type XML example .

Return value

A [message bucket](#) is returned.

5.1.12.5 DeleteFunctionBlockType

```
HRESULT DeleteFunctionBlockType([in] BSTR functionBlockTypePath);
```

```
Sub DeleteFunctionBlockType(functionBlockTypePath As String)
```

Description

Deletes the function block type specified by **functionBlockTypePath** from the project. If there are instances of the type, the type can not be removed.

Parameters

Name	Type	Description
functionBlockTypePath	String	Path to the function block type to delete. Example: "MyLibrary.MyFBType"

5.1.12.6 **RenameFunctionBlockType**

```
HRESULT RenameFunctionBlockType([in] BSTR functionBlockTypePath,  
                                [in] BSTR newFunctionBlockTypeName);  
  
Sub RenameFunctionBlockType(functionBlockTypePath As String, _  
                             newFunctionBlockTypeName As String)
```

Description

Renames the function block type specified by **functionBlockTypePath** to **newFunctionBlockTypeName**. The type attribute of function blocks of the renamed type will also be changed.

Parameters

Name	Type	Description
functionBlockTypePath	String	Path to the function block type to rename. Example: “MyLibrary.MyFunctionBlockType”
newFunctionBlockTypeName	String	The new name of the selected function block type. (Only name, not complete path)

5.1.13 Methods for FunctionBlock

5.1.13.1 FunctionBlock XML example

XML example:

```
<?xml version="1.0" encoding="UTF-16"?>
<FunctionBlock xmlns="CBOpenIFSschema3_0" Name="FBInst1" Type="AnotherFBType"
TaskConnection="NewCont.Fast" TypePath="MyApplication 1.0-0.AnotherFBType">
  <Description>Description of the instance</Description>
</FunctionBlock>
```

For further information see the [XML Schema](#) and [semantics for omitted attributes and elements](#).

See also [What XML elements and attributes are needed by Set and New – methods?](#)

5.1.13.2 NewFunctionBlock

```
HRESULT NewFunctionBlock([in] BSTR functionName,  
                        [in] BSTR functionBlockType,  
                        [in] BSTR pathToParent,  
                        [in] BSTR functionBlockContent,  
                        [out, retval] BSTR* messages);
```

```
Function NewFunctionBlock(functionBlockName As String, _  
                        functionBlockType As String, _  
                        pathToParent As String, _  
                        functionBlockContent As String) As String
```

Description

Creates a new function block with the name **functionBlockName** and of the type **functionBlockType**. The parameter **functionBlockContent** is optional. It sets the content of the function block using an XML-string. If it is empty, an empty functionBlock will be created. If content is provided the “Name” and “Type” attributes in the XML-string are ignored since the parameter **functionBlockName** is used as the function block name and the parameter **functionBlockType** is used as the function block type name.

Parameters

Name	Type	Description
functionBlockName	String	The name of the function block to create.
functionBlockType	String	The type of function block type the created function block should be an instance of.
pathToParent	String	Path to the object where to put the new function block. Example: “MyLib.MyFBType”
functionBlockContent	String	Optional parameter. A string containing an XML description of the new function block. See function block XML example . If this string is empty, an empty function block will be created.

Return value

A [message bucket](#) is returned.

5.1.13.3 GetFunctionBlock

```
HRESULT GetFunctionBlock([in] BSTR functionBlockPath,  
                          [out, retval] BSTR* functionBlockContent);
```

```
Function GetFunctionBlock(functionBlockPath As String) As String
```

Description

Returns the content of the function block, specified by **functionBlockPath**, as an XML-string.

Parameters

Name	Type	Description
functionBlockPath	String	Path to the function block to retrieve information from. Example: "MyLibrary.MyFBType.MyFB". <i>Note! The path can contain several instance levels.</i> <i>Example:</i> <i>"MyApplic.Program1.FBInst1.FBInst2.FBInst3"</i>

Return value

A string containing an XML description of the instance is returned. See [function block XML example](#).

5.1.13.4 SetFunctionBlock

```
HRESULT SetFunctionBlock([in] BSTR functionBlockPath,  
                        [in] BSTR functionBlockContent,  
                        [out, retval] BSTR* messages);  
  
Function SetFunctionBlock(functionBlockPath As String, _  
                        functionBlockContent As String) As String
```

Description

Sets the content of the function block specified by **functionBlockPath** using an XML-string as the parameter **functionBlockContent**. You can change the name and the type of the function block using the XML.

Parameters

Name	Type	Description
functionBlockPath	String	Path to the function block to set information for. Example: "MyLibrary.MyFBType.MyFB" <i>Note! The path can contain several instance levels.</i> <i>Example:</i> <i>"MyApplic.Program1.FBInst1.FBInst2.FBInst3"</i>
functionBlockContent	String	The information to set for the selected function block as an XML-string. See function block XML example .

Return value

A [message bucket](#) is returned.

5.1.13.5 DeleteFunctionBlock

```
HRESULT DeleteFunctionBlock([in] BSTR functionBlockPath);
```

```
Sub DeleteFunctionBlock(functionBlockPath As String)
```

Description

Deletes the function block specified by **functionBlockPath** from the project.

Parameters

Name	Type	Description
functionBlockPath	String	Path to the function block to delete. Example: “MyLibrary.MyFBType.MyFB”

5.1.13.6 RenameFunctionBlock

```
HRESULT RenameFunctionBlock([in] BSTR functionBlockPath,  
                             [in] BSTR newFunctionBlockName);
```

```
Sub RenameFunctionBlock(functionBlockPath As String, _  
                        newFunctionBlockName As String)
```

Description

Renames the function block specified by **functionBlockPath** to **newFunctionBlockName**.

Parameters

Name	Type	Description
functionBlockPath	String	Path to the function block to rename. Example: “MyApplication.MyProgram.MyFB”
newFunctionBlockName	String	The new name of the selected function block. (Only name, not complete path)

5.1.14 Methods for ControlModuleType

5.1.14.1 ControlModuleType XML example

XML example:

```
<?xml version="1.0" encoding="UTF-16"?>
<ControlModuleType xmlns="CBOpenIFSchema3_0" Name="MyCMType" Protected="0" Hidden="0"
Scope="public" InteractionWindow="" AlarmOwner="0" BatchObject="" SILLevel="NonSIL"
SimulationMark="0">
  <CMPParameters>
    <CMPParameter Name="Par1" Type="dint" InitialValue="7" ReadPermission=""
WritePermission="" AuthenticationLevel="None" BatchProperty=""
TypePath="System.dint">
      <Description>Description of Par1</Description>
    </CMPParameter>
  </CMPParameters>
  <Variables>
    <Variable Name="Var1" Type="real" Attribute="retain" InitialValue="4.9"
ReadPermission="" WritePermission="" AuthenticationLevel="None" BatchProperty=""
TypePath="System.real">
      <Description>Description of Var1</Description>
    </Variable>
    <Variable Name="Calstruct" Type="CalendarStruct" Attribute="" InitialValue=""
ReadPermission="" WritePermission="" AuthenticationLevel="None" BatchProperty=""
TypePath="System.CalendarStruct"/>
  </Variables>
  <ExternalVariables/>
  <FunctionBlocks>
    <FunctionBlock Name="FbInst1" Type="AnotherFBType" TaskConnection=""
TypePath="MyApplication 1.0-0.AnotherFBType">
      <Description>Description of the instance</Description>
    </FunctionBlock>
  </FunctionBlocks>
  <ControlModules>
    <ControlModule Name="AnotherCMType1" Type="AnotherCMType" TaskConnection=""
VisibilityInGraphics="invisible" TypePath="MyApplication 1.0-0.AnotherCMType">
      <CMConnections/>
      <GraphPos Xpos="0.0" Ypos="0.0" Rotation="0.0" Xscale="1.0" Yscale="1.0"/>
    </ControlModule>
  </ControlModules>
  <CodeBlocks>
    <STCodeBlock Name="Code">
      <ST_Code>FbInst1(ParA := Par1 );</ST_Code>
    </STCodeBlock>
  </CodeBlocks>
  <Description>Description of MyCMType</Description>
  <GraphSize>
    <Point X="-1.0" Y="-1.0"/>
    <Point X="1.0" Y="1.0"/>
  </GraphSize>
</ControlModuleType>
```

GraphSize specifies the size of the coordinate system for the Control Module Type. The default, as used here, ranges from -1,-1 to 1,1 corresponding to the lower left corner and the right upper corner respectively. Origo, in the middle, will then have the coordinates 0,0.

For further information see the [XML Schema](#) and [semantics for omitted attributes and elements](#).

5.1.14.2 NewControlModuleType

```
HRESULT NewControlModuleType([in] BSTR controlModuleName,  
                             [in] BSTR appOrLibName,  
                             [in] BSTR controlModuleTypeContent,  
                             [out, retval] BSTR* messages);
```

```
Function NewControlModuleType(controlModuleName As String, _  
                             appOrLibName As String, _  
                             controlModuleTypeContent As String _  
                             ) As String
```

Description

Creates a new control module type with the name **controlModuleName**. The parameter **controlModuleTypeContent** is optional. It sets the content of the control module type using an XML-string. If it is empty, an empty control module type will be created. The “Name” attribute in the XML-string is ignored.

Parameters

Name	Type	Description
controlModuleName	String	The name of the control module type to create
appOrLibName	String	The name of the application or library the control module type will be created in.
controlModuleTypeContent	String	Optional parameter. A string containing an XML description of the new control module type. See control module type XML example . If this string is empty, an empty control module type will be created.

Return value

A [message bucket](#) is returned.

5.1.14.3 GetControlModuleType

```
HRESULT GetControlModuleType([in] BSTR controlModuleTypePath,  
                             [out, retval] BSTR*  
                             controlModuleTypeContent);
```

```
Function GetControlModuleType(controlModuleTypePath As String _  
                             ) As String
```

Description

Returns the content of the control module type, specified by **controlModuleTypePath**, as an XML-string.

Parameters

Name	Type	Description
controlModuleTypePath	String	Path to the control module type to retrieve information from. Example: "MyLibrary.MyCMType"

Return value

A string containing an XML description of the control module type is returned. See [control module type XML example](#).

5.1.14.4 SetControlModuleType

```
HRESULT SetControlModuleType([in] BSTR controlModuleTypePath,  
                             [in] BSTR controlModuleTypeContent,  
                             [out, retval] BSTR* messages);
```

```
Function SetControlModuleType(controlModuleTypePath As String, _  
                             controlModuleTypeContent As String _  
                             ) As String
```

Description

Sets the content of the control module type specified by **controlModuleTypePath** using an XML-string as the parameter **controlModuleTypeContent**. The “Name” attribute in the XML-string is ignored, i.e. you can’t change the name of the control module type using the XML.

Parameters

Name	Type	Description
controlModuleTypePath	String	Path to the control module type to set information for. Example: “MyApplication.MyCMType”
controlModuleTypeContent	String	The information to set for the selected control module type as an XML-string. See control module type XML example .

Return value

A [message bucket](#) is returned.

5.1.14.5 DeleteControlModuleType

```
HRESULT DeleteControlModuleType([in] BSTR controlModuleTypePath);
```

```
Sub DeleteControlModuleType(controlModuleTypePath As String)
```

Description

Deletes the control module type specified by **controlModuleTypePath** from the project. If there are instances of the type, the type can not be removed.

Parameters

Name	Type	Description
controlModuleTypePath	String	Path to the control module type to delete. Example: "MyLibrary.MyCMTType"

5.1.14.6 **RenameControlModuleType**

```
HRESULT RenameControlModuleType([in] BSTR controlModuleTypePath,  
                                [in] BSTR newControlModuleTypeName);  
  
Sub RenameControlModuleType(controlModuleTypePath As String,  
                             newControlModuleTypeName As String)
```

Description

Renames the control module type specified by **controlModuleTypePath** to **newControlModuleTypeName**. The type attribute of control modules of the renamed type will also be changed.

Parameters

Name	Type	Description
controlModuleTypePath	String	Path to the control module type to rename. Example: “MyLibrary.MyControlModuleType”
newControlModuleTypeName	String	The new name of the selected control module type. (Only name, not complete path)

5.1.15 Methods for ControlModule

5.1.15.1 Control Module XML example

Control modules exists in two different flavors:

1. (Ordinary) control modules.
2. Single control modules.

XML example for (ordinary) control module:

```
<?xml version="1.0" encoding="UTF-16"?>
<ControlModule xmlns="CBOpenIFSchema3_0" Name="MyControlModule1" Type="MyCMType"
TaskConnection="" VisibilityInGraphics="invisible" TypePath="MyApplication 1.0-
0.MyCMType">
  <CMConnections>
    <CMConnection Name="Par1" ActualParameter="GlobVar1" GraphicalConnection="0"/>
  </CMConnections>
  <GraphPos Xpos="0.0" Ypos="0.0" Rotation="0.0" Xscale="1.0" Yscale="1.0"/>
</ControlModule>
```

GraphPos specifies the placement, rotation and size of this control module in the surrounding control module or control module type.

For further information see the [XML Schema](#) and [semantics for omitted attributes and elements](#).

XML example for instance part of a single control module:

```
<?xml version="1.0" encoding="UTF-16"?>
<SingleControlModule xmlns="CBOpenIFSchema3_0" Name="SingleControlModule1"
TaskConnection="" VisibilityInGraphics="default" TypeGuid="8d28a131-973b-4ac3-ac80-
7c91477ab714" TypePath="MyApplication 1.0-0.MyCMType2.SingleControlModule1">
  <CMConnections>
    <CMConnection Name="Par1" ActualParameter="Par1" GraphicalConnection="0"/>
  </CMConnections>
  <GraphPos Xpos="0.0" Ypos="0.0" Rotation="0.0" Xscale="0.666667"
Yscale="0.666667"/>
</SingleControlModule>
```

For explanation of Single Control Modules see the chapter [Methods for SingleControlModules](#)

For further information about XML see the [XML Schema](#) and [semantics for omitted attributes and elements](#).

5.1.15.2 NewControlModule

```
HRESULT NewControlModule([in] BSTR controlModuleName,  
                          [in] BSTR controlModuleType,  
                          [in] BSTR pathToParent,  
                          [in] BSTR controlModuleContent,  
                          [out, retval] BSTR* messages);
```

```
Function NewControlModule(controlModuleName As String, _  
                          controlModuleType As String, _  
                          pathToParent As String, _  
                          controlModuleContent As String) As String
```

Description

Creates a new control module with the name **controlModuleName** and of the type **controlModuleType**. The parameter **controlModuleContent** is optional. It sets the content of the control module using an XML-string. If it is empty, an empty control module will be created. If content is provided the “Name” and “Type” attributes in the XML-string are ignored since the parameter **controlModuleName** is used as the control module name and the parameter **controlModuleType** is used as the control module type name. This method can not be used to create a new SingleControlModule. See method [NewSingleControlModule](#) instead.

Parameters

Name	Type	Description
controlModuleName	String	The name of the control module to create
controlModuleType	String	The type of control module type the created control module should be an instance of.
pathToParent	String	Path to the object where to put the new control module. Example: “MyLib.MyCMTType”
controlModuleContent	String	Optional parameter. A string containing an XML description of the new control module. See control module XML example . If this string is empty, an empty control module type will be created.

Return value

A [message bucket](#) is returned.

5.1.15.3 GetControlModule

```
HRESULT GetControlModule([in] BSTR controlModulePath,  
                          [out, retval] BSTR* controlModuleContent);
```

```
Function GetControlModule(controlModulePath As String) As String
```

Description

Returns the content of the control module (or the single control module) specified by **controlModulePath**, including connections, as an XML-string. Handles both single and standard control modules.

Parameters

Name	Type	Description
controlModulePath	String	Path to the control module (or single control module) to retrieve information from. Example: "MyApplication.MyCMType.MyCM" <i>Note! The path can contain several instance levels. Example: "MyApplic.MyCM1.CM2.CM4"</i>

Return value

A string containing an XML description of the control module(or single control module) is returned. See [control module type XML example](#) or [single control module XML example](#).

5.1.15.4 SetControlModule

```
HRESULT SetControlModule([in] BSTR controlModulePath,  
                          [in] BSTR controlModuleContent,  
                          [out, retval] BSTR* messages);
```

```
Function SetControlModule(controlModulePath As String, _  
                          controlModuleContent As String) As String
```

Description

Sets the content of the control module (or the single control module) specified by **controlModulePath**, including connections, using an XML-string as the parameter **controlModuleContent**. Handles both single and (ordinary) control modules. You can change the name and the type of the function block using the XML.

Parameters

Name	Type	Description
controlModulePath	String	Path to the control module to set information for. Example: "MyApplication.MyCMType.MyCM" <i>Note! The path can contain several instance levels. Example: "MyApplic.MyCM1.CM2.CM4"</i>
controlModuleContent	String	The information to set for the selected control module as an XML-string. See control module type XML example or single control module XML example .

Return value

A [message bucket](#) is returned.

5.1.15.5 DeleteControlModule

`HRESULT DeleteControlModule([in] BSTR controlModulePath);`

`Sub DeleteControlModule(controlModulePath As String)`

Description

Deletes the control module (or the single control module) specified by **controlModulePath** from the project. Handles both single and standard control modules.

Parameters

Name	Type	Description
controlModulePath	String	Path to the control module (or the single control module) to delete. Example: "MyApplication.MyCMType.MyCM"

5.1.15.6 RenameControlModule

```
HRESULT RenameControlModule([in] BSTR controlModulePath,  
                             [in] BSTR newControlModuleName);
```

```
Sub RenameControlModule(controlModulePath As String,  
                        newControlModuleName As String)
```

Description

Renames the control module specified by **controlModulePath** to **newControlModuleName**. This method also works for SingleControlModules.

Parameters

Name	Type	Description
controlModulePath	String	Path to the control module or single control module to rename. Example: "MyApplication.MyCM"
newControlModuleName	String	The new name of the selected control module. (Only name, not complete path)

5.1.15.7 GetApplicationControlModules

```
HRESULT GetApplicationControlModules([in] BSTR applicationName,  
                                     [out, retval] BSTR*  
applicationControlModulesContent);
```

```
Function GetApplicationControlModules(applicationName As String _  
                                     ) As String
```

Description

Returns a list (XML: <ControlModules>) of all the control modules (and single control modules) that reside directly in the application specified by **applicationName**, as an XML-string.

Parameters

Name	Type	Description
applicationName	String	Name of the application to retrieve controlmodules from. Example: "MyApplication"

Return value

A string containing an XML description of the <ControlModules> XML-list of control modules (or single control modules) is returned. See [control module type XML example](#) or [single control module XML example](#).

5.1.15.8 SetApplicationControlModules

```
HRESULT SetApplicationControlModules([in] BSTR applicationName,  
                                     [in] BSTR  
applicationControlModulesContent,  
                                     [out, retval] BSTR* messages);
```

```
Function SetApplicationControlModules(applicationName As String, _  
                                     applicationControlModulesContent As String  
_                                     ) As String
```

Description

Sets a list (XML: <ControlModules>) of all the control modules (and single control modules) that reside directly in the application specified by **applicationName** using the parameter **applicationControlModulesContent** as an XML-string.

Parameters

Name	Type	Description
applicationName	String	Path to the application to set control modules for. Example: "MyApplication"
applicationControlModulesContent	String	The control modules to set for the selected application as an XML-string in a <ControlModules> XML-list. See control module type XML example or single control module XML example for an example of contents in the list.

Return value

A [message bucket](#) is returned.

5.1.16 Methods for SingleControlModules

A Single Control Module (SCM) is represented in the CB as one composite object where the type and instance exists in the same object. To be able to handle the source code for Single Control Modules, OpenIF have to handle a Single Control Module as one instance part and one type part of the same object.

There are different methods for handling the instance and the type part of a Single Control Module. All Get and Set methods can handle complete XML for both instance and type part, but they will only use the part that they were intended for, i.e. the methods will throw away anything that they are not interested in. That path used by the different methods are the same for both type and instance part.

The following methods handle the instance part:

- **GetControlModule**
- **SetControlModule**
- **DeleteControlModule** (will delete the whole SCM)

The following methods handle the type part:

- **GetSingleControlModule**
- **SetSingleControlModule**
- **DeleteSingleControlModule** (will delete the whole SCM)

There is one method that handles the complete SCM. That is **NewSingleControlModule**. It is possible to send the complete XML for both instance and type part as one XML to **NewSingleControlModule** and it will save all of it.

For information about which part of the XML for a SCM that belong to the different parts of a SCM see [Complete XML](#), [InstanceXML](#) and [TypeXML](#) for a SCM.

When a Single Control Module is part of the XML for a parent it is only the instance part of the Single Control Module that is part of the parent XML. It is possible to add SCMs by adding the instance part to the XML for a parent. The type part for any new SCM instances will be created automatically (empty).

5.1.16.1 SingleControlModule XML example

XML example for complete Single Control Module (only used in NewSingleControlModule):

```
<?xml version="1.0" encoding="UTF-16"?>
<SingleControlModule xmlns="CBOpenIFSschema3_0" Name="SingleControlModule1"
InteractionWindow="" AlarmOwner="0" BatchObject="" TaskConnection=""
VisibilityInGraphics="default" TypeGuid="8d28a131-973b-4ac3-ac80-7c91477ab714"
SILLevel="NonSIL" SimulationMark="0">
  <CMPParameters>
    <CMPParameter Name="Par1" Type="dint" InitialValue="" ReadPermission=""
WritePermission="" AuthenticationLevel="None" BatchProperty=""
TypePath="System.dint"/>
  </CMPParameters>
  <Variables>
    <Variable Name="Var1" Type="real" Attribute="retain" InitialValue=""
ReadPermission="" WritePermission="" AuthenticationLevel="None" BatchProperty=""
TypePath="System.real"/>
  </Variables>
  <ExternalVariables>
    <ExternalVariable Name="ExtVar1" Type="dint" Attribute="constant"
ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.dint"/>
  </ExternalVariables>
  <FunctionBlocks/>
  <ControlModules>
    <ControlModule Name="MyCMType1" Type="MyCMType" TaskConnection=""
VisibilityInGraphics="invisible" TypePath="MyApplication 1.0-0.MyCMType">
      <CMInstGraphics>
        Invocation ( 0.0 , 0.0 , 0.0 , 1.0 , 1.0 Invisible_
        )
      </CMInstGraphics>
      <CMConnections>
        <CMConnection Name="Par1" ActualParameter="Par1" GraphicalConnection="0"/>
      </CMConnections>
      <GraphPos Xpos="0.0" Ypos="0.0" Rotation="0.0" Xscale="1.0" Yscale="1.0"/>
    </ControlModule>
  </ControlModules>
  <CodeBlocks>
    <STCodeBlock Name="Code">
      <ST_Code>if Par1 &gt;0 then Var1:=Var1+1; end_if;</ST_Code>
    </STCodeBlock>
  </CodeBlocks>
  <CMConnections>
    <CMConnection Name="Par1" ActualParameter="Par1" GraphicalConnection="0"/>
  </CMConnections>
  <GraphSize>
    <Point X="-1.0" Y="-1.0"/>
    <Point X="1.0" Y="1.0"/>
  </GraphSize>
  <GraphPos Xpos="0.0" Ypos="0.0" Rotation="0.0" Xscale="0.666667"
Yscale="0.666667"/>
</SingleControlModule>
```

XML example for type part of Single Control Module (only by GetSingleControlModule and SetSingleControlModule):

```
<?xml version="1.0" encoding="UTF-16"?>
<SingleControlModule xmlns="CBOpenIFSchema3_0" Name="SingleControlModule1"
InteractionWindow="" AlarmOwner="0" BatchObject="" TypeGuid="f4a4f744-3135-49b7-b250-
9db213dcbeb5" SILLevel="NonSIL" SimulationMark="0">
  <CMPParameters>
    <CMPParameter Name="Par1" Type="dint" InitialValue="" ReadPermission=""
WritePermission="" AuthenticationLevel="None" BatchProperty=""
TypePath="System.dint"/>
  </CMPParameters>
  <Variables>
    <Variable Name="Var1" Type="real" Attribute="retain" InitialValue=""
ReadPermission="" WritePermission="" AuthenticationLevel="None" BatchProperty=""
TypePath="System.real"/>
  </Variables>
  <ExternalVariables>
    <ExternalVariable Name="ExtVar1" Type="dint" Attribute="constant"
ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.dint"/>
  </ExternalVariables>
  <FunctionBlocks/>
  <ControlModules>
    <ControlModule Name="MyCMType1" Type="MyCMType" TaskConnection=""
VisibilityInGraphics="invisible" TypePath="MyApplication 1.0-0.MyCMType">
      <CMInstGraphics>
        Invocation ( 0.0 , 0.0 , 0.0 , 1.0 , 1.0 Invisible_
        )
      </CMInstGraphics>
      <CMConnections>
        <CMConnection Name="Par1" ActualParameter="Par1" GraphicalConnection="0"/>
      </CMConnections>
      <GraphPos Xpos="0.0" Ypos="0.0" Rotation="0.0" Xscale="1.0" Yscale="1.0"/>
    </ControlModule>
  </ControlModules>
  <CodeBlocks>
    <STCodeBlock Name="Code">
      <ST_Code>if Par1 &gt;0 then Var1:=Var1+1; end_if;</ST_Code>
    </STCodeBlock>
  </CodeBlocks>
  <GraphSize>
    <Point X="-1.0" Y="-1.0"/>
    <Point X="1.0" Y="1.0"/>
  </GraphSize>
</SingleControlModule>
```

For further information see the [XML Schema](#) and [semantics for omitted attributes and elements](#).

See also [What XML elements and attributes are needed by Set and New – methods?](#)

5.1.16.2 NewSingleControlModule

```
HRESULT NewSingleControlModule([in] BSTR controlModuleName,  
                               [in] BSTR pathToParent,  
                               [in] BSTR controlModuleContent,  
                               [out, retval] BSTR* messages);
```

```
Function NewSingleControlModule(controlModuleName As String, _  
                                pathToParent As String, _  
                                controlModuleContent As String _  
                                ) As String
```

Description

Creates a new single control module with the name **controlModuleName**. The parameter **controlModuleContent** is optional. It sets the content of the single control module using an XML-string. If it is empty, an empty single control module will be created. If content is provided the “Name” attribute in the XML-string is ignored since the parameter **controlModuleName** is used as the control module name.

Parameters

Name	Type	Description
controlModuleName	String	The name of the single control module to create
pathToParent	String	Path to the object where to put the new single control module.
controlModuleContent	String	Optional parameter. A string containing an XML description of the new single control module. See new single control module XML example . If this string is empty, an empty single control module type will be created

Return value

A [message bucket](#) is returned.

5.1.16.3 GetSingleControlModule

```
HRESULT GetSingleControlModule([in] BSTR singleControlModulePath,  
                               [out,retval] BSTR*  
                               singleControlModuleContent);
```

```
Function GetSingleControlModule(singleControlModulePath As String _  
                                ) As String
```

Description

Returns the content of the type part of the single control module, specified by **singleControlModulePath**, as an XML-string.

Parameters

Name	Type	Description
singleControlModulePath	String	Path to the single control module to retrieve information from. Example: "MyApplication.MySingleCM". <i>Note! The path can contain several single module levels. Example: "MyApplic.CMT1.SM1.SM2.SM3"</i>

Return value

A string containing an XML description of the instance is returned. [See type part of single control module XML example.](#)

5.1.16.4 SetSingleControlModule

```
HRESULT SetSingleControlModule([in] BSTR singleControlModulePath,  
                               [in] BSTR singleControlModuleContent,  
                               [out, retval] BSTR* messages);  
  
Function SetSingleControlModule(singleControlModulePath As String, _  
                               singleControlModuleContent As String  
—  
                               ) As String
```

Description

Sets the type part of the single control module specified by **singleControlModulePath** using an XML-string as the parameter **singleControlModuleContent**.

Parameters

Name	Type	Description
singleControlModulePath	String	Path to the single control module to set type information for. Example: "MyApplication.MyCMType.MySCM" <i>Note! The path can contain several single module levels. Example: "MyApplic.CMT1.SM1.SM2.SM3"</i>
singleControlModuleContent	String	The information to set for the selected single control module as an XML-string. See type part of single control module XML example .

Return value

A [message bucket](#) is returned.

5.1.16.5 DeleteSingleControlModule

```
HRESULT DeleteSingleControlModule([in] BSTR singleControlModulePath);
```

```
Sub DeleteSingleControlModule(singleControlModulePath As String)
```

Description

Deletes the single control module specified by **singleControlModulePath** from the project.

Parameters

Name	Type	Description
singleControlModulePath	String	Path to the single control module to delete. Example: “MyApplication.MyCMType.MySCM”

5.1.17 Using graphical connections in Control Module Types

It is possible to use so called *graphical connections* between Control Modules in a Control Module Type, by using the proper XML content. This will be described step by step below.

Before continuing the reader is encouraged to get familiar with the concept of graphical nodes and graphical connections by studying the Control Builder Help, Content->Control Module Diagram Editor->Working with Control Modules->Connect Control Modules.

The following subchapters will cover:

- Creating a Control Module Type with a graphical node
- Creating a Control Module Type with a graphical node, using AutoPoint attribute
- Creating a Control Module Type with instances of the previous Control Module Types and connecting the instances with a graphical connection
- Using and connecting pre-defined Control Modules in a cascade PID loop

Each subchapter will be presented as an XML example, comments to the XML content and a picture showing the result in the CMD editor.

There will be some discussions on graphical coordinates. You should then know that the default coordinate system for a Control Module Type ranges from -1,-1 to 1,1 corresponding to the lower left corner and the right upper corner respectively. Origo, in the middle, will then have the coordinates 0,0.

Also note that graphical objects that are not included in the XML content will be unchanged by the execution of the “Set” method. Graphical objects that are not module instances, graphical nodes or graphical connections will always be unchanged. Examples of such objects are lines, rectangles and text fields.

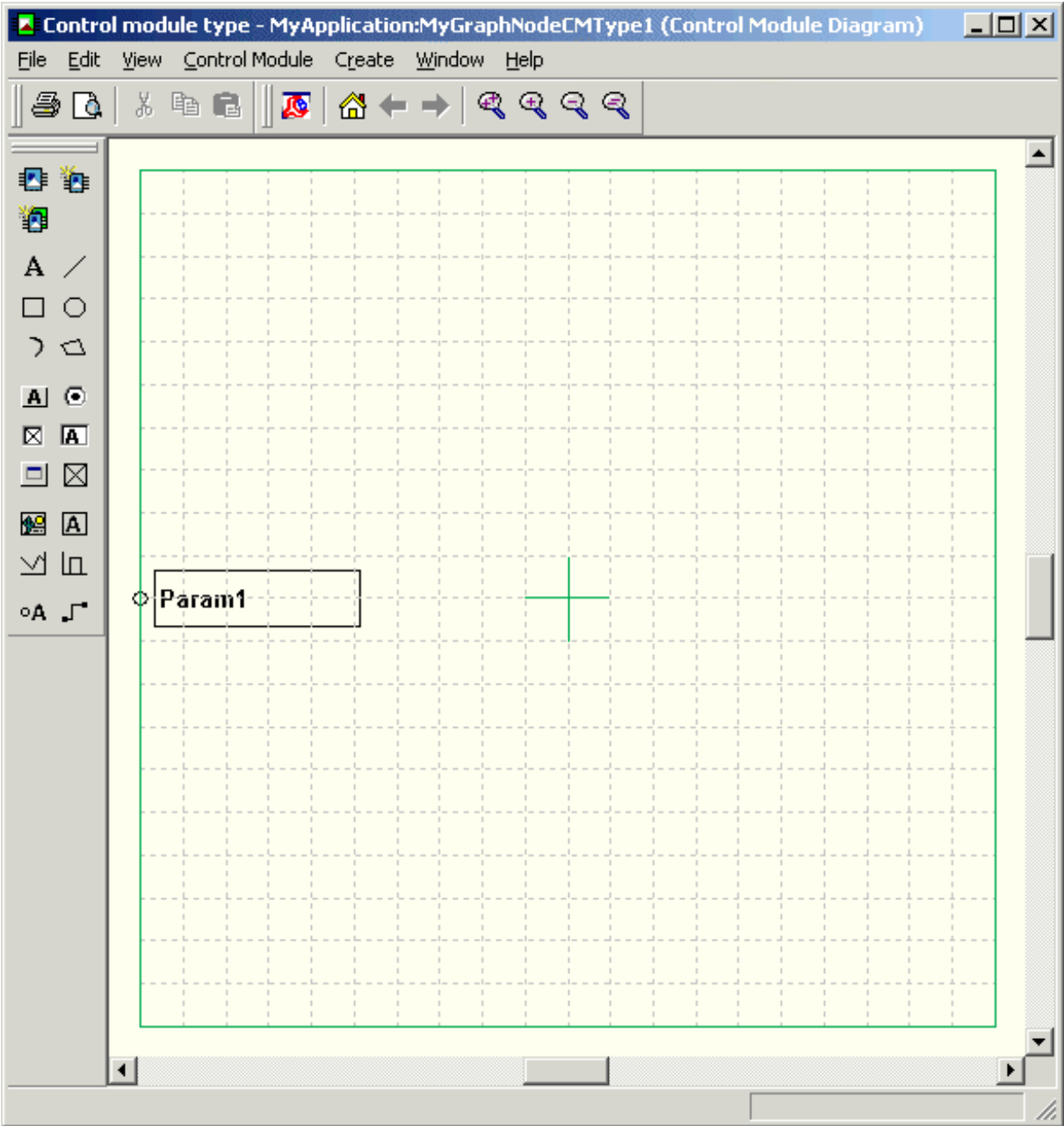
5.1.17.1 Creating a Control Module Type with a graphical node XML example

XML example:

```
<?xml version="1.0" encoding="UTF-16"?>
<ControlModuleType xmlns="CBOpenIFSchema3_0" Name="MyGraphNodeCMType1" Protected="0"
Hidden="0" Scope="public" InteractionWindow="" AlarmOwner="0" BatchObject=""
SILLevel="NonSIL" SimulationMark="0">
  <CMPParameters>
    <CMPParameter Name="Param1" Type="dint" InitialValue="" ReadPermission=""
WritePermission="" AuthenticationLevel="None" BatchProperty=""
TypePath="System.dint">
      <GraphNode Name="Param1" X="-1.0" Y="0.0"/>
    </CMPParameter>
  </CMPParameters>
  <Variables/>
  <ExternalVariables/>
  <FunctionBlocks/>
  <ControlModules/>
  <CodeBlocks>
    <ILCodeBlock Name="Code"/>
  </CodeBlocks>
  <GraphSize>
    <Point X="-1.0" Y="-1.0"/>
    <Point X="1.0" Y="1.0"/>
  </GraphSize>
</ControlModuleType>
```

Comments: There is one parameter called Param1. The XML part “<GraphNode Name=“Param1” X=“-1.0” Y=“0.0”/>” states that this parameter should be a graphical node placed in the coordinates -1, 0 which means the middle of the left side.

Shown in the CMD editor it looks like this:



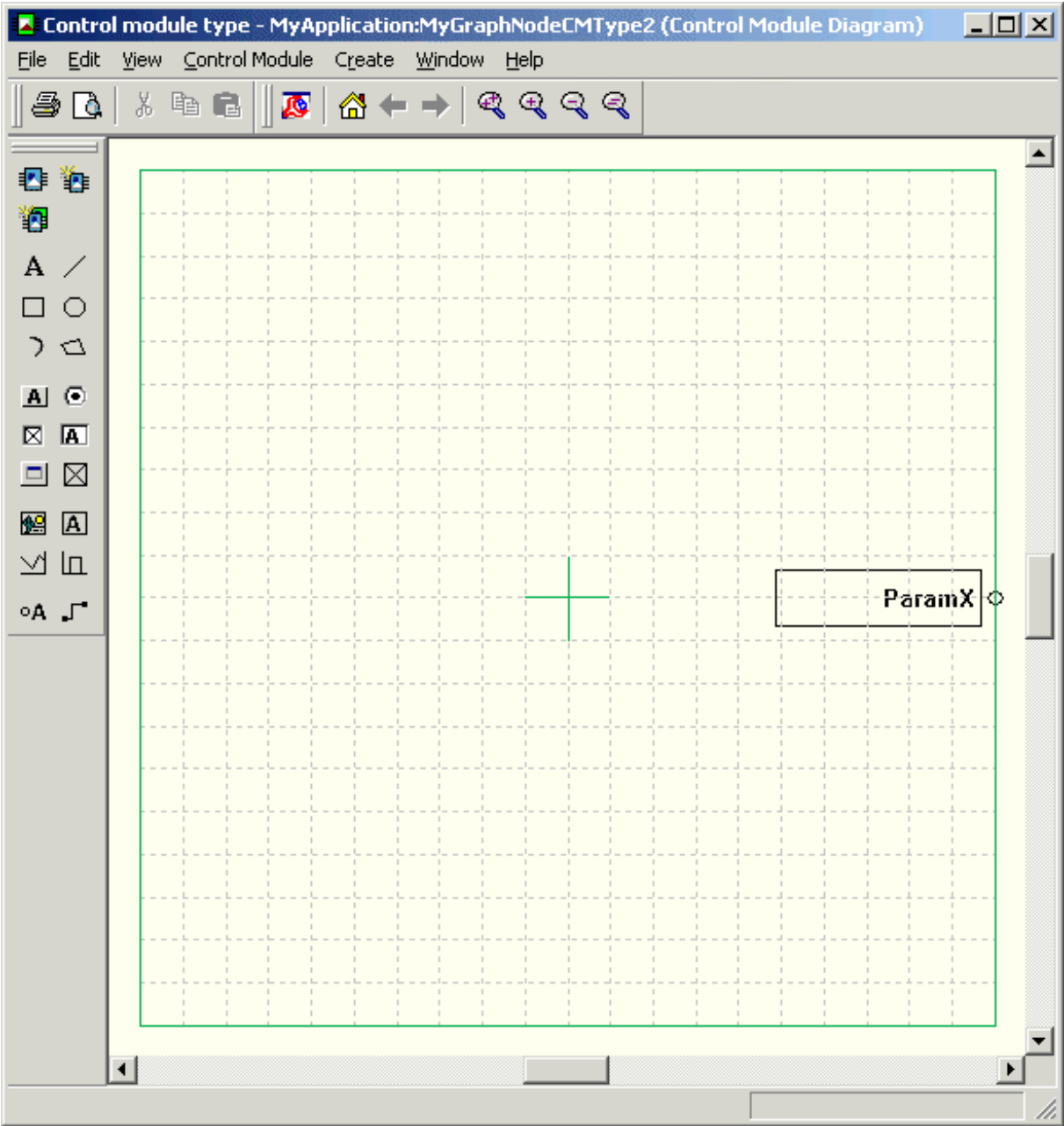
5.1.17.2 Creating a Control Module Type with a graphical node, using AutoPoint attribute, XML example

XML example:

```
<?xml version="1.0" encoding="utf-16"?>
<ControlModuleType xmlns="CBOpenIFSchema3_0" Name="MyGraphNodeCMType1" Protected="0"
Hidden="0" Scope="public" InteractionWindow="" AlarmOwner="0" BatchObject=""
SILLevel="NonSIL" SimulationMark="0">
  <CMPParameters>
    <CMPParameter Name="Param1" Type="dint" InitialValue="" ReadPermission=""
WritePermission="" AuthenticationLevel="None" BatchProperty=""
TypePath="System.dint">
      <AutoPoint AutoPos="Right"/>
    </CMPParameter>
  </CMPParameters>
  <Variables/>
  <ExternalVariables/>
  <FunctionBlocks/>
  <ControlModules/>
  <CodeBlocks>
    <ILCodeBlock Name="Code"/>
  </CodeBlocks>
  <GraphSize>
    <Point X="-1.0" Y="-1.0"/>
    <Point X="1.0" Y="1.0"/>
  </GraphSize>
</ControlModuleType>
```

Comments: There is one parameter called ParamX. The XML part “<AutoPoint AutoPos=“Right”/>” may be used instead of “GraphNode” in the previous example and states that this parameter should be a graphical node automatically placed in the “Right” position, which is some free position at the right side. The values **Left**, **Top** and **Bottom** may also be used.

Shown in the CMD editor it looks like this:



5.1.17.3 Creating a Control Module Type with a graphical connection

XML example:

```
<?xml version="1.0" encoding="UTF-16"?>
<ControlModuleType xmlns="CBOpenIFSschema3_0" Name="MyGraphConnCMType" Protected="0"
Hidden="0" Scope="public" InteractionWindow="" AlarmOwner="0" BatchObject=""
SILLevel="NonSIL" SimulationMark="0">
  <CMPParameters/>
  <Variables/>
  <ExternalVariables/>
  <FunctionBlocks/>
  <ControlModules>
    <ControlModule Name="MyGraphNodeCMType2_1"
Type="MyApplication.MyGraphNodeCMType2" TaskConnection=""
VisibilityInGraphics="default" TypePath="MyApplication 1.0-0.MyGraphNodeCMType2">
      <CMInstGraphics>Invocation ( -0.5 , 0.0 , 0.0 , 0.2 , 0.2 ) </CMInstGraphics>
      <CMConnections>
        <CMConnection Name="ParamX" ActualParameter="MyGraphNodeCMType1_1.Param1"
GraphicalConnection="1"/>
      </CMConnections>
      <GraphPos Xpos="-0.5" Ypos="0.0" Rotation="0.0" Xscale="0.2" Yscale="0.2"/>
    </ControlModule>
    <ControlModule Name="MyGraphNodeCMType1_1"
Type="MyApplication.MyGraphNodeCMType1" TaskConnection=""
VisibilityInGraphics="default" TypePath="MyApplication 1.0-0.MyGraphNodeCMType1">
      <CMInstGraphics>Invocation ( 0.5 , 0.0 , 0.0 , 0.2 , 0.2 ) </CMInstGraphics>
      <CMConnections>
        <CMConnection Name="Param1" ActualParameter="MyGraphNodeCMType2_1.ParamX"
GraphicalConnection="1"/>
      </CMConnections>
      <GraphPos Xpos="0.5" Ypos="0.0" Rotation="0.0" Xscale="0.2" Yscale="0.2"/>
    </ControlModule>
  </ControlModules>
  <CodeBlocks/>
  <GraphSize>
    <Point X="-1.0" Y="-1.0"/>
    <Point X="1.0" Y="1.0"/>
  </GraphSize>
</ControlModuleType>
```

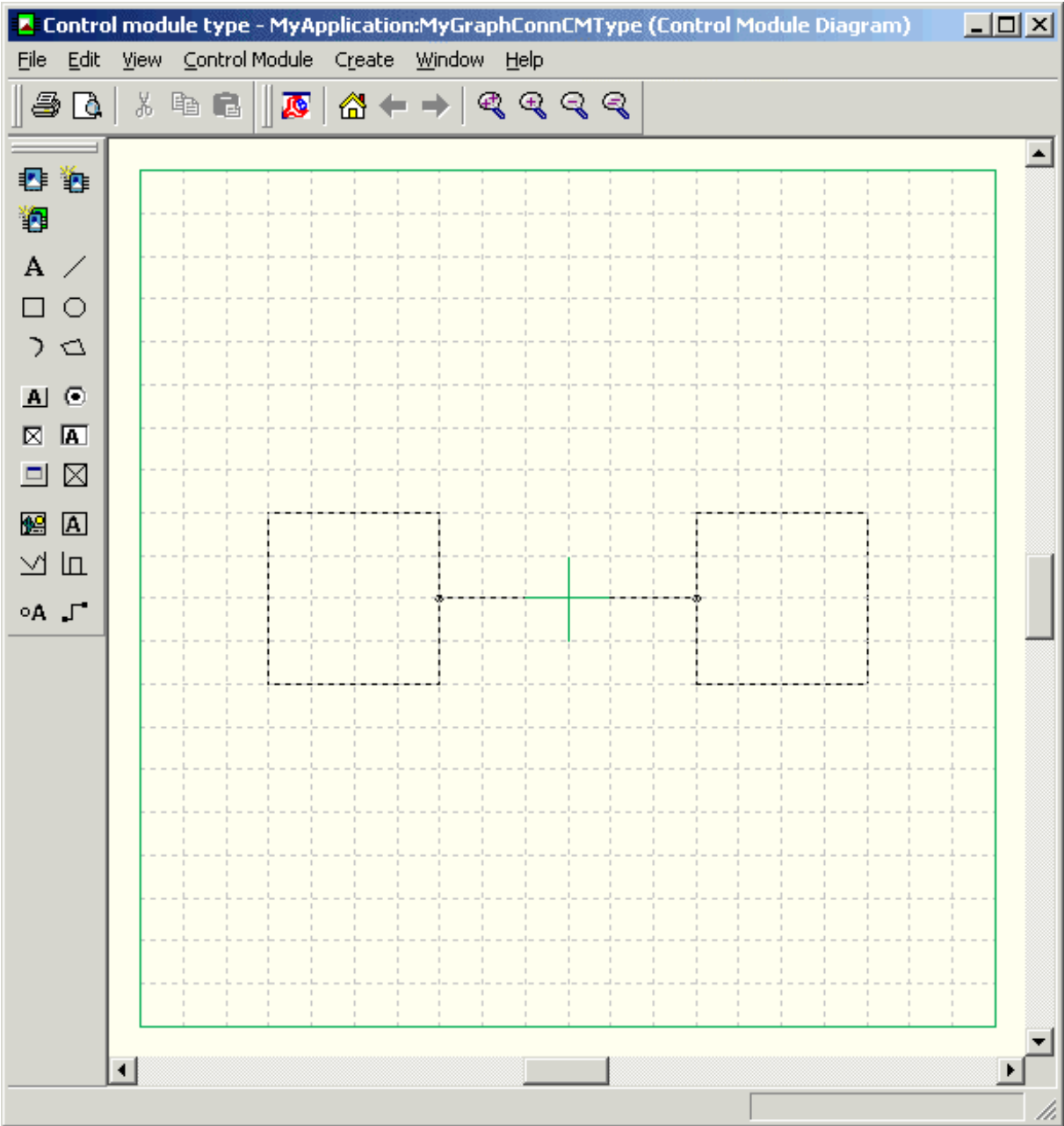
Comments: There is one instance of each of the previous types called **MyGraphNodeCMType2_1** and **MyGraphNodeCMType1_1** respectively. The XML part

```
<CMConnection Name="ParamX"
ActualParameter="MyGraphNodeCMType1_1.Param1"
GraphicalConnection="1"/>
```

says that parameter **ParamX** of **MyGraphNodeCMType2_1** should be connected to **Param1** of **MyGraphNodeCMType1_1** and that the connection is graphical.

For **MyGraphNodeCMType1_1** the reverse is true.

Shown in the CMD editor it looks like this:



5.1.17.4 Using and connecting pre-defined Control Modules in a cascade PID loop

XML example:

```
<?xml version="1.0" encoding="UTF-16"?>
<ControlModuleType xmlns="CBOpenIFSschema3_0" Name="MyControlCMType" Protected="0"
Hidden="0" Scope="public" InteractionWindow="" AlarmOwner="0" BatchObject=""
SILLevel="NonSIL" SimulationMark="0">
  <CMPParameters>
    <CMPParameter Name="Pres" Type="RealIO" InitialValue="" ReadPermission=""
WritePermission="" AuthenticationLevel="None" BatchProperty=""
TypePath="System.RealIO"/>
    <CMPParameter Name="Temp" Type="RealIO" InitialValue="" ReadPermission=""
WritePermission="" AuthenticationLevel="None" BatchProperty=""
TypePath="System.RealIO"/>
    <CMPParameter Name="Valve" Type="RealIO" InitialValue="" ReadPermission=""
WritePermission="" AuthenticationLevel="None" BatchProperty=""
TypePath="System.RealIO"/>
  </CMPParameters>
  <Variables/>
  <ExternalVariables/>
  <FunctionBlocks/>
  <ControlModules>
    <ControlModule Name="AnalogInCC1" Type="ControlStandardLib.AnalogInCC"
TaskConnection="" VisibilityInGraphics="default" TypePath="ControlStandardLib 1.0-
0.AnalogInCC">
      <CMConnections>
        <CMConnection Name="Name" ActualParameter="" GraphicalConnection="0"/>
        <CMConnection Name="Description" ActualParameter="" GraphicalConnection="0"/>
        <CMConnection Name="AnalogInput" ActualParameter="Temp"
GraphicalConnection="0"/>
        <CMConnection Name="Out" ActualParameter="" GraphicalConnection="0"/>
        <CMConnection Name="RedIncDecLim" ActualParameter=""
GraphicalConnection="0"/>
        <CMConnection Name="Level6Connection" ActualParameter=""
GraphicalConnection="0"/>
        <CMConnection Name="InteractionPar" ActualParameter=""
GraphicalConnection="0"/>
        <CMConnection Name="DisplayBars" ActualParameter="" GraphicalConnection="0"/>
      </CMConnections>
      <GraphPos Xpos="-0.7" Ypos="0.4" Rotation="0.0" Xscale="0.1" Yscale="0.1"/>
    </ControlModule>
    <ControlModule Name="AnalogInCC2" Type="ControlStandardLib.AnalogInCC"
TaskConnection="" VisibilityInGraphics="default" TypePath="ControlStandardLib 1.0-
0.AnalogInCC">
      <CMConnections>
        <CMConnection Name="Name" ActualParameter="" GraphicalConnection="0"/>
        <CMConnection Name="Description" ActualParameter="" GraphicalConnection="0"/>
        <CMConnection Name="AnalogInput" ActualParameter="Pres"
GraphicalConnection="0"/>
        <CMConnection Name="Out" ActualParameter="" GraphicalConnection="0"/>
        <CMConnection Name="RedIncDecLim" ActualParameter=""
GraphicalConnection="0"/>
        <CMConnection Name="Level6Connection" ActualParameter=""
GraphicalConnection="0"/>
        <CMConnection Name="InteractionPar" ActualParameter=""
GraphicalConnection="0"/>
        <CMConnection Name="DisplayBars" ActualParameter="" GraphicalConnection="0"/>
      </CMConnections>
      <GraphPos Xpos="-0.7" Ypos="0.0" Rotation="0.0" Xscale="0.1" Yscale="0.1"/>
    </ControlModule>
    <ControlModule Name="PidCC1" Type="ControlStandardLib.PidCC" TaskConnection=""
VisibilityInGraphics="default" TypePath="ControlStandardLib 1.0-0.PidCC">
```



```

    <CMConnections>
      <CMConnection Name="IconName" ActualParameter="" GraphicalConnection="0"/>
      <CMConnection Name="Name" ActualParameter="&apos;Temp&apos;"
GraphicalConnection="0"/>
      <CMConnection Name="Description" ActualParameter="" GraphicalConnection="0"/>
      <CMConnection Name="Pv" ActualParameter="AnalogInCC1.Out"
GraphicalConnection="0"/>
      <CMConnection Name="Sp" ActualParameter="" GraphicalConnection="0"/>
      <CMConnection Name="SpExternalInit" ActualParameter=""
GraphicalConnection="0"/>
      <CMConnection Name="SpManValueInit" ActualParameter=""
GraphicalConnection="0"/>
      <CMConnection Name="Feedforward" ActualParameter="" GraphicalConnection="0"/>
      <CMConnection Name="Out" ActualParameter="" GraphicalConnection="0"/>
      <CMConnection Name="AutoModeInit" ActualParameter=""
GraphicalConnection="0"/>
      <CMConnection Name="OutValueInit" ActualParameter=""
GraphicalConnection="0"/>
      <CMConnection Name="Track" ActualParameter="" GraphicalConnection="0"/>
      <CMConnection Name="TrackValue" ActualParameter="" GraphicalConnection="0"/>
      <CMConnection Name="AEConfig" ActualParameter="" GraphicalConnection="0"/>
      <CMConnection Name="AESeverity" ActualParameter="" GraphicalConnection="0"/>
      <CMConnection Name="AECClass" ActualParameter="" GraphicalConnection="0"/>
      <CMConnection Name="GTDevPos" ActualParameter="" GraphicalConnection="0"/>
      <CMConnection Name="AlStateDevPos" ActualParameter=""
GraphicalConnection="0"/>
      <CMConnection Name="GTDevNeg" ActualParameter="" GraphicalConnection="0"/>
      <CMConnection Name="AlStateDevNeg" ActualParameter=""
GraphicalConnection="0"/>
      <CMConnection Name="ATWarning" ActualParameter="" GraphicalConnection="0"/>
      <CMConnection Name="EnableSupOut" ActualParameter=""
GraphicalConnection="0"/>
      <CMConnection Name="LevelPar" ActualParameter="" GraphicalConnection="0"/>
      <CMConnection Name="InteractionPar" ActualParameter=""
GraphicalConnection="0"/>
    </CMConnections>
    <GraphPos Xpos="-0.2" Ypos="0.44" Rotation="0.0" Xscale="0.1" Yscale="0.1"/>
  </ControlModule>
  <ControlModule Name="PidCC2" Type="ControlStandardLib.PidCC" TaskConnection=""
VisibilityInGraphics="default" TypePath="ControlStandardLib 1.0-0.PidCC">
    <CMConnections>
      <CMConnection Name="IconName" ActualParameter="" GraphicalConnection="0"/>
      <CMConnection Name="Name" ActualParameter="&apos;Pres&apos;"
GraphicalConnection="0"/>
      <CMConnection Name="Description" ActualParameter="" GraphicalConnection="0"/>
      <CMConnection Name="Pv" ActualParameter="AnalogInCC2.Out"
GraphicalConnection="0"/>
      <CMConnection Name="Sp" ActualParameter="PidCC1.Out"
GraphicalConnection="0"/>
      <CMConnection Name="SpExternalInit" ActualParameter=""
GraphicalConnection="0"/>
      <CMConnection Name="SpManValueInit" ActualParameter=""
GraphicalConnection="0"/>
      <CMConnection Name="Feedforward" ActualParameter="" GraphicalConnection="0"/>
      <CMConnection Name="Out" ActualParameter="" GraphicalConnection="0"/>
      <CMConnection Name="AutoModeInit" ActualParameter=""
GraphicalConnection="0"/>
      <CMConnection Name="OutValueInit" ActualParameter=""
GraphicalConnection="0"/>
      <CMConnection Name="Track" ActualParameter="" GraphicalConnection="0"/>
      <CMConnection Name="TrackValue" ActualParameter="" GraphicalConnection="0"/>
      <CMConnection Name="AEConfig" ActualParameter="" GraphicalConnection="0"/>
      <CMConnection Name="AESeverity" ActualParameter="" GraphicalConnection="0"/>
      <CMConnection Name="AECClass" ActualParameter="" GraphicalConnection="0"/>

```

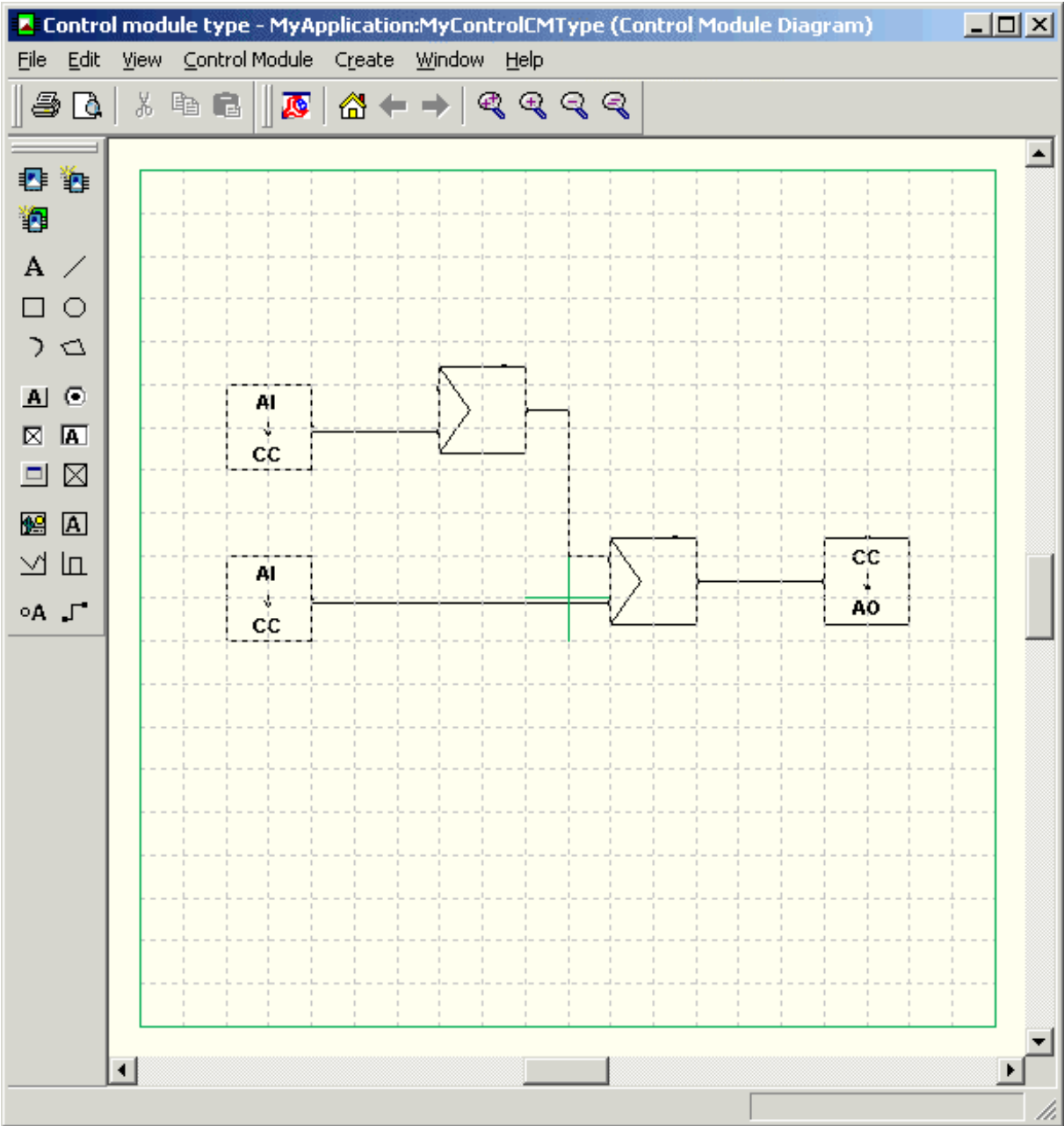
```

        <CMConnection Name="GTDevPos" ActualParameter="" GraphicalConnection="0"/>
        <CMConnection Name="AlStateDevPos" ActualParameter=""
GraphicalConnection="0"/>
        <CMConnection Name="GTDevNeg" ActualParameter="" GraphicalConnection="0"/>
        <CMConnection Name="AlStateDevNeg" ActualParameter=""
GraphicalConnection="0"/>
        <CMConnection Name="ATWarning" ActualParameter="" GraphicalConnection="0"/>
        <CMConnection Name="EnableSupOut" ActualParameter=""
GraphicalConnection="0"/>
        <CMConnection Name="LevelPar" ActualParameter="" GraphicalConnection="0"/>
        <CMConnection Name="InteractionPar" ActualParameter=""
GraphicalConnection="0"/>
    </CMConnections>
    <GraphPos Xpos="0.2" Ypos="0.04" Rotation="0.0" Xscale="0.1" Yscale="0.1"/>
</ControlModule>
    <ControlModule Name="AnalogOutCC1" Type="ControlStandardLib.AnalogOutCC"
TaskConnection="" VisibilityInGraphics="default" TypePath="ControlStandardLib 1.0-
0.AnalogOutCC">
    <CMConnections>
        <CMConnection Name="Name" ActualParameter="" GraphicalConnection="0"/>
        <CMConnection Name="Description" ActualParameter="" GraphicalConnection="0"/>
        <CMConnection Name="In" ActualParameter="PidCC2.Out"
GraphicalConnection="0"/>
        <CMConnection Name="StictionComp" ActualParameter=""
GraphicalConnection="0"/>
        <CMConnection Name="AnalogOutput" ActualParameter="Valve"
GraphicalConnection="0"/>
        <CMConnection Name="Level6Connection" ActualParameter=""
GraphicalConnection="0"/>
        <CMConnection Name="InteractionPar" ActualParameter=""
GraphicalConnection="0"/>
        <CMConnection Name="DisplayBars" ActualParameter="" GraphicalConnection="0"/>
    </CMConnections>
    <GraphPos Xpos="0.7" Ypos="0.04" Rotation="0.0" Xscale="0.1" Yscale="0.1"/>
</ControlModule>
</ControlModules>
<CodeBlocks>
    <STCodeBlock Name="Code">
        <ST_Code></ST_Code>
    </STCodeBlock>
</CodeBlocks>
<GraphSize>
    <Point X="-1.0" Y="-1.0"/>
    <Point X="1.0" Y="1.0"/>
</GraphSize>
</ControlModuleType>

```

Comments: To get this example working it is necessary to have the library controlstandardlib loaded.

Shown in the CMD editor it looks like this:



Important note!

It may be difficult or undesirable to calculate appropriate coordinates for the Control Module instances.

However, it is essential for graphical connections to work correctly that the Control Module instances are placed on different coordinates with enough spacing.

The reason is that the graphical connections are dependent on their graphical positions to be uniquely identified.

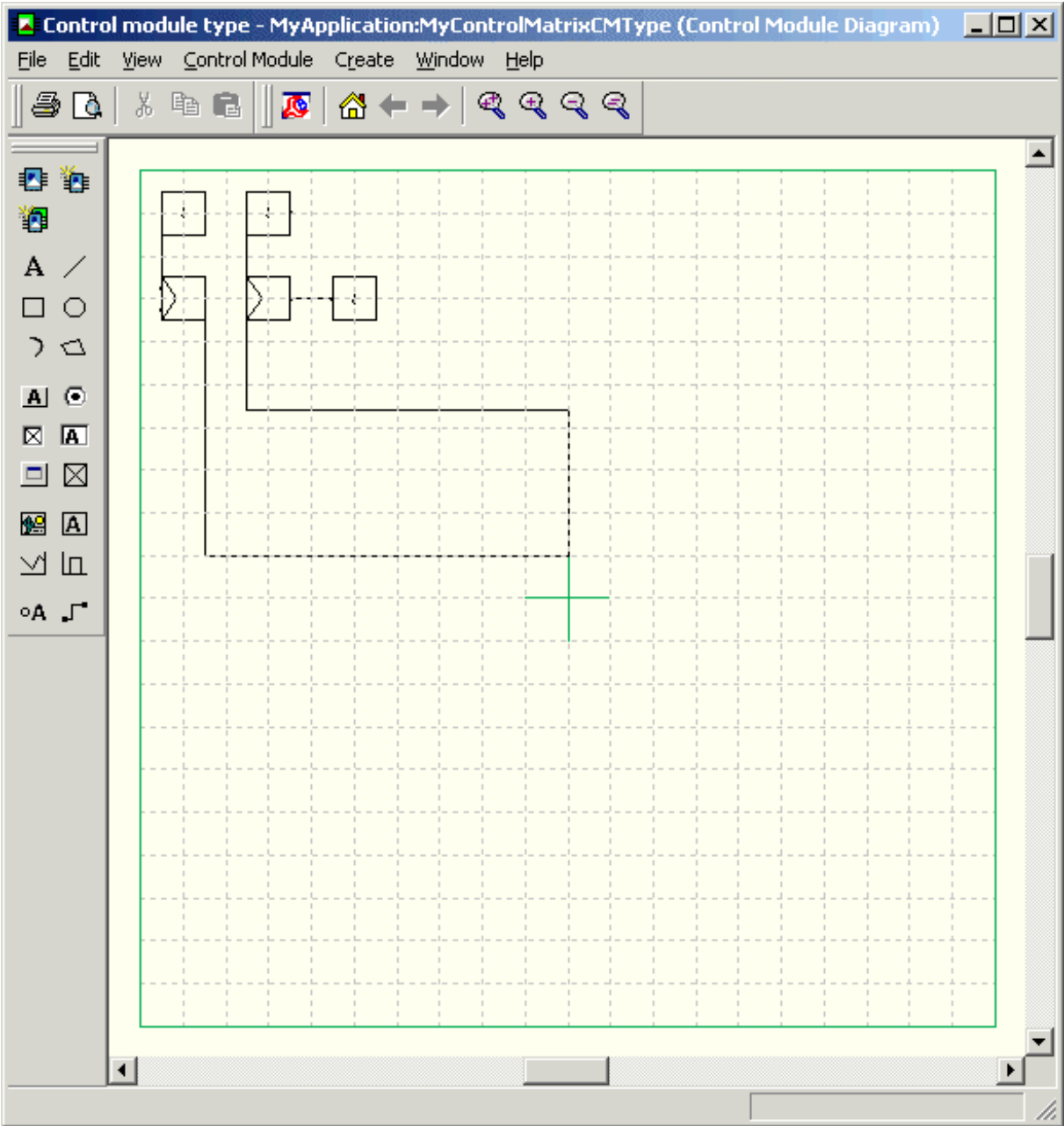
To overcome this problem we suggest the following solution:

Place the instances in a matrix and add an incremental x- or y-value of 0.2 for each new instance. Start near the upper left corner. Use a scale factor of 0.05. The matrix could then handle up to 100 instances.

Example: Using this approach the modules in this PID loop example could have the following attributes as shown in the XML:

```
<GraphPos Xpos="-0.9" Ypos="0.9" Rotation="0.0" Xscale="0.05" Yscale="0.05"/>
<GraphPos Xpos="-0.7" Ypos="0.9" Rotation="0.0" Xscale="0.05" Yscale="0.05"/>
<GraphPos Xpos="-0.9" Ypos="0.7" Rotation="0.0" Xscale="0.05" Yscale="0.05"/>
<GraphPos Xpos="-0.7" Ypos="0.7" Rotation="0.0" Xscale="0.05" Yscale="0.05"/>
<GraphPos Xpos="-0.5" Ypos="0.7" Rotation="0.0" Xscale="0.05" Yscale="0.05"/>
```

Shown in the CMD editor this would look like this:



5.1.18 Methods for DiagramTypes

5.1.18.1 DiagramType XML example

XML example:

```
<?xml version="1.0" encoding="UTF-16"?>
<DiagramType xmlns="CBOpenIFSchema3_0" Name="MyDiagramType" Protected="0" Hidden="0"
Scope="public" AlarmOwner="1" InstantiateAsAspectObject="1" SILLevel="NonSIL"
RestrictedSIL="0" SimulationMark="0" EmbeddedGraphicsVisible="0">
  <Parameters>
    <Parameter Name="Par1" Type="dint" TypePath="System.dint" AccessLevel="Read Only"
SafetyType="None" Attribute="" InitialValue="" ReadPermission="" WritePermission=""
AuthenticationLevel="None" Direction="in" FDPort="yes">
      <Description>Description of Par1</Description>
    </Parameter>
    <Parameter Name="Par2" Type="bool" TypePath="System.bool" AccessLevel="Read Only"
SafetyType="None" Attribute="" InitialValue="" ReadPermission="" WritePermission=""
AuthenticationLevel="None" Direction="in" FDPort="yes">
      <Description>Description of Par2</Description>
    </Parameter>
  </Parameters>
  <Variables>
    <Variable Name="Var1" Type="dint" TypePath="System.dint" AccessLevel="Read Only"
SafetyType="None" Attribute="retain" InitialValue="" ReadPermission=""
WritePermission="" AuthenticationLevel="None">
      <Description>Description of Var1</Description>
    </Variable>
    <Variable Name="Var2" Type="bool" TypePath="System.bool" AccessLevel="Read Only"
SafetyType="None" Attribute="retain" InitialValue="" ReadPermission=""
WritePermission="" AuthenticationLevel="None">
      <Description>Description of Var2</Description>
    </Variable>
  </Variables>
  <FunctionBlocks>
    <FunctionBlock Name="MyFBType_1" Type="Appl.MyFBType" TypePath="Appl.MyFBType"
AccessLevel="Read Only" SafetyType="None" TaskConnection="" AspectObject="1"
ExposePropertiesInParent="0"/>
  </FunctionBlocks>
  <ControlModules>
    <ControlModule Name="MyControlModuleType_1" Type="Appl.MyControlModuleType"
TypePath="Appl.MyControlModuleType" AccessLevel="Read Only" SafetyType="None"
TaskConnection="" AspectObject="1" ExposePropertiesInParent="0"
VisibilityInGraphics="default">
      <CMInstGraphics>Invocation ( 0.0 , 0.0 , 0.0 , 1.0 , 1.0 ) </CMInstGraphics>
      <CMConnections/>
      <GraphPos Xpos="0.0" Ypos="0.0" Rotation="0.0" Xscale="1.0" Yscale="1.0"/>
    </ControlModule>
  </ControlModules>
  <DiagramInstances>
    <DiagramInstance Name="DiagramType2_1" Type="Appl.DiagramType2"
TypePath="Appl.DiagramType2" AccessLevel="Read Only" SafetyType="None"
AspectObject="1" ExposePropertiesInParent="0"/>
  </DiagramInstances>
  <CodeBlocks>
    <FDCodeBlock Name="Code">
      <FunctionDiagram>
        <Pages>
          <Layout FormatName="A4" Orientation="Portrait" Unit="0.01 mm" Width="21000"
Height="29700" Top="1500" Bottom="1500" Left="1500" Right="1500"/>
          <Page PageNo="1" PageName="">
            <Blocks>
```

```

        <Invocation Name="and(1)" DataFlowOrder="2" ENPort="false"
Id="be0c6b41-0437-4693-8e53-aa0376b3904c">
        <Description></Description>
        <Layout X="5080" Y="4064" Width="677" Height="1354">
        <Ports>
        <Port Name="IN1" Visible="true"/>
        <Port Name="IN2" Visible="true"/>
        <Port Name="" Visible="true"/>
        </Ports>
        </Layout>
        </Invocation>
        <DataRef Name="Var2(1)" Id="375da18b-ab6f-446c-96f0-7d30cf978531">
        <Description></Description>
        <Layout X="11853" Y="4402" Width="846" Height="338"></Layout>
        </DataRef>
        <Invocation Name="MyControlModuleType_1" DataFlowOrder="1"
ENPort="false" Id="4a37d562-2f14-4a4e-9fee-4de7ace938e8">
        <Description></Description>
        <Layout X="4572" Y="16933" Width="27093" Height="677"></Layout>
        </Invocation>
        <Invocation Name="MyFBType_1" DataFlowOrder="3" ENPort="false"
Id="824b4a0d-8959-467c-87cc-d92b54806dfe">
        <Description></Description>
        <Layout X="8974" Y="16933" Width="1524" Height="677"></Layout>
        </Invocation>
        <Invocation Name="DiagramType2_1" DataFlowOrder="4" ENPort="false"
Id="e2e6cc11-b31d-478e-8a6e-0554377b5b6b">
        <Description></Description>
        <Layout X="13546" Y="1524" Width="1862" Height="677"></Layout>
        </Invocation>
    </Blocks>
    <DataConnections>
        <DataConnection Src="and(1)" Dest="Var2(1)" Id="32d5b69b-0ddd-4999-
9e10-3c26ee168033">
        <Layout>
        <SrcPos X="5969" Y="47413"/>
        <LinePos DX="-169" DY="0"/>
        <LinePos DX="0" DY="-169"/>
        <DestPos X="11641" Y="4572"/>
        </Layout>
        </DataConnection>
        <DataConnection Src="Var2" SrcDirect="true" Dest="and(1).IN1"
Id="fe9e1b59-9451-4ba8-82c0-a671f6ecf81f"/>
        <DataConnection Src="Par2" SrcDirect="true" Dest="and(1).IN2"
Id="ebc9eb17-742d-4d89-b24f-22138f8406ee"/>
    </DataConnections>
</Page>
</Pages>
</FunctionDiagram>
</FDCodeBlock>
</CodeBlocks>
<Description/>
</DiagramType>

```

For further information see the [XML Schema](#) and [semantics for omitted attributes and elements](#).

See also [What XML elements and attributes are needed by Set and New – methods?](#)

5.1.18.2 NewDiagramType

```
HRESULT NewDiagramType([in] BSTR diagramTypeName,  
                        [in] BSTR appOrLibName,  
                        [in] BSTR diagramTypeContent,  
                        [out, retval] BSTR* messages);
```

```
Function NewDiagramType(diagramTypeName As String, _  
                        appOrLibName As String, _  
                        diagramTypeContent As String _  
                        ) As String
```

Description

Creates a new diagram type with the name **diagramTypeName**. The parameter **diagramTypeContent** is an optional parameter. It sets the content of the diagram type using an XML-string. If it is empty, an empty diagram type will be created. If content is provided the “Name” attribute in the XML-string is ignored since the parameter **diagramTypeName** is used as the diagram type name.

Parameters

Name	Type	Description
diagramTypeName	String	The name of the diagram type to create.
appOrLibName	String	The name of the application or library the diagram type will be created in.
diagramTypeContent	String	Optional parameter. A string containing an XML description of the new diagram type. See diagram type XML example . If this string is empty, an empty diagram type will be created.

Return value

A [message bucket](#) is returned.

5.1.18.3 **GetDiagramType**

```
HRESULT GetDiagramType([in] BSTR diagramTypePath,  
                        [out, retval] BSTR* diagramTypeContent);
```

```
Function GetDiagramType(diagramTypePath As String) As String
```

Description

Returns the content of the diagram type, specified by **diagramTypePath**, as an XML-string.

Parameters

Name	Type	Description
diagramTypePath	String	Path to the diagram type to retrieve information from. Example: “MyLibrary.MyDiagramType”

Return value

A string containing an XML description of the data type is returned. See [diagram type XML example](#).

5.1.18.4 SetDiagramType

```
HRESULT SetDiagramType([in] BSTR diagramTypePath,  
                        [in] BSTR diagramTypeContent,  
                        [out, retval] BSTR* messages);
```

```
Function SetDiagramType(diagramTypePath As String, _  
                        diagramTypeContent As String _  
                        ) As String
```

Description

Sets the content of the diagram type specified by **diagramTypePath** using an XML-string as the parameter **diagramTypeContent**. The “Name” attribute in the XML-string is ignored, i.e. you can’t change the name of the diagram type using the XML.

Parameters

Name	Type	Description
diagramTypePath	String	Path to the diagram type to set information for. Example: “MyLibrary.MyDiagramType”
diagramTypeContent	String	The information to set for the selected diagram type as an XML-string. See diagram type XML example

Return value

A [message bucket](#) is returned.

5.1.18.5 DeleteDiagramType

```
HRESULT DeleteDiagramType([in] BSTR diagramTypePath);
```

```
Sub DeleteDiagramType(diagramTypePath As String)
```

Description

Deletes the diagram type specified by **diagramTypePath** from the project. If there are instances of the type, the type can not be removed.

Parameters

Name	Type	Description
diagramTypePath	String	Path to the diagram type to delete. Example: "MyLibrary.MyDiagramType"

5.1.18.6 **RenameDiagramType**

```
HRESULT RenameDiagramType([in] BSTR diagramTypePath,  
                           [in] BSTR newDiagramTypeName);
```

```
Sub RenameDiagramType(diagramTypePath As String, _  
                      newDiagramTypeName As String)
```

Description

Renames the diagram type specified by **diagramTypePath** to **newDiagramTypeName**. The type attribute of diagram instances of the renamed type will also be changed.

Parameters

Name	Type	Description
diagramTypePath	String	Path to the diagram type to rename. Example: “MyLibrary.MyDiagramType”
newDiagramTypeName	String	The new name of the selected diagram type. (Only name, not complete path)

5.1.19 Methods for DiagramInstances

5.1.19.1 DiagramInstance XML example

XML example:

```
<?xml version="1.0" encoding="UTF-16"?>
<DiagramInstance xmlns="CBOpenIFSchema3_0" Name="MyDiagramType_1"
Type="Appl.MyDiagramType" TypePath="Appl.MyDiagramType" AccessLevel="Read Only"
SafetyType="None" AspectObject="1" ExposePropertiesInParent="0">
  <Description>Diagram instance description</Description>
</DiagramInstance>
```

For further information see the [XML Schema](#) and [semantics for omitted attributes and elements](#).

See also [What XML elements and attributes are needed by Set and New – methods?](#)

5.1.19.2 NewDiagramInstance

```
HRESULT NewDiagramInstance([in] BSTR diagramInstanceName,  
                           [in] BSTR diagramType,  
                           [in] BSTR pathToParent,  
                           [in] BSTR diagramInstanceContent,  
                           [out, retval] BSTR* messages);
```

```
Function NewDiagramInstance(diagramInstanceName As String, _  
                           diagramType As String, _  
                           pathToParent As String, _  
                           diagramInstanceContent As String _  
                           ) As String
```

Description

Creates a new diagram instance with the name **diagramInstanceName** and of the type **diagramType**. The parameter **diagramInstanceContent** is optional. It sets the content of the diagram instance using an XML-string. If it is empty, an empty diagram instance will be created. If content is provided the “Name” and “Type” attributes in the XML-string are ignored since the parameter **diagramInstanceName** is used as the diagram instance name and the parameter **diagramType** is used as the diagram type name. Diagram instances can only be created in Diagram Types or in Diagrams.

Parameters

Name	Type	Description
diagramInstanceName	String	The name of the diagram instance to create.
diagramType	String	The diagram type the created diagram instance should be an instance of.
pathToParent	String	Path to the object where to put the new diagram instance. Example: “MyLib.MyDiagramType”
diagramInstanceContent	String	Optional parameter. A string containing an XML description of the new diagram instance. See diagram instance XML example . If this string is empty, an empty diagram instance will be created.

Return value

A [message bucket](#) is returned.

5.1.19.3 GetDiagramInstance

```
HRESULT GetDiagramInstance([in] BSTR diagramInstancePath,  
                           [out, retval] BSTR*  
diagramInstanceContent);
```

```
Function GetDiagramInstance(diagramInstancePath As String) As String
```

Description

Returns the content of the diagram instance, specified by **diagramInstancePath**, as an XML-string.

Parameters

Name	Type	Description
diagramInstancePath	String	Path to the diagram instance to retrieve information from. Example: "MyLibrary.MyDiagramType.MyDI". <i>Note! The path can contain several instance levels.</i> <i>Example:</i> <i>"MyApplic.Diagram1.DiagInst1.DiagInst2.DiagInst3"</i>

Return value

A string containing an XML description of the instance is returned. See [diagram instance XML example](#).

5.1.19.4 SetDiagramInstance

```
HRESULT SetDiagramInstance([in] BSTR diagramInstancePath,  
                           [in] BSTR diagramInstanceContent,  
                           [out, retval] BSTR* messages);
```

```
Function SetDiagramInstance(diagramInstancePath As String, _  
                           diagramInstanceContent As String) As  
String
```

Description

Sets the content of the diagram instance specified by **diagramInstancePath** using an XML-string as the parameter **diagramInstanceContent**. You can change the name and the type of the diagram instance using the XML.

Parameters

Name	Type	Description
diagramInstancePath	String	Path to the diagram instance to set information for. Example: "MyLibrary.MyDiagramType.MyDI" <i>Note! The path can contain several instance levels.</i> <i>Example:</i> <i>"MyApplic.Diagram1.DiagInst1.DiagInst2.DiagInst3"</i>
diagramInstanceContent	String	The information to set for the selected diagram instance as an XML-string. See diagram instance XML example .

Return value

A [message bucket](#) is returned.

5.1.19.5 DeleteDiagramInstance

```
HRESULT DeleteDiagramInstance([in] BSTR diagramInstancePath);
```

```
Sub DeleteDiagramInstance(diagramInstancePath As String)
```

Description

Deletes the diagram instance specified by **diagramInstancePath** from the project.

Parameters

Name	Type	Description
diagramInstancePath	String	Path to the diagram instance to delete. Example: “MyLibrary.MyDiagramType.MyDI”

5.1.19.6 **RenameDiagramInstance**

```
HRESULT RenameDiagramInstance([in] BSTR diagramInstancePath,  
                               [in] BSTR newDiagramInstanceName);
```

```
Sub RenameDiagramInstance(diagramInstancePath As String, _  
                           newDiagramInstanceName As String)
```

Description

Renames the diagram instance specified by **diagramInstancePath** to **newDiagramInstanceName**.

Parameters

Name	Type	Description
diagramInstancePath	String	Path to the diagram instance to rename. Example: “MyApplication.MyDiagram.MyDI”
newDiagramInstanceName	String	The new name of the selected diagram instance. (Only name, not complete path)

5.1.20 Methods for Program

5.1.20.1 Program XML example

XML Example for program:

```
<?xml version="1.0" encoding="UTF-16"?>
<Program xmlns="CBOpenIFSschema3_0" Name="MyProgram"
TaskConnection="MyController.Normal" SILLevel="NonSIL" SimulationMark="0">
  <Variables>
    <Variable Name="Var1" Type="dint" Attribute="retain" InitialValue="7"
ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.dint"/>
    <Variable Name="Var2" Type="dint" Attribute="retain" InitialValue="0"
ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.dint"/>
  </Variables>
  <Signals>
    <Signal Name="Signal1" Path="MyApp.Var1" Direction="in" AcknowledgeGroup="auto">
      <Description>Description of Signal1</Description>
    </Signal>
  </Signals>
  <FunctionBlocks>
    <FunctionBlock Name="FBInst1" Type="AnotherFBType"
TaskConnection="MyController.Fast" TypePath="MyApplication 1.0-0.AnotherFBType">
      <Description>Description of FBInst1</Description>
    </FunctionBlock>
    <FunctionBlock Name="FBInst2" Type="MyFBType" TaskConnection=""
TypePath="MyApplication 1.0-0.MyFBType"/>
  </FunctionBlocks>
  <CodeBlocks>
    <STCodeBlock Name="CodeBlock1">
      <ST_Code>
        FBInst1( Parl := Var1 );
        FBInst2( Parl := Var2 );
      </ST_Code>
    </STCodeBlock>
    <STCodeBlock Name="CodeBlock2">
      <ST_Code>Var1:=Var2+1;</ST_Code>
    </STCodeBlock>
  </CodeBlocks>
  <Description>Description of the program "MyProgram"</Description>
</Program>
```

For further information see the [XML Schema](#) and [semantics for omitted attributes and elements](#).

5.1.20.2 NewProgram

```
HRESULT NewProgram([in] BSTR programName,  
                  [in] BSTR applicationName,  
                  [in] BSTR programContent,  
                  [out, retval] BSTR* messages);
```

```
Function NewProgram(programName As String, _  
                    applicationName As String, _  
                    programContent As String) As String
```

Description

Creates a new program with the name **programName**. The parameter **programContent** is optional. It sets the content of the program using an XML-string. If this parameter is empty, an empty program will be created. If content is provided the “Name” attribute in the XML-string is ignored since the parameter **programName** is used as the program name.

Parameters

Name	Type	Description
programName	String	The name of the program to create
applicationName	String	Name of the application where to put the new program
programContent	String	Optional parameter. A string containing an XML description of the new program. See program XML example

Return value

A [message bucket](#) is returned.

5.1.20.3 GetProgram

```
HRESULT GetProgram([in] BSTR programPath,  
                  [out, retval] BSTR* programContent);
```

```
Function GetProgram(programPath As String) As String
```

Description

Returns the content of the program, specified by **programPath**, as an XML-string.

Parameters

Name	Type	Description
programPath	String	Path to the program to retrieve information from. Example: "MyApplication.MyProgram"

Return value

A string containing an XML description of the program is returned. See [program XML example](#)

5.1.20.4 SetProgram

```
HRESULT SetProgram([in] BSTR programPath,  
                  [in] BSTR programContent,  
                  [out, retval] BSTR* messages);
```

```
Function SetProgram(programPath As String, _  
                  programContent As String) As String
```

Description

Sets the content of the program specified by **programPath** using an XML-string as the parameter **programContent**. The “Name” attribute in the XML-string is ignored, i.e. you can’t change the name of the program using the XML.

Parameters

Name	Type	Description
programPath	String	Path to the program to set information for. Example: “MyApplication.MyProgram”
programContent	String	The information to set for the selected program as an XML-string. See program XML example

Return value

A [message bucket](#) is returned.

5.1.20.5 DeleteProgram

```
HRESULT DeleteProgram([in] BSTR programPath);
```

```
Sub DeleteProgram(programPath As String)
```

Description

Deletes the program specified by programPath from the project.

Parameters

Name	Type	Description
programPath	String	Path to the program to delete. Example: MyApplication.MyProgram

5.1.20.6 RenameProgram

```
HRESULT RenameProgram([in] BSTR programPath,  
                      [in] BSTR newProgramName);
```

```
Sub RenameProgram(programPath As String, newProgramName As String)
```

Description

Renames the program specified by **programPath** to **newProgramName**.

Parameters

Name	Type	Description
programPath	String	Path to the program to rename. Example: "MyApplication.MyProgram"
newProgramName	String	The new name of the selected program. (Only name, not complete path)

5.1.21 Methods for Diagram

5.1.21.1 Diagram XML example

XML Example for diagram:

```
<?xml version="1.0" encoding="utf-16"?>
<Diagram xmlns="CBOpenIFSchema3_0" Name="Diagram1" TaskConnection=""
SILLevel="NonSIL" RestrictedSIL="0" SimulationMark="0" AccessLevel="Read Only"
SafetyType="None">
  <Variables>
    <Variable Name="Var1" Type="dint" TypePath="System.dint" AccessLevel="Read Only"
SafetyType="None" Attribute="retain" InitialValue="" ReadPermission=""
WritePermission="" AuthenticationLevel="None"/>
    <Variable Name="Var2" Type="dint" TypePath="System.dint" AccessLevel="Read Only"
SafetyType="None" Attribute="retain" InitialValue="" ReadPermission=""
WritePermission="" AuthenticationLevel="None"/>
  </Variables>
  <CommVariables>
    <CommVariable Name="CommVar1" Type="dint" TypePath="System.dint"
Attribute="retain" InitialValue="" ReadPermission="" Direction="in" IPAddress="auto"
IntervalTime="normal" Priority="normal" ISPValue="" ExpectedSIL="same" UniqueID="0"
AcknowledgeGroup="auto"/>
  </CommVariables>
  <Signals>
    <Signal Name="Signal1" Path="MyApp.Var1" Direction="in" AcknowledgeGroup="auto">
      <Description>Description of Signal1</Description>
    </Signal>
  </Signals>
  <FunctionBlocks>
    <FunctionBlock Name="MyFBType_1" Type="Appl.MyFBType" TypePath="Appl.MyFBType"
AccessLevel="Read Only" SafetyType="None" TaskConnection="" AspectObject="1"
ExposePropertiesInParent="0"/>
  </FunctionBlocks>
  <ControlModules>
    <ControlModule Name="MyControlModuleType_1" Type="Appl.MyControlModuleType"
TypePath="Appl.MyControlModuleType" AccessLevel="Read Only" SafetyType="None"
TaskConnection="" AspectObject="1" ExposePropertiesInParent="0"
VisibilityInGraphics="default"/>
  </ControlModules>
  <DiagramInstances>
    <DiagramInstance Name="MyDiagramType_1" Type="Appl.MyDiagramType"
TypePath="Appl.MyDiagramType" AccessLevel="Read Only" SafetyType="None"
AspectObject="1" ExposePropertiesInParent="0"/>
  </DiagramInstances>
  <InitValues/>
  <CodeBlocks>
    <FDCodeBlock Name="Code">
      <FunctionDiagram>
        <Pages>
          <Layout FormatName="A4" Orientation="Portrait" Unit="0.01 mm" Width="21000"
Height="29700" Top="1500" Bottom="1500" Left="1500" Right="1500"/>
          <Page PageNo="1" PageName="">
            <Blocks>
              <Invocation Name="MyDiagramType_1" DataFlowOrder="2" ENPort="false"
Id="83fb5a03-4f3d-48ce-9b90-a493fa58913a">
                <Description></Description>
                <Layout X="5757" Y="1524" Width="2032" Height="1354">
                  <Ports>
                    <Port Name="Par1" Visible="true"/>
                    <Port Name="Par2" Visible="true"/>
                  </Ports>
                </Invocation>
              </Blocks>
            </Page>
          </Layout>
        </Pages>
      </FunctionDiagram>
    </FDCodeBlock>
  </CodeBlocks>
</Diagram>
```

```

        </Layout>
    </Invocation>
    <Invocation Name="add(1)" DataFlowOrder="1" ENPort="false"
Id="8e062d74-5971-45da-a5af-430c87c109d1">
        <Description></Description>
        <Layout X="3386" Y="2370" Width="677" Height="1354">
            <Ports>
                <Port Name="IN1" Visible="true"/>
                <Port Name="IN2" Visible="true"/>
                <Port Name="" Visible="true"/>
            </Ports>
        </Layout>
    </Invocation>
    <DataRef Name="Var1(1)" Id="c113a137-6418-4db9-87a6-cfdd1d939dee">
        <Description></Description>
        <Layout X="1016" Y="3217" Width="677" Height="338"></Layout>
    </DataRef>
    <DataRef Name="CommVar1(1)" Id="b7d3edb1-89ee-4583-a5d0-5dbc0de33f49">
        <Description></Description>
        <Layout X="3386" Y="1185" Width="677" Height="338"></Layout>
    </DataRef>
    <DataRef Name="Var2(1)" Id="a84c40ba-bc5a-4e24-8683-d5ef1d97c511">
        <Description></Description>
        <Layout X="1016" Y="2032" Width="677" Height="338"></Layout>
    </DataRef>
    <Invocation Name="MyControlModuleType_1" DataFlowOrder="3"
ENPort="false" Id="80701b4b-7feb-42d6-bfe7-300ce13262db">
        <Description></Description>
        <Layout X="9821" Y="677" Width="2709" Height="677"></Layout>
    </Invocation>
    <Invocation Name="MyFBType_1" DataFlowOrder="4" ENPort="false"
Id="f85c8255-bc24-4184-9bf3-6486520ba8a4">
        <Description></Description>
        <Layout X="9821" Y="2540" Width="1524" Height="677"></Layout>
    </Invocation>
</Blocks>
<DataConnections>
    <DataConnection Src="add(1)" Dest="MyDiagramType_1.Par2" Id="d26b6bbb-
a371-4619-9bc0-77ebd183b666">
        <Layout>
            <SrcPos X="4275" Y="3048"/>
            <LinePos DX="-169" DY="0"/>
            <LinePos DX="0" DY="-508"/>
            <DestPos X="5545" Y="2540"/>
        </Layout>
    </DataConnection>
    <DataConnection Src="CommVar1(1)" Dest="MyDiagramType_1.Par1"
Id="0fef3aca-5eb2-4e5e-932f-b12eef582e94">
        <Layout>
            <SrcPos X="4275" Y="1354"/>
            <LinePos DX="-84" DY="0"/>
            <LinePos DX="0" DY="846"/>
            <DestPos X="5545" Y="2201"/>
        </Layout>
    </DataConnection>
    <DataConnection Src="Var2(1)" Dest="add(1).IN1" Id="8dac1468-468a-4c69-
bce5-49f9a7a465f3">
        <Layout>
            <SrcPos X="1905" Y="2201"/>
            <LinePos DX="-169" DY="0"/>
            <LinePos DX="0" DY="846"/>
            <DestPos X="3175" Y="3048"/>
        </Layout>
    </DataConnection>

```

```

        <DataConnection Src="Var1(1)" Dest="add(1).IN2" Id="2586a95a-4333-4909-
8cae-5f215c2aafef">
            <Layout>
                <SrcPos X="1905" Y="3386"/>
                <DestPos X="3175" Y="3386"/>
            </Layout>
        </DataConnection>
    </DataConnections>
</Page>
</Pages>
</FunctionDiagram>
</FDCodeBlock>
</CodeBlocks>
<Description/>
</Diagram>

```

For further information see the [XML Schema](#) and [semantics for omitted attributes and elements](#).

5.1.21.2 NewDiagram

```
HRESULT NewDiagram([in] BSTR diagramName,  
                  [in] BSTR applicationName,  
                  [in] BSTR diagramContent,  
                  [out, retval] BSTR* messages);
```

```
Function NewDiagram(diagramName As String, _  
                  applicationName As String, _  
                  diagramContent As String) As String
```

Description

Creates a new diagram with the name **diagramName**. The parameter **diagramContent** is optional. It sets the content of the diagram using an XML-string. If this parameter is empty, an empty diagram will be created. If content is provided the “Name” attribute in the XML-string is ignored since the parameter **diagramName** is used as the diagram name.

Parameters

Name	Type	Description
diagramName	String	The name of the diagram to create
applicationName	String	Name of the application where to put the new diagram
diagramContent	String	Optional parameter. A string containing an XML description of the new diagram. See diagram XML example

Return value

A [message bucket](#) is returned.

5.1.21.3 GetDiagram

```
HRESULT GetDiagram([in] BSTR diagramPath,  
                   [out, retval] BSTR* diagramContent);
```

```
Function GetDiagram(diagramPath As String) As String
```

Description

Returns the content of the diagram, specified by **diagramPath**, as an XML-string.

Parameters

Name	Type	Description
diagramPath	String	Path to the diagram to retrieve information from. Example: "MyApplication.MyDiagram"

Return value

A string containing an XML description of the diagram is returned. See [diagram XML example](#)

5.1.21.4 SetDiagram

```
HRESULT SetDiagram([in] BSTR diagramPath,  
                  [in] BSTR diagramContent,  
                  [out, retval] BSTR* messages);
```

```
Function SetDiagram(diagramPath As String, _  
                  diagramContent As String) As String
```

Description

Sets the content of the diagram specified by **diagramPath** using an XML-string as the parameter **diagramContent**. The “Name” attribute in the XML-string is ignored, i.e. you can’t change the name of the diagram using the XML.

Parameters

Name	Type	Description
diagramPath	String	Path to the diagram to set information for. Example: “MyApplication.MyDiagram”
diagramContent	String	The information to set for the selected diagram as an XML-string. See diagram XML example

Return value

A [message bucket](#) is returned.

5.1.21.5 DeleteDiagram

```
HRESULT DeleteDiagram([in] BSTR DiagramPath);
```

```
Sub DeleteDiagram(diagramPath As String)
```

Description

Deletes the diagram specified by diagramPath from the project.

Parameters

Name	Type	Description
diagramPath	String	Path to the diagram to delete. Example: MyApplication.MyDiagram

5.1.21.6 RenameDiagram

```
HRESULT RenameDiagram([in] BSTR diagramPath,  
                        [in] BSTR newDiagramName);
```

```
Sub RenameDiagram(diagramPath As String, newDiagramName As String)
```

Description

Renames the diagram specified by **diagramPath** to **newDiagramName**.

Parameters

Name	Type	Description
diagramPath	String	Path to the diagram to rename. Example: "MyApplication.MyDiagram"
newDiagramName	String	The new name of the selected diagram. (Only name, not complete path)

5.1.22 Methods for HardwareType

5.1.22.1 HardwareType XML example

XML example:

```
<?xml version="1.0" encoding="utf-16"?>
<HardwareType xmlns="CBOpenIFSchema3_0" Name="MyHwType" Guid="52F8CF75-395E-4942-
AD4B-6467D5029B5A" HWTypeId="3E8802D3-DAA0-4D3C-AA9D-3240F62F6F1F"
Icon="CustomIcon.ico" RedundantIcon="RedundantIO" HardwareUnitAsEntity="0"
RestrictInstantiation="0">
  <Description>Description of this hardware type</Description>
  <HWDefinitionFile Name="MyHwType" Version="" TimeStamp="2015-04-16-12:33:15.211"
Extension="hwd"/>
</HardwareType>
```

For further information see the [XML Schema](#) and [semantics for omitted attributes and elements](#).

See also [What XML elements and attributes are needed by Set and New – methods?](#)

5.1.22.2 InsertHardwareDefinitionFile

```
HRESULT InsertHardwareDefinitionFile([in] BSTR hardwareLibraryName,  
                                     [in] BSTR filePath,  
                                     [out] VARIANT_BOOL* fileAdded,  
                                     [out, retval] BSTR* messages);
```

```
Function InsertHardwareDefinitionFile(hardwareLibraryName As String,  
_                                     filePath As String, _  
                                     fileAdded As Boolean) As  
String
```

Description

Inserts a hardware definition file into a hardware library thereby creating or updating a hardware type.

Parameters

Name	Type	Description
hardwareLibraryName	String	The GUID or the name, including version, of the hardware library in which to insert the hardware definition file.
filePath	String	The full file path to the hardware definition file (.hwd) to insert.
fileAdded	Boolean	fileAdded is an out parameter indicating whether the hardware definition file was added or not. Note! The returned HRESULT code will be S_OK even if the file was not added. The reason is that the returned message bucket contains valuable information about the reasons for the failure, and it is not possible (in some languages) to retrieve out parameters when the HRESULT code indicates errors.

5.1.22.3 GetHardwareType

```
HRESULT GetHardwareType([in] BSTR hardwareLibraryName,  
                        [in] BSTR hardwareTypeName,  
                        [out, retval] BSTR* hardwareTypeContent);  
  
Function GetHardwareType(hardwareLibraryName As String, _  
                        hardwareTypeName As String) As String
```

Description

Returns the content of the hardware type, specified by **hardwareLibraryName** and **hardwareTypeName**, as an XML-string.

Parameters

Name	Type	Description
hardwareLibraryName	String	The GUID or the name, including version, of the hardware library from which to get the hardware type.
hardwareTypeName	String	The type GUID or the name of the hardware type to get. Please don't confuse the type GUID of the hardware type with the hardware type identifier which can also be a GUID.

Return value

A string containing an XML description of the data type is returned. See [hardware type XML example](#).

5.1.22.4 SetHardwareType

```
HRESULT SetHardwareType([in] BSTR hardwareLibraryName,  
                        [in] BSTR hardwareTypeName,  
                        [in] BSTR hardwareTypeContent,  
                        [out, retval] BSTR* messages);
```

```
Function SetHardwareType(hardwareLibraryName As String, _  
                        hardwareTypeName As String,  
                        hardwareTypeContent As String _) As String
```

Description

Sets the content of the hardware type specified by **hardwareLibraryName** and **hardwareTypeName** using an XML-string as the parameter **hardwareTypeContent**. The “Name” attribute in the XML-string is ignored, i.e. you can’t change the name of the hardware type using the XML.

Parameters

Name	Type	Description
hardwareLibraryName	String	The GUID or the name, including version, of the hardware library in which to set the hardware type.
hardwareTypeName	String	The type GUID or the name of the hardware type to set information for. Please don’t confuse the type GUID of the hardware type with the hardware type identifier which can also be a GUID.
hardwareTypeContent	String	The information to set for the selected hardware type as an XML-string. See hardware type XML example . Note! The returned HRESULT code will be S_OK even if some properties couldn’t be set. Review the content of the returned message bucket as it contains valuable information about the reasons for the failure. Also note that it is not possible (in some programming languages) to retrieve out parameters when the HRESULT code indicates errors.

Return value

A [message bucket](#) is returned.

5.1.22.5 DeleteHardwareType

```
HRESULT DeleteHardwareType([in] BSTR hardwareLibraryName,  
                           [in] BSTR hardwareTypeName);
```

```
Sub DeleteHardwareType (hardwareLibraryName As String, _  
                       hardwareTypeName As String)
```

Description

Deletes a hardware type from a hardware library. The hardware type can only be deleted if it's not used.

Parameters

Name	Type	Description
hardwareLibraryName	String	The GUID or the name, including version, of the hardware library from which to delete the hardware type.
hardwareTypeName	String	The type GUID or the name of the hardware type to delete. Please don't confuse the type GUID of the hardware type with the hardware type identifier which can also be a GUID.

5.1.23 Methods for Access Variables

5.1.23.1 Access variables XML example

XML example for access variables:

```
<?xml version="1.0" encoding="UTF-16"?>
<Varaccess xmlns="CBOpenIFSschema3_0">
  <VANamedProtocol Name="MMS">
    <VANamedVariable Name="MMSvar1" Path="MyApplication.MyProgram.Var1"
Attribute="ReadOnly" VAType="dint" Row="1"/>
    <VANamedVariable Name="MMSvar2" Path="MyApplication.GlobVar2" Attribute=""
VAType="real" Row="2"/>
  </VANamedProtocol>
  <VANamedProtocol Name="SattBus"/>
  <VAAddressedProtocol Name="Comli">
    <VAAddressedVariable Name="X0" Path="MyApplication.ILProg.LevelTooHigh"
VAType="bool" Row="1"/>
  </VAAddressedProtocol>
</Varaccess>
```

For further information see the [XML Schema](#) and [semantics for omitted attributes and elements](#).

5.1.23.2 GetAccessVariables

```
HRESULT GetAccessVariables([in] BSTR controllerName,
                           [out, retval] BSTR* accessVarContent);
```

```
Function GetAccessVariables(controllerName As String) As String
```

Description

Returns the access variables of the controller, specified by **controllerName**, as an XML-string.

Parameters

Name	Type	Description
controllerName	String	Path to the controller to retrieve information from. Example: "MyController"

Return value

A string containing an XML description of the access variables in the controller is returned. See [access variables XML example](#)

5.1.23.3 SetAccessVariables

```
HRESULT SetAccessVariables([in] BSTR controllerName,  
                           [in] BSTR accessVarContent,  
                           [out, retval] BSTR* messages);
```

```
Function SetAccessVariables(controllerName As String, _  
                           accessVarContent As String) As String
```

Description

Replaces current Access Variables, if any, with new ones described in the **accessVarContent** parameter.

Parameters

Name	Type	Description
controllerName	String	Name of the controller. Example: "MyController"
accessVarContent	String	A string containing an XML description of the new Access Variables. See access variables XML example

Return value

A [message bucket](#) is returned.

Remarks

The XML attribute "VAType" has a default value and may be omitted. The meaning of the "VAType" attribute is to give the users of the **GetAccessVariables** method some extra read only information.

The XML attribute "Row" is not required and may be omitted.

5.1.24 Methods for Task

5.1.24.1 Task XML example

XML example for task:

```
<?xml version="1.0" encoding="UTF-16"?>
<Task xmlns="CBOpenIFSschema3_0" Name="MyTask" IntervalTime="700" Priority="5"
Offset="10" OutputUpdate="last" LatencySupervision="0" LatencyPercentage="10"
TaskSILLevel="0"/>
```

For further information see the [XML Schema](#) and [semantics for omitted attributes and elements](#).

5.1.24.2 NewTask

```
HRESULT NewTask([in] BSTR taskName,
                [in] BSTR controllerName,
                [in] BSTR taskContent);
```

```
Sub NewTask(taskName As String, _
            controllerName As String, _
            taskContent As String)
```

Description

Creates a new task in a controller.

Parameters

Name	Type	Description
taskName	String	Name of the new task
controllerName	String	Name of the controller
taskContent	String	Optional parameter. A string containing an XML description of the task properties. See task XML example . If empty, a task with default properties will be created

5.1.24.3 GetTask

```
HRESULT GetTask([in] BSTR taskPath,  
                [out, retval] BSTR* taskContent);
```

Function GetTask(taskPath As String) As String

Description

Returns the content of the task specified by **taskPath** as an XML string.

Parameters

Name	Type	Description
taskPath	String	Path to the task to retrieve information from. Example: "ControllerName.TaskName"

Return value

A string containing an XML description of the task is returned. See [task XML example](#)

5.1.24.4 SetTask

```
HRESULT SetTask([in] BSTR taskPath,  
                [in] BSTR taskContent);
```

```
Sub SetTask(taskPath As String, taskContent As String)
```

Description

Replaces current task properties, with new ones described by the **taskContent** parameter.
If interval time has a lower value than allowed, it is set to the lowest possible value.
If offset has a value greater than interval time, it is set to interval time.

Parameters

Name	Type	Description
taskPath	String	Path to a task. Example: "ControllerName.TaskName"
taskContent	String	A string containing an XML description of the new task properties. See task XML example .

5.1.24.5 DeleteTask

`HRESULT DeleteTask([in] BSTR taskPath);`

`Sub DeleteTask(taskPath As String)`

Description

Deletes a task in a controller.

Parameters

Name	Type	Description
taskPath	String	Path to the task to delete. Example: “ControllerName.TaskName”

5.1.24.6 RenameTask

```
HRESULT RenameTask([in] BSTR taskPath,  
                    [in] BSTR newTaskName);
```

```
Sub RenameTask (taskPath As String, newTaskName As String)
```

Description

Renames the task specified by **taskPath** to **newTaskName**.

Parameters

Name	Type	Description
taskPath	String	Path to the task to rename. Example: "MyController.MyTask"
newTaskName	String	The new name of the selected task. (Only name, not complete path)

5.1.25 Methods for ConnectedApplications

5.1.25.1 Connected applications XML example

XML example for connected applications:

```
<?xml version="1.0" encoding="UTF-16"?>
<ConnectedApplications xmlns="CBOpenIFSchema3_0">
  <ConnectedApplication Name="MyApplication" MajorVersion="1" MinorVersion="0"
Revision="0"/>
  <ConnectedApplication Name="NewApp" MajorVersion="1" MinorVersion="0"
Revision="0"/>
  <ConnectedApplication Name="Application3" MajorVersion="1" MinorVersion="0"
Revision="0"/>
</ConnectedApplications>
```

For further information see the [XML Schema](#) and [semantics for omitted attributes and elements](#).

5.1.25.2 GetConnectedApplications

```
HRESULT GetConnectedApplications([in] BSTR controllerName,
                                [out, retval] BSTR*
                                connectedApplicationsContent);
```

```
Function GetConnectedApplications(controllerName As String) As String
```

Description

Returns the connected applications of the controller specified by **controllerName** as an XML-string.

Parameters

Name	Type	Description
controllerName	String	Name of the controller to retrieve information from. Example: "MyController"

Return value

A string containing an XML description of the connected applications is returned. See [connected applications XML example](#)

5.1.25.3 SetConnectedApplications

```
HRESULT SetConnectedApplications([in] BSTR controllerName,  
                                [in] BSTR  
connectedApplicationsContent,  
                                [out, retval] BSTR* messages);  
  
Function SetConnectedApplications(controllerName As String, _  
                                connectedApplicationsContent As String _  
                                ) As String
```

Description

Replaces current application connections in a controller, if any, with new ones described in the **connectedApplicationsContent** parameter.

Parameters

Name	Type	Description
controllerName	String	Name of the controller
connectedApplicationsContent	String	A string containing an XML description of the new connected applications. See connected applications XML example

Return value

A [message bucket](#) is returned.

5.1.26 Methods for HWUnits

5.1.26.1 HWUnits XML example

XML example for hardware units:

```
<?xml version="1.0" encoding="UTF-16"?>
<HWUnit xmlns="CBOpenIFSschema3_0" Path="MyCont.0.11.6" TypeID="8914196"
TypeDescription="AO820" HWSimulation="0" HWSimulationSupported="1">
  <ParameterSettings>
    <ParameterSetting Name="ACTIVATE1" Value="1"/>
    <ParameterSetting Name="ACTIVATE2" Value="1"/>
    <ParameterSetting Name="ACTIVATE3" Value="1"/>
    <ParameterSetting Name="ACTIVATE4" Value="1"/>
    <ParameterSetting Name="OSPCTRL_1" Value="Set OSP value"/>
    <ParameterSetting Name="OSPCTRL_2" Value="Set OSP value"/>
    <ParameterSetting Name="OSPCTRL_3" Value="Set OSP value"/>
    <ParameterSetting Name="OSPCTRL_4" Value="Set OSP value"/>
    <ParameterSetting Name="OSPValue_1" Value="0.0"/>
    <ParameterSetting Name="OSPValue_2" Value="0.0"/>
    <ParameterSetting Name="OSPValue_3" Value="0.0"/>
    <ParameterSetting Name="OSPValue_4" Value="0.0"/>
    <ParameterSetting Name="SignalRange_1" Value="4..20mA"/>
    <ParameterSetting Name="SignalRange_2" Value="4..20mA"/>
    <ParameterSetting Name="SignalRange_3" Value="4..20mA"/>
    <ParameterSetting Name="SignalRange_4" Value="4..20mA"/>
  </ParameterSettings>
  <HWChannels>
    <HWChannel Address="QW0.11.6.1" Name="Output 1" Min="0.0" Max="100.0" Unit=""
Fraction="1" Reversed="0">
      <ConVariable
Signal="0">MyApp.SingleControlModule1.MyCMType1.MyFBInst.FBInst.X</ConVariable>
      <IODescription>My IO Description</IODescription>
    </HWChannel>
    <HWChannel Address="QW0.11.6.2" Name="Output 2" Min="0.0" Max="100.0" Unit=""
Fraction="1" Reversed="0">
      <ConVariable Signal="0">MyApp.MyProgram.Sum</ConVariable>
      <IODescription>An IO Description</IODescription>
    </HWChannel>
    <HWChannel Address="QW0.11.6.3" Name="Output 3" Min="0.0" Max="100.0" Unit=""
Fraction="1" Reversed="0">
      <ConVariable Signal="1">MySignalName</ConVariable>
      <IODescription>A Signal Description</IODescription>
    </HWChannel>
    <HWChannel Address="QW0.11.6.4" Name="Output 4" Min="0.0" Max="100.0" Unit=""
Fraction="1" Reversed="0">
      <ConVariable/>
      <IODescription/>
    </HWChannel>
    <HWChannel Address="IW0.11.6.5" Name="UnitStatus">
      <ConVariable/>
      <IODescription/>
    </HWChannel>
  </HWChannels>
</HWUnit>
```

For further information see the [XML Schema](#) and [semantics for omitted attributes and elements](#).

5.1.26.2 NewHardwareUnit

```
HRESULT NewHardwareUnit([in] BSTR hwPath,  
                        [in] VARIANT hwTypeID,  
                        [in] BSTR hwQualifier,  
                        [in] BSTR hwUnitContent,  
                        [in] BSTR redundantTo,  
                        [out, retval] BSTR* messages);  
  
Function NewHardwareUnit(hwPath As String, _  
                        hwTypeID As Variant, _  
                        hwQualifier As String, _  
                        hwUnitContent As String, _  
                        redundantTo As String) As String
```

Description

Part 1. The following applies when the **redundantTo** parameter is an empty string (“”).

Creates a new hardware unit, of the type **hwTypeID**, at the position **hwPath**. The parameter **hwQualifier** is used to precisely determine which hardware type to use in case of several candidates. If the parameter is empty the type is fetched from the connected hardware library where the type is first found.

The format of **hwQualifier** can be either of the two following:

- The name and version of the hardware library to fetch the hardware type from
- A GUID. Note that this is **not** the same GUID as the one that can be submitted in **hwTypeID**. Hardware Type ID will remain intact across different versions of the hardware library whereas the qualifier GUID will change whenever a new major version of the hardware library is created.

The parameter **hwUnitContent** is an optional XML-string that can be used in order to specify the content of the hardware unit.

The method creates a **single** hardware unit. This unit may or may not contain non-removable subunits. The following picture illustrates this:

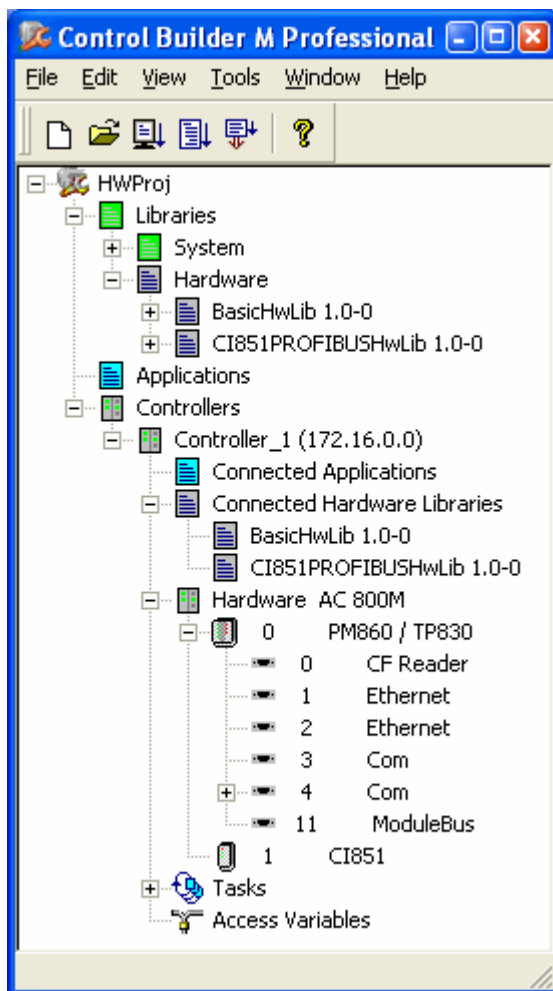


Figure 10

Here CI851 (a Communication Interface PROFIBUS-DP) contains no sub units. On the other hand PM860 / TP830 (a Controller 800 CPU with Base plate) contains five sub units. These sub units can neither be created with the NewHardwareUnit method, nor deleted with the DeleteHardwareUnit method. They are always part of their parent unit and will automatically be created when the parent unit is created and deleted when the parent unit is deleted.

Part 2. The following applies when the **redundantTo** parameter contains a non-empty string.

The parameter **redundantTo** is intended for creating a backup unit for an existing unit, for those hardware unit types that supports this function. **redundantTo** then should contain the hardware path to the existing (or *primary*) hardware unit. The parameters **hwTypeID** and **hwQualifier** are ignored since a backup unit must be of the same type as the primary unit.

The position of the backup unit is handled in two different ways, depending on the type of hardware unit. *Fix position* means that the Control Builder automatically selects the position for the backup unit. *Free position* means that the user is allowed to select position among those that are still available.

When the position is fixed, the **hwPath** parameter is ignored. When the position is free, the **hwPath** parameter shall contain the hardware path to the desired position. However, all but the last number is ignored since the backup unit must have the same parent unit as the primary unit.

Trying to create a backup unit on a hardware unit that does not support this will result in an error.

Parameters

Name	Type	Description
hwPath	String	hwPath specifies the position of the new hardware unit. Example: “ MyController.0.11.1 ” hwTypeID specifies the type of the hardware unit. hwTypeID is either: <ul style="list-style-type: none">• A string specifying a hardware unit type. Example: “DI810”.• A number (Long) uniquely identifying the hardware type.• A GUID uniquely identifying the hardware type. The GUID should be transferred as a string hwQualifier specifies which hardware type to use in case of several candidates. hwQualifier is either: <ul style="list-style-type: none">• A string specifying the name and version of the hardware library to fetch the hardware type from. Example: “HWLib 1.0-0”.• A GUID hwUnitContent is an optional XML-string that can be used in order to specify the content of the hardware unit. See hardware unit XML example redundantTo This is an optional hardware path that is used when adding a backup unit to an existing hardware unit. For more details see description above.
hwTypeID	Variant	
hwQualifier	String	
hwUnitContent	String	
redundantTo	String	

Return value

A [message bucket](#) is returned.

5.1.26.3 GetHardwareUnit

```
HRESULT GetHardwareUnit([in] BSTR hwPath,  
                        [in] VARIANT_BOOL includeSubUnits,  
                        [out, retval] BSTR* hwUnitContent );
```

```
Function GetHardwareUnit(hwPath As String, _  
                        includeSubUnits As Boolean) As String
```

Description

Returns the content of the specified hardware unit as an XML-string.

Parameters

Name	Type	Description
hwPath	String	hwPath specifies the position of the hardware unit. Example: “ MyController.0.11.1 ”
includeSubUnits	Boolean	includeSubUnits specifies whether subordinate hardware units shall be included in the returned XML-string.

Return value

An XML-string describing the content of the hardware unit is returned. See [hardware unit XML example](#). The RedundantPos attribute is only present if the hardware unit has a backup unit.

5.1.26.4 GetHardwareUnitType

This method has been moved to chapter Future Development.

5.1.26.5 SetHardwareUnit

```
HRESULT SetHardwareUnit([in] BSTR hwPath,  
                        [in] BSTR hwUnitContent,  
                        [out, retval] BSTR* messages );
```

```
Function SetHardwareUnit(hwPath As String, _  
                        hwUnitContent As String) As String
```

Description

Sets the content of the hardware unit using an XML-string. The hardware unit is specified by the parameter **hwPath**.

The method sets one **single** hardware unit. The XML-string hwUnitContent may contain information for sub units, but this information is discarded.

Parameters

Name	Type	Description
hwPath	String	hwPath specifies the position of the hardware unit. Example: “ MyController.0.11.1 ”
hwUnitContent	String	hwUnitContent is an XML-string describing the content of the hardware unit. See hardware unit XML example

Return value

A [message bucket](#) is returned.

Remarks

See [semantics for omitted attributes and elements](#). The *SetHardwareUnit* method treat omitted XML attributes and XML elements as follows:

- **<ParameterSetting>** elements not included in the XML document are not affected. That is the Control Builder will retain the old value.
- **<HWChannel>** elements not included in the XML document are not affected.
- If the attribute **Min** for a real channel is omitted, or empty, then the channels scaling settings are not affected.

If a **<ConVariable>** element is empty the current connection is removed.

The name attribute of the **<ParameterSetting>** element should have values according to the hardware definition as opposed to the descriptive texts shown in a hardware editor,

In the address attribute (example **Address=“QW0.11.6.1”**), only the last number is used. This is to allow the same XML content to be used on multiple hardware units without having to recalculate the

channel address. This may be used to get XML content from one hardware unit and set it in another, i.e. a simple copy function.
If the RedundantPos attribute is present it is ignored.

5.1.26.6 DeleteHardwareUnit

```
HRESULT DeleteHardwareUnit([in] BSTR hwPath,  
                           [in] VARIANT_BOOL removeRedundantOnly);
```

```
Sub DeleteHardwareUnit(hwPath As String, _  
                      removeRedundantOnly As Boolean)
```

Description

Deletes a hardware unit specified by the parameter **hwPath**, if **removeRedundantOnly** is false. If the unit is redundant both the primary and the backup unit are deleted.

If **removeRedundantOnly** is true then, if present, the backup unit is removed for the hardware unit specified by the parameter **hwPath**. The primary unit remains.

Trying to remove a backup unit on a hardware unit that does not support this, or does not have a backup unit, will result in an error.

Parameters

Name	Type	Description
hwPath	String	hwPath specifies the position of the hardware unit. Example: “ MyController.0.11.1 ”
removeRedundantOnly	boolean	If true, then only the backup unit is removed.

5.1.26.7 MoveHardwareUnitTo

```
HRESULT MoveHardwareUnitTo([in] BSTR* oldHwPath,  
                           [in] BSTR * newHwPath,  
                           [in] VARIANT_BOOL doSwap);
```

```
Sub MoveHardwareUnitTo(oldHwPath As String, _  
                      newHwPath As String, _  
                      doSwap As Boolean)
```

Description

Moves a hardware unit from one position to another position. If the new position is occupied, the parameter **doSwap** indicates where or not it is allowed to swap the two units. If the new position is occupied and **doSwap** is false, an error is returned.

Parameters

Name	Type	Description
oldHwPath	String	oldHwPath specifies the old position of the hardware unit. Example: “ MyController.0.11.1 ”
newHwPath	String	newHwPath specifies the new position of the hardware unit.
doSwap	boolean	If this parameter is set to true, the units at the old and new path are allowed to trade places (if there is a unit at the new place).

5.1.26.8 ReplaceHardwareUnitType

```
HRESULT ReplaceHardwareUnitType([in] BSTR hwPath,  
                                [in] VARIANT hwTypeID,  
                                [in] BSTR hwQualifier);
```

```
Sub ReplaceHardwareUnitType(hwPath As String, _  
                            hwTypeID As Variant,  
                            hwQualifier As String)
```

Description

Replaces the current type for the HWUnit located at **hwPath** to the new type indicated by **hwTypeID** and **hwQualifier**. . The parameter hwQualifier is used to precisely determine which hardware type to use in case of several candidates. If the parameter is empty the type is fetched from the connected hardware library where the type is first found.

The format of **hwQualifier** can be either of the two following:

- The name and version of the hardware library to fetch the hardware type from
- A GUID. Note that this is **not** the same GUID as the one that can be submitted in **hwTypeID**. Hardware Type ID will remain intact across different versions of the hardware library whereas the qualifier GUID will change whenever a new major version of the hardware library is created.

Parameters

Name	Type	Description
hwPath	String	hwPath specifies the position of the hardware unit. Example: "MyController.0.11.1"
hwTypeID	Variant	hwTypeID specifies the new type of the hardware unit. hwTypeID is either: <ul style="list-style-type: none">• A string specifying a hardware unit type. Example: "DI810".• A number (Long) uniquely identifying the hardware type.• A GUID uniquely identifying the hardware type. The GUID should be transferred as a string.
hwQualifier	String	hwQualifier specifies which hardware type to use in case of several candidates. hwQualifier is either: <ul style="list-style-type: none">• A string specifying the name and version of the hardware library to fetch the hardware type from. Example: "HWLib 1.0-0".• A GUID



ABB AB

Doc. no.

3BSE033313

Lang.

en

Rev. ind.

Q

Page

168

5.1.27 Methods for Online

5.1.27.1 Online

```
HRESULT Online([out] VARIANT_BOOL* isOnline,  
               [out, retval] BSTR* messages);
```

```
Function Online(isOnline As Boolean) As String
```

Description

Changes the mode of the Control Builder from **Offline** to **Online**. If the project contains errors preventing the **Online** transition the returned message bucket contains information about the reason.

Parameters

Name	Type	Description
isOnline	Boolean	<p>isOnline is an out parameter telling whether the transition to Online succeeded or not.</p> <p>Note! The returned HRESULT code will be S_OK even if the transition to Online mode failed. The reason is that the returned message bucket contains valuable information about the reasons for the failure, and it is not possible (in some languages) to retrieve out parameters when the HRESULT code indicates errors.</p>

Return value

A [message bucket](#) is returned

5.1.27.2 DownloadAndGoOnline

```
HRESULT DownloadAndGoOnline([out] VARIANT_BOOL* isOnline,  
                             [out, retval] BSTR* messages);
```

```
Function DownloadAndGoOnline(isOnline As Boolean) As String
```

Description

Changes the mode of the Control Builder to **Online** and download applications to the controllers. If the project contains errors preventing the **Online** transition the returned message bucket contains information about the reason.

Parameters

Name	Type	Description
isOnline	Boolean	isOnline is an out parameter telling whether the transition to Online mode succeeded or not. Note! The returned HRESULT code will be S_OK even if the transition to Online mode failed. The reason is that the returned message bucket contains valuable information about the reasons for the failure, and it is not possible (in some languages) to retrieve out parameters when the HRESULT code indicates errors.

Return value

A [message bucket](#) is returned

5.1.27.3 TestMode

```
HRESULT TestMode([out] VARIANT_BOOL* isTestMode,  
                 [out, retval] BSTR* messages);
```

```
Function TestMode(isTestMode As Boolean) As String
```

Description

Changes the mode of the Control Builder from **Offline** to **TestMode**. If the project contains errors preventing the **TestMode** transition the returned message bucket contains information about the reason.

Parameters

Name	Type	Description
isTestMode	Boolean	isTestMode is an out parameter telling whether the transition to TestMode succeeded or not. Note! The returned HRESULT code will be S_OK even if the transition to TestMode mode failed. The reason is that the returned message bucket contains valuable information about the reasons for the failure, and it is not possible (in some languages) to retrieve out parameters when the HRESULT code indicates errors.

Return value

A [message bucket](#) is returned.

5.1.27.4 **Offline**

`HRESULT Offline([out, retval] BSTR* messages);`

`Function Offline() As String`

Description

Changes the mode of the Control Builder from **Online** or **TestMode** to **Offline**.

Return value

A [message bucket](#) is returned.

5.1.28 Methods for Folder

Folders can appear in different structures, for example the Applications structure. To select what structure the folder belongs to, the first parameter named **typeOfFolder** of the type **enum CBOpenIFFolderType** can have one of the following values:

- OI_APPLICATIONFOLDER

5.1.28.1 NewFolder

```
HRESULT NewFolder([in] enum COpenIFFolderType typeOfFolder,  
                  [in] BSTR folderName,  
                  [in] BSTR pathToParentFolder,  
                  [in] BSTR guid);
```

```
Sub NewFolder(typeOfFolder As COpenIFFolderType , _  
              folderName As String, _  
              pathToParentFolder As String, _  
              guid As String)
```

Description

Creates a new folder with the name **folderName** in the parent folder pointed out by **pathToParentFolder** and in the structure indicated by **typeOfFolder**.

Parameters

Name	Type	Description
typeOfFolder	enum COpenIFFolderType	Determines in which structure to create the folder.
folderName	String	Name of the new folder.
pathToParentFolder	String	An optional parameter specifying in which parent folder to create the new folder. The parameter can be either a path to the parent folder or its GUID. Example: "MyFolder.MySubFolder". If this parameter is an empty string, the folder will be created directly under the selected structure's root folder, for example directly under the "Applications" leaf in Project Explorer.
Guid	String	Remark: This method will fail if the provided parent folder path points to a non-existing folder. An optional parameter specifying a Globally Unique Identifier. If this parameter is an empty string the Control Builder will assign a GUID to the folder. If the parameter is present the syntax must be as follows: "12345678-1234-1234-1234-123456789ABC"

5.1.28.2 RenameFolder

```
HRESULT RenameFolder([in] enum COpenIFFolderType typeOfFolder,  
                    [in] BSTR folderName,  
                    [in] BSTR newFolderName);
```

```
Sub RenameFolder(typeOfFolder As COpenIFFolderType , _  
                folderName As String, _  
                newFolderName As String)
```

Description

Renames the folder.

Parameters

Name	Type	Description
typeOfFolder	enum COpenIFFolderType	Determines in which structure the folder is located.
folderName	String	This parameter can be either the path and name of the folder to rename or its GUID. Example: "MyParentFolder.MyFolder". Omitting the path (providing the folder name only) implies that the folder is located directly under the selected structure's root folder, for example directly under the "Applications" leaf in Project Explorer.
newFolderName	String	The new name of the folder. Remark: The new folder name must not contain a folder path.

5.1.28.3 DeleteFolder

```
HRESULT DeleteFolder([in] enum CBOpenIFFolderType typeOfFolder,  
                    [in] BSTR folderName);
```

```
Sub DeleteFolder(typeOfFolder As CBOpenIFFolderType , _  
                folderName As String)
```

Description

Deletes the folder. Any sub folders and sub objects (for example applications) residing in the folder will also be deleted.

Parameters

Name	Type	Description
typeOfFolder	enum CBOpenIFFolderType	Determines in which structure the folder is located.
folderName	String	This parameter can be either the path and name of the folder to delete or its GUID. Example: "MyParentFolder.MyFolder". Omitting the path (providing the folder name only) implies that the folder is located directly under the selected structure's root folder, for example directly under the "Applications" leaf in Project Explorer.

5.1.28.4 MoveFolder

```
HRESULT MoveFolder([in] enum CBOpenIFFolderType typeOfFolder,  
                  [in] BSTR folderName,  
                  [in] BSTR destinationFolderName);
```

```
Sub MoveFolder(typeOfFolder As CBOpenIFFolderType , _  
              folderName As String, _  
              destinationFolderName As String)
```

Description

Moves the folder. The destination folder must be located in the same structure as the folder to move. It is not possible to move a parent folder to one of its child folders. All sub folder and sub objects (for example applications) residing in the folder will also be moved.

Parameters

Name	Type	Description
typeOfFolder	enum CBOpenIFFolderType	Determines in which structure the folder is located.
folderName	String	This parameter can be either the path and name of the folder to move or its GUID. Example: "MyParentFolder.MyFolder". Omitting the path (providing the folder name only) implies that the folder is located directly under the selected structure's root folder, for example directly under the "Applications" leaf in Project Explorer.
destinationFolder Name	String	This parameter can be either the path and name of the destination folder or its GUID. Example: "MyParentFolder.MyFolder". If this parameter is an empty string, the folder will be moved to directly under the selected structure's root folder, for example directly under the "Applications" leaf in Project Explorer. Remark: This method will fail if the provided destination folder name points to a non-existing folder or to a folder in a different structure.

5.1.28.5 MoveFolderObject

```
HRESULT MoveFolderObject([in] enum COpenIFFolderType typeOfFolder,  
                          [in] BSTR objectName,  
                          [in] BSTR destinationFolderName);
```

```
Sub MoveFolderObject(typeOfFolder As COpenIFFolderType , _  
                    objectName As String, _  
                    destinationFolderName As String)
```

Description

Moves an object (for example an application) from one folder to another. The destination folder must be located in the same structure as the object to move.

Parameters

Name	Type	Description
typeOfFolder	enum COpenIFFolderType	Determines in which structure the object is located.
objectName	String	This parameter can be either the path and name of the object to move or its GUID. Example: "MyParentFolder.MyApplication". Omitting the path (providing the object name only) implies that the object is located directly under the selected structure's root folder, for example directly under the "Applications" leaf in Project Explorer.
destinationFolder Name	String	This parameter can be either the path and name of the destination folder or its GUID. Example: "MyParentFolder.MyFolder". If this parameter is an empty string, the object will be moved to directly under the selected structure's root folder, for example directly under the "Applications" leaf in Project Explorer. Remark: This method will fail if the provided destination folder name points to a non-existing folder or to a folder in a different structure.

5.1.29 Methods for retrieving information from the current project

5.1.29.1 GetProjectTree XML example

GetProjectTree is a general method for retrieving information about the entire or parts of the current project. The returned XML-string depends on the filter parameters **path**, **depth** and **includeRuntimeInstances**. The examples below start out from the project shown in Figure 11.

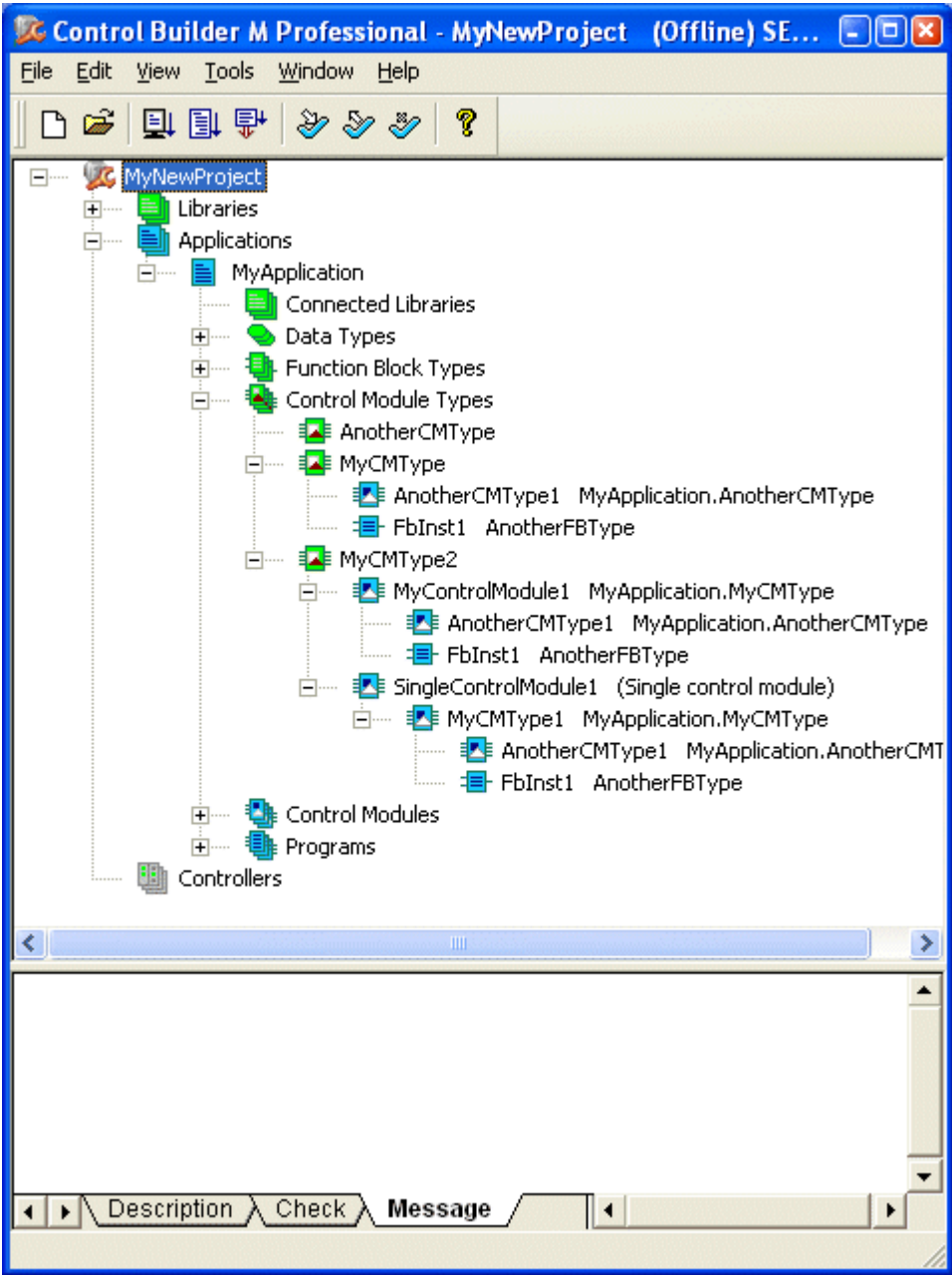


Figure 11

5.1.29.1.1 GetProjectTree example 1

Assume the following parameters: **path** = “Applications.MyApplication.ControlModuleTypes”, **depth** = 0 and **includeRuntimeInstances** = false. (Study Figure 11 above!)

The parameters above mean: “return all control module types in the application named “MyApplication”. Thus, the GetProjectTree method will return the following XML string:

```
<?xml version="1.0" encoding="UTF-16"?>
<ControlModuleTypes xmlns="CBOpenIFSschema3_0">
  <ControlModuleType Name="AnotherCMType" Protected="0" Hidden="0" Scope="public"
InteractionWindow="" AlarmOwner="1" BatchObject="" SILLevel="NonSIL"
SimulationMark="0">
    <GraphSize>
      <Point X="-1.0" Y="-1.0"/>
      <Point X="1.0" Y="1.0"/>
    </GraphSize>
  </ControlModuleType>
  <ControlModuleType Name="MyCMType2" Protected="0" Hidden="0" Scope="public"
InteractionWindow="" AlarmOwner="1" BatchObject="" SILLevel="NonSIL"
SimulationMark="0">
    <GraphSize>
      <Point X="-1.0" Y="-1.0"/>
      <Point X="1.0" Y="1.0"/>
    </GraphSize>
  </ControlModuleType>
</ControlModuleTypes>
```

5.1.29.1.2 GetProjectTree example 2

Assume the following parameters: **path** = “Applications.MyApplication.ControlModuleTypes.MyCMType2”, **depth** = 1 and **includeRuntimeInstances** = false. (Study Figure 11 above!)

The parameters above mean: “return the control module type “MyCMType2” in the application named “MyApplication” and information about the sub instances (function blocks, control modules or single control modules) of the type. Thus, the GetProjectTree method will return the following XML string:

```
<?xml version="1.0" encoding="UTF-16"?>
<ControlModuleType xmlns="CBOpenIFSschema3_0" Name="MyCMType2" Protected="0"
Hidden="0" Scope="public" InteractionWindow="" AlarmOwner="1" BatchObject=""
SILLevel="NonSIL" SimulationMark="0">
  <FunctionBlocks/>
  <ControlModules>
    <ControlModule Name="MyControlModule1" Type="MyCMType" TaskConnection=""
VisibilityInGraphics="invisible" TypePath="MyApplication 1.0-0.MyCMType"/>
    <SingleControlModule Name="SingleControlModule1" TaskConnection=""
VisibilityInGraphics="default" TypeGuid="f4a4f744-3135-49b7-b250-9db213dcbeb5"
TypePath="MyApplication 1.0-0.MyCMType2.SingleControlModule1"/>
  </ControlModules>
  <GraphSize>
    <Point X="-1.0" Y="-1.0"/>
    <Point X="1.0" Y="1.0"/>
  </GraphSize>
</ControlModuleType>
```

5.1.29.1.3 GetProjectTree example 3

Assume the following parameters: **path** =

“Applications.MyApplication.ControlModuleTypes.MyCMType2”, **depth** = 5 and

includeRuntimeInstances = true. (Study Figure 11 above!)

The parameters above mean: “return the control module type “MyCMType2” in the application named “MyApplication” and information about the sub instances (function blocks, control modules or single control modules) of the type. Return sub instances until a depth of 5 and include runtime instances. Thus, the GetProjectTree method will return the following XML string:

```
<?xml version="1.0" encoding="UTF-16"?>
<ControlModuleType xmlns="CBOpenIFSschema3_0" Name="MyCMType2" Protected="0"
Hidden="0" Scope="public" InteractionWindow="" AlarmOwner="1" BatchObject=""
SILLevel="NonSIL" SimulationMark="0">
  <FunctionBlocks/>
  <ControlModules>
    <ControlModule Name="MyControlModule1" Type="MyCMType" TaskConnection=""
VisibilityInGraphics="invisible" TypePath="MyApplication 1.0-0.MyCMType">
      <FunctionBlocks>
        <FunctionBlock Name="FbInst1" Type="AnotherFBType" TaskConnection=""
TypePath="MyApplication 1.0-0.AnotherFBType">
          <FunctionBlocks/>
          <ControlModules/>
          <Description>Description of the instance</Description>
        </FunctionBlock>
      </FunctionBlocks>
    </ControlModules>
    <ControlModule Name="AnotherCMType1" Type="AnotherCMType" TaskConnection=""
VisibilityInGraphics="invisible" TypePath="MyApplication 1.0-0.AnotherCMType">
      <FunctionBlocks/>
      <ControlModules/>
    </ControlModule>
  </ControlModules>
  <SingleControlModule Name="SingleControlModule1" TaskConnection=""
VisibilityInGraphics="default" TypeGuid="f4a4f744-3135-49b7-b250-9db213dcbeb5"
TypePath="MyApplication 1.0-0.MyCMType2.SingleControlModule1">
    <FunctionBlocks/>
    <ControlModules>
      <ControlModule Name="MyCMType1" Type="MyCMType" TaskConnection=""
VisibilityInGraphics="invisible" TypePath="MyApplication 1.0-0.MyCMType">
        <FunctionBlocks>
          <FunctionBlock Name="FbInst1" Type="AnotherFBType" TaskConnection=""
TypePath="MyApplication 1.0-0.AnotherFBType">
            <FunctionBlocks/>
            <ControlModules/>
            <Description>Description of the instance</Description>
          </FunctionBlock>
        </FunctionBlocks>
      </ControlModules>
      <ControlModule Name="AnotherCMType1" Type="AnotherCMType"
TaskConnection="" VisibilityInGraphics="invisible" TypePath="MyApplication 1.0-
0.AnotherCMType">
        <FunctionBlocks/>
        <ControlModules/>
      </ControlModule>
    </ControlModules>
  </SingleControlModule>
</ControlModules>
</GraphSize>
```

```

    <Point X="-1.0" Y="-1.0"/>
    <Point X="1.0" Y="1.0"/>
  </GraphSize>
</ControlModuleType>

```

5.1.29.2 GetProjectTree Folder XML example

The example below starts out from the project shown in Figure 12. The project contains three application organized in a couple of folders. Some of the folders are empty.

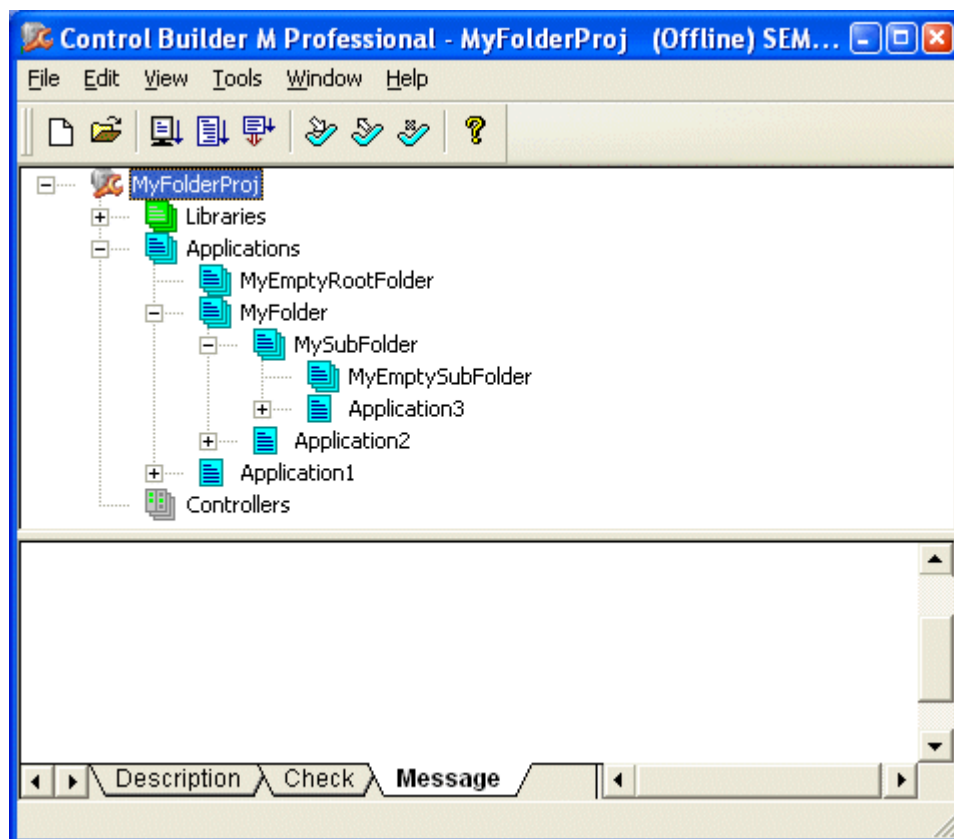


Figure 12

The XML below was retrieved by calling GetProjectTree with **path** = "Applications", **depth** = 0 and **includeRuntimeInstances** = false.

The folders are listed directly below the Applications element and the folders are always listed before the applications. Parent folders always appear before child folder. Each Folder element contains the name of the folder and a path to its parent folder. An empty path indicates that the folder is located directly under the Applications node in Project Explorer.

Each Application element contains a folder path to the application's parent folder. An empty path indicates that the application is located directly under the Applications node in Project Explorer.

```

<?xml version="1.0" encoding="utf-16"?>
<Applications xmlns="CBOpenIFSchema3_0">
  <Folder Name="MyEmptyRootFolder" ParentPath=""/>
  <Folder Name="MyFolder" ParentPath=""/>
  <Folder Name="MySubFolder" ParentPath="MyFolder"/>
  <Folder Name="MyEmptySubFolder" ParentPath="MyFolder.MySubFolder"/>
  <Application Name="Application1" FilePath="Projects\MyFolderProj\Application1"
    MajorVersion="1" MinorVersion="0" Revision="0" ReservationState="NotReserved"
    SILLevel="NonSIL" SimulationMark="0" FolderPath="">
    <DataTypes/>
    <FunctionBlockTypes/>
  </Application>
  <Application Name="Application2" ParentPath="MyFolder" />
  <Application Name="Application3" ParentPath="MyFolder.MySubFolder" />
</Applications>

```

```

    <ControlModuleTypes/>
    <ControlModules/>
    <Diagrams/>
    <Programs/>
  </Application>
  <Application Name="Application2" FilePath="Projects\MyFolderProj\Application2"
MajorVersion="1" MinorVersion="0" Revision="0" ReservationState="NotReserved"
SILLevel="NonSIL" SimulationMark="0" FolderPath="MyFolder">
    <DataTypes/>
    <FunctionBlockTypes/>
    <ControlModuleTypes/>
    <ControlModules/>
    <Diagrams/>
    <Programs/>
  </Application>
  <Application Name="Application3" FilePath="Projects\MyFolderProj\Application3"
MajorVersion="1" MinorVersion="0" Revision="0" ReservationState="NotReserved"
SILLevel="NonSIL" SimulationMark="0" FolderPath="MyFolder.MySubFolder">
    <DataTypes/>
    <FunctionBlockTypes/>
    <ControlModuleTypes/>
    <ControlModules/>
    <Diagrams/>
    <Programs/>
  </Application>
</Applications>

```

5.1.29.3 GetProjectTree

```
HRESULT GetProjectTree ([in] BSTR path,  
                        [in] long depth,  
                        [in] VARIANT_BOOL includeRuntimeInstances,  
                        [out, retval] BSTR* projectTreeContent );
```

```
Function GetProjectTree (path As String, _  
                        Depth As Long, _  
                        IncludeRuntimeInstances As Boolean) As  
String
```

Description

GetProjectTree is a general method for retrieving information about the current project or parts of the current project as an XML-string. The returned content depends on the filter parameters **path**, **depth** and **includeRuntimeInstances**.

Note! The method returns hierarchical information about the pieces (objects) of the project, not complete information.

Example of returned information: Information about the objects names and types and how the objects relate to each other. Information about general data as GUID, protection, visibility, scope, interaction window and so on.

Example of **omitted** (i.e. not included in the returned XML-string) information: Internal description of types such as parameters, variables, codeblocks and so on.

The parameter **path** decides from which object in the project tree the method shall start. If the path is an empty string, the method will start from the project root. The path is somewhat different from the regular paths in Open Interface, and some additions are needed to be able to navigate through the entire project tree:

The first part of the path (if it isn't blank) must be one of the keywords:

- **Libraries**
- **HardwareLibraries**
- **Applications**
- **Controllers**

Otherwise you will not be able to tell which part of the tree you're referring to.

To be able to separate the different types and instances in **Applications** and **Libraries**, you have to use one of the keywords:

- **Functions** (only for library)
- **DataTypes**
- **FunctionBlockTypes**
- **ControlModuleTypes**
- **DiagramTypes**

- **ControlModules** (only for application)
- **Diagrams** (only for application)
- **Programs** (only for application)

To list hardware types in **HardwareLibraries** use the keyword:

- **HardwareTypes**

To separate the different parts in the **Controllers** part of the project tree, you use one of the keywords:

- **HWUnits**
- **Tasks**
- **ConnectedApplications**
- **ConnectedHardwareLibraries**

The parameter **depth** decides how many levels of the tree that should be returned by the method. If **depth** is equal to 0, then only the level pointed to by the path will be returned.

Level descriptions:

- One level down from the project root returns a list of all the libraries, applications and controllers.
- One level down from a library returns all the types in the library.
- One level down from an application returns all the types, control modules, single control modules and programs that lie directly in that application.
- One level down from a type returns one level of instances, i.e. control modules, single control modules and function blocks.
- One level down from an instance returns another level of instances. This also applies to single control modules.

Depth = -1 means you would like to receive the entire sub-tree.

The parameter **includeRuntimeInstances** decides whether or not the method shall return the instances that "resides" inside other instances, i.e. all instances that lies more than one level beneath a type, application or single control module, if the parameter **depth** goes that deep. If the parameter **includeRuntimeInstances** is false, it is only the first instance level that will be returned. In this case single control modules are regarded as types and not instances, which means that you are able to go as deep as you like in a tree with single control modules.

Example 1:

Assume **path** = "Libraries.MyLibrary.FunctionBlockTypes" and **depth** = 1 and **includeRuntimeInstances** = false. The **GetProjectTree** method will then return an XML description containing information about all function block types belonging to the library "MyLibrary". Furthermore, information about the function block type's sub objects will also be included in the returned XML description.

Example 2:

Assume **path** = "Libraries.MyLibrary" and **depth** = 1 and **includeRuntimeInstances** = false. The **GetProjectTree** method will then return an XML description containing information about all function

block types, all data types, and all control module types belonging to the library “MyLibrary”. Since **depth** = 1 no information about sub objects for the types are returned.

Example 3:

Assume **path** = “” and **depth** = 1 and **includeRuntimeInstances** = false. The **GetProjectTree** method will then return an XML description containing information about all libraries, all applications and all Controllers. Since **depth** =1 no information about sub objects of the libraries etc will be returned.

Example 4:

Assume **path** = “HardwareLibraries.MyHWLibrary” and **depth** = 1 and **includeRuntimeInstances** = false. The **GetProjectTree** method will then return an XML description containing information about all hardware types belonging to the hardware library “MyHWLibrary”. Since **depth** = 1 the list of hardware types is also included.

Parameters

Name	Type	Description
Path	String	The parameter path specifies the starting point. Example: path= “Libraries.MyLibrary.FunctionBlockTypes” says that you would like to receive information about all function block types in the library MyLibrary. See description above. Some other examples: <ul style="list-style-type: none">• “Applications.MyApplication.ControlModules”• “Applications.MyApplication.Programs”• “Applications.MyApplication.ControlModuleTypes”• “Applications.MyApplication.DataTypes”• “Libraries.MyLibrary.DiagramTypes”
Depth	Long	The parameter depth specifies the depth of the information. Depth = 0 means you would like to receive information for the level specified by the path parameter. Depth = 1 means you would like to receive information for the level specified by the path parameter and one additional level down and so on. Depth = -1 means you would like to receive the whole sub-tree.
includeRuntimeInstances	Boolean	If this parameter is true information about instances in instances is returned. This kind of information is needed by external project explorer in Online or TestMode mode. This parameter only affects POU instances and does not affect hardware libraries or controllers.

Return value

An XML-string describing the current project or parts of the current project is returned. The returned information depends on the filter parameters. Note! The method returns hierarchical information about the pieces of the project, not complete information. See [GetProjectTree XML example](#)

5.1.29.4 GetTypePathFromGuid

```
HRESULT GetTypePathFromGuid([in] BSTR appOrLibName,  
                             [in] BSTR guid,  
                             [out,retval] BSTR* typePath);
```

```
Function GetTypePathFromGuid(appOrLibName As String,  
                             guid As String) As String
```

Description

Retrieves the path to the type with the guid **guid**. If the application or library where the type is declared is known, the name can be entered as **appOrLibName**. If the application or library is unknown, just leave **appOrLibName** blank, and the whole project will be searched. This method will find the following types:

- DataTypes
- FunctionBlockTypes
- ControlModuleTypes
- SingleControlModules
- DiagramTypes
- Diagrams
- Programs

Parameters

Name	Type	Description
appOrLibName	String	Optional parameter. Name of the application or library where the type one is searching for is declared. Example: "MyApplication"
guid	String	The GUID of the type one is searching for.

Return value

The method returns the path to the type searched for.

Example: "Application1 1.2-3.MyFBType"

Example: "Library1 2.0-7.MyCMType.MySCM"

5.1.29.5 **GetProjectAndEnvironmentInformation**

```
HRESULT GetProjectAndEnvironmentInformation(  
    [out] BSTR* projectName,  
    [out] BSTR* projectGuid,  
    [out] BSTR* environmentName,  
    [out] BSTR* environmentGuid);  
  
Sub GetProjectAndEnvironmentInformation (  
    projectName As String, _  
    projectGuid As String, _  
    environmentName As String, _  
    environmentGuid As String)
```

Description

GetProjectAndEnvironmentInformation is a general method for retrieving information about the current project and which environment it is currently using. It will retrieve the name and guid of the current project, and if running CB Professional it will also retrieve the name and guid of the current environment.

Parameters

Name	Type	Description
projectName	String	Out parameter containing the name of the currently loaded project.
projectGuid	String	Out parameter containing the GUID of the currently loaded project.
environmentName	String	Out parameter containing the name of the currently loaded environment. Only valid for CB Professional. In other versions, a blank string is returned.
environmentGuid	String	Out parameter containing the GUID of the currently loaded environment. Only valid for CB Professional. In other versions, a blank string is returned.

Return value

Returns and error if no project is loaded.

5.1.30 Methods for reservation of File Organization Units

See also section [Handling of Source Distribution and Source Code Reservation](#)

5.1.30.1 Reserve

```
HRESULT Reserve([in] BSTR fouName);
```

```
Sub Reserve(fouName As String)
```

Description

There is no reservation mechanism in the current CB, but this method exists for possible future use.

The method will always fail and return the errorcode E_NOTIMPL.

Parameters

Name	Type	Description
fouName	String	

5.1.30.2 IsReservedBy

```
HRESULT IsReservedBy([in] BSTR fouName, [out, retval] BSTR*  
reserver);
```

Sub IsReservedBy(fouName As String) As String

Description

There is no reservation mechanism in the current CB, but this method exists for possible future use.
The method will always fail and return the errorcode E_NOTIMPL.

Parameters

Name	Type	Description
fouName	String	

Return value

E_NOTIMPL

5.1.30.3 ReleaseReservation

HRESULT ReleaseReservation([in] BSTR fouName);

Sub ReleaseReservation(fouName As String)

Description

There is no reservation mechanism in the current CB, but this method exists for possible future use.
The method will always fail and return the errorcode E_NOTIMPL.

Parameters

Name	Type	Description
fouName	String	

Return value

E_NOTIMPL

5.1.31 Methods for settings

The SetSetting and GetSetting methods are two general methods. These methods can be used in order to influence some settings on a name value basis. The settings are defined on the COM Object basis so one clients setting cannot affect the behavior for another client.

Settings:

Name	Type	Valid values	Default value	Description
Reduced Check	boolean	true, false	false	If set to true the checks performed on set and new-methods will not check code blocks for the objects affected. This will speed up the methods. See performance . This will not mark the checked object as 'checked'. It will require a complete check later (during online or similar).

5.1.31.1 GetSetting

```
HRESULT GetSetting([in] BSTR settingName,  
                  [out, retval] VARIANT* theValue);
```

```
Function GetSetting(settingName As String) As Variant
```

Description

Returns the value of the actual setting.

Parameters

Name	Type	Description
settingName	String	The name of the setting (the property)

Return Value

The value for the actual setting (property)

Examples of some very useful predefined settings

settingName	Return Value
WorkingFolder	The file path to the CBM workingfolder. Example1: C:\ABB Industrial IT Data\Engineer IT Data\Control Builder M Professional Example2: C:\ABB Industrial IT Data\Engineer IT Data\Compact Control Builder AC 800M

ProjectsFolder	<p>The path to the folder holding the projects:</p> <p>Example1: C:\ABB Industrial IT Data\Engineer IT Data\Control Builder M Professional\Projects</p> <p>Example2: C:\ABB Industrial IT Data\Engineer IT Data\Compact Control Builder AC 800M\Projects</p>
HWLibrariesFolder	<p>When running CBM Professional, it will return the path to the hardware libraries cache. When running Compact CB it will return the path to where Control Builder's standard hardware libraries are installed.</p> <p>Example1: C:\ABB Industrial IT Data\Engineer IT Data\Control Builder M Professional\HWLibraries</p> <p>Example2: C:\Program Files\ABB Industrial IT\Engineer IT\Compact Control Builder AC 800M\HWLibraries</p>
ControlBuilderType	<p>Control Builder type (or name)</p> <p>Example1: Control Builder M Professional</p> <p>Example2: Compact Control Builder AC 800M</p>
ControlBuilderVersion	<p>Control Builder version</p> <p>Example: 5.0.1</p>
ControlBuilderBuildVersion	<p>Control Builder build version</p> <p>Example: 5.0.1004.3</p>
InstallationFolder	<p>The path to where the Control Builder M is installed</p> <p>Example1: C:\Program Files\ABB Industrial IT\Engineer IT\Control Builder M Professional 5.0</p> <p>Example2: C:\Program Files\ABB Industrial IT\Engineer IT\Compact Control Builder AC 800M</p>

5.1.31.2 SetSetting

```
HRESULT SetSetting([in] BSTR settingName,  
[in] VARIANT theValue);
```

```
Sub SetSetting(settingName As String, _  
theValue As Variant)
```

Description

Set the value of the actual setting.

Parameters

Name	Type	Description
------	------	-------------

SettingName	String	The name of the setting (the property)
TheValue	Variant	The new setting (property) value

5.1.32 Methods for Parameters

There are several types of parameters that can be used. To select what kind of parameter to use, the first parameter named **typeOfParameter** of the type **enum CBOpenIFParameterType** can have one of the following values:

- OI_CMPARAMETER
- OI_EXTENSIBLEPARAMETER
- OI_PARAMETER

5.1.32.1 Parameter XML example

XML example:

```
<?xml version="1.0" encoding="UTF-16"?>
<Parameter xmlns="CBOpenIFSchema3_0" Name="Par1" Type="dint" Attribute="retain"
Direction="in" InitialValue="7" ReadPermission="" WritePermission=""
AuthenticationLevel="None" TypePath="System.dint">
  <Description>Description of Par1</Description>
</Parameter>
```

For further information see the [XML Schema](#) and [semantics for omitted attributes and elements](#).

See also [What XML elements and attributes are needed by Set and New – methods?](#)

5.1.32.2 NewParameter

```
HRESULT NewParameter([in] enum CBOpenIFParameterType typeOfParameter,  
                    [in] BSTR parameterName,  
                    [in] BSTR 196atatype,  
                    [in] BSTR pathToParent,  
                    [in] BSTR parameterContent,  
                    [out, retval] BSTR* messages);
```

```
Function NewParameter(typeOfParameter As CBOpenIFParameterType, _  
                    parameterName As String,  
                    196atatype As String,  
                    pathToParent As String,  
                    parameterContent As String) As String
```

Description

Creates a new parameter with the name **parameterName** and of the type **196atatype**. The parameter **parameterContent** is optional. It sets the content of the parameter using an XML-string. If it is empty, an empty parameter will be created. If content is provided the “Name” and “Type” attributes in the XML-string are ignored since the parameter **parameterName** is used as the 196atatype196 name and the parameter **196atatype** is used as the parameter type name.

Parameters

Name	Type	Description
typeOfParameter	enum CBOpenIFParameterType	Decides what kind of parameter to create.
parameterName	String	The name of the parameter to create.
dataType	String	The data type the created parameter should have.
pathToParent	String	Path to the object where to put the new parameter. Example: “MyLib.MyFBType”
parameterContent	String	Optional parameter. A string containing an XML description of the new parameter. See parameter XML example . If this string is empty, an empty parameter will be created.

Return value

A [message bucket](#) is returned.

5.1.32.3 GetParameter

```
HRESULT GetParameter([in] enum CBOpenIFParameterType typeOfParameter,  
                    [in] BSTR parameterPath,  
                    [out, retval] BSTR* parameterContent);
```

```
Function GetParameter(typeOfParameter As CBOpenIFParameterType, _  
                    parameterPath As String) As String
```

Description

Returns the content of the parameter specified by **parameterPath**, as an XML-string.

Parameters

Name	Type	Description
typeOfParameter	enum CBOpenIFParameterType	Decides what kind of parameter to get.
parameterPath	String	Path to the parameter to retrieve information from. Example: "MyLibrary.MyFBType.MyPar".

Return value

A string containing an XML description of the parameter is returned. See [parameter XML example](#).

5.1.32.4 SetParameter

```
HRESULT SetParameter([in] enum COpenIFParameterType typeOfParameter,  
                    [in] BSTR parameterPath,  
                    [in] BSTR parameterContent,  
                    [out, retval] BSTR* messages);
```

```
Function SetParameter(typeOfParameter As COpenIFParameterType, _  
                    parameterPath As String, _  
                    parameterContent As String) As String
```

Description

Sets the content of the parameter specified by **parameterPath** using an XML-string as the parameter **parameterContent**. You can change the name and the type of the parameter using the XML.

Parameters

Name	Type	Description
typeOfParameter	enum COpenIFParameterType	Decides what kind of parameter to set.
parameterPath	String	Path to the parameter to set information for. Example: "MyLibrary.MyFBType.MyPar"
parameterContent	String	The information to set for the selected parameter as an XML-string. See parameter XML example .

Return value

A [message bucket](#) is returned.

5.1.32.5 DeleteParameter

```
HRESULT DeleteParameter(  
    [in] enum CBOpenIFParameterType  
    typeOfParameter,  
    [in] BSTR parameterPath);  
  
Sub DeleteParameter(typeOfParameter As CBOpenIFParameterType, _  
    parameterPath As String)
```

Description

Deletes the parameter specified by **parameterPath** from the project.

Parameters

Name	Type	Description
typeOfParameter	enum CBOpenIFParameterType	Decides what kind of parameter to delete.
parameterPath	String	Path to the parameter to delete. Example: "MyLibrary.MyFBType.MyPar"

5.1.33 Methods for Variables

There are several types of variables that can be used. To select what kind of variables to use, the first parameter named **typeOfVariable** of the type **enum CBOpenIFVariableType** can have one of the following values:

- OI_EXTERNALVARIABLE
- OI_GLOBALVARIABLE
- OI_VARIABLE
- OI_COMMUNICATIONVARIABLE

5.1.33.1 Variables XML example

XML example:

```
<?xml version="1.0" encoding="utf-16"?>
<Variable xmlns="CBOpenIFSchema3_0" Name="Var1" Type="real" Attribute="retain"
InitialValue="7.0" ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.real">
  <Description>Description of Var1</Description>
</Variable>
```

For further information see the [XML Schema](#) and [semantics for omitted attributes and elements](#).

See also [What XML elements and attributes are needed by Set and New – methods?](#)

5.1.33.2 NewVariable

```
HRESULT NewVariable([in] enum CBOpenIFVariableType typeOfVariable,  
                    [in] BSTR variableName,  
                    [in] BSTR 201atatype,  
                    [in] BSTR pathToParent,  
                    [in] BSTR variableContent,  
                    [out, retval] BSTR* messages);
```

```
Function NewVariable(typeOfVariable As CBOpenIFVariableType, _  
                    variableName As String, _  
                    201atatype As String, _  
                    pathToParent As String, _  
                    variableContent As String) As String
```

Description

Creates a new variable with the name **variableName** and of the type **201atatype**. The parameter **variableContent** is optional. It sets the content of the variable using an XML-string. If it is empty, an empty variable will be created. If content is provided the “Name” and “Type” attributes in the XML-string are ignored since the parameter **variableName** is used as the variable name and the parameter **diagramType** is used as the variable type name

Parameters

Name	Type	Description
typeOfVariable	enum CBOpenIFVariableType	Determines what kind of variable to create.
variableName	String	The name of the variable to create.
dataType	String	The data type the created variable should have.
pathToParent	String	Path to the object where to put the new variable. Example: “MyLib.MyFBType”
variableContent	String	Optional parameter. A string containing an XML description of the new variable. See variable XML example . If this string is empty, an empty variable will be created.

Return value

A [message bucket](#) is returned.

5.1.33.3 GetVariable

```
HRESULT GetVariable([in] enum CBOpenIFVariableType typeOfVariable,  
                    [in] BSTR variablePath,  
                    [out, retval] BSTR* variableContent);
```

```
Function GetVariable(typeOfVariable As CBOpenIFVariableType, _  
                    variablePath As String) As String
```

Description

Returns the content of the variable specified by **variablePath**, as an XML-string.

Parameters

Name	Type	Description
typeOfVariable	enum CBOpenIFVariableType	Determines what kind of variable to get.
variablePath	String	Path to the variable to retrieve information from. Example: "MyLibrary.MyFBType.MyVar".

Return value

A string containing an XML description of the variable is returned. See [variable XML example](#).

5.1.33.4 SetVariable

```
HRESULT SetVariable([in] enum CBOpenIFVariableType typeOfVariable,  
                    [in] BSTR variablePath,  
                    [in] BSTR variableContent,  
                    [out, retval] BSTR* messages);
```

```
Function SetVariable(typeOfVariable As CBOpenIFVariableType, _  
                    variablePath As String, _  
                    variableContent As String) As String
```

Description

Sets the content of the variable specified by **variablePath** using an XML-string as the parameter **variableContent**. You can change the name and the type of the variable using the XML.

Parameters

Name	Type	Description
typeOfVariable	enum CBOpenIFVariableType	Determines what kind of variable to set.
variablePath	String	Path to the variable to set information for. Example: "MyLibrary.MyFBType.MyVar"
variableContent	String	The information to set for the selected variable as an XML-string. See variable XML example .

Return value

A [message bucket](#) is returned.

5.1.33.5 DeleteVariable

```
HRESULT DeleteVariable([in] enum CBOpenIFVariableType typeOfVariable,  
                        [in] BSTR variablePath);
```

```
Sub DeleteVariable(typeOfVariable As CBOpenIFVariableType, _  
                  variablePath As String)
```

Description

Deletes the variable specified by **variablePath** from the project.

Parameters

Name	Type	Description
typeOfVariable	enum CBOpenIFVariableType	Determines what kind of variable to delete.
variablePath	String	Path to the variable to delete. Example: "MyLibrary.MyFBType.MyVar"

5.1.34 Methods for Signals

To select what kind of signal to use, the first parameter named **typeOfSignal** of the type **enum CBOpenIFSignalType** can have one of the following values:

- OI_SIGNAL

5.1.34.1 Signals XML example

XML example:

```
<?xml version="1.0" encoding="utf-16"?>
<Signal xmlns="CBOpenIFSchema3_0" Name="Signal1" Path="MyApp.Var1" Direction="in"
AcknowledgeGroup="auto">
  <Description>Description of Signal1</Description>
</Signal>
```

For further information see the [XML Schema](#) and [semantics for omitted attributes and elements](#).

See also [What XML elements and attributes are needed by Set and New – methods?](#)

5.1.34.2 NewSignal

```
HRESULT NewSignal([in] enum CBOpenIFSignalType typeOfSignal,  
                  [in] BSTR signalName,  
                  [in] BSTR pathToParent,  
                  [in] BSTR signalContent,  
                  [out, retval] BSTR* messages);
```

```
Function NewSignal(typeOfSignal As CBOpenIFSignalType, _  
                  signalName As String, _  
                  pathToParent As String, _  
                  signalContent As String) As String
```

Description

Creates a new signal with the name **signalName**. The parameter **signalContent** is optional. It sets the content of the signal using an XML-string. If it is empty, an empty signal will be created. If content is provided the “Name” attribute in the XML-string is ignored since the parameter **signalName** is used as the signal name. Signals can only be created in applications, programs, and diagrams.

Parameters

Name	Type	Description
typeOfSignal	enum CBOpenIFSignalType	Determines what kind of signal to create.
signalName	String	The name of the signal to create.
pathToParent	String	Path to the object where to put the new signal. Example: “MyApp.MyProgram”
signalContent	String	Optional parameter. A string containing an XML description of the new signal. See signal XML example . If this string is empty, an empty signal will be created.

Return value

A [message bucket](#) is returned.

5.1.34.3 GetSignal

```
HRESULT GetSignal([in] enum COpenIFSignalType typeOfSignal,  
                  [in] BSTR signalName,  
                  [in] BSTR signalContent,  
                  [out, retval] BSTR* signalContent);
```

```
Function GetSignal(typeOfSignal As COpenIFSignalType, _  
                  signalName As String, _  
                  pathToParent As String) As String
```

Description

Returns the content of the signal specified by **signalName** and **pathToParent** as an XML-string. Signals are only available in applications, programs, and diagrams.

Parameters

Name	Type	Description
typeOfSignal	enum COpenIFSignalType	Determines what kind of signal to get.
signalName	String	Name of the signal to retrieve information from.
pathToParent	String	Path to the object where the signal is declared. Example: "MyApp.MyProgram"

Return value

A string containing an XML description of the signal is returned. See [signal XML example](#).

5.1.34.4 SetSignal

```
HRESULT SetSignal([in] enum COpenIFSignalType typeOfSignal,  
                  [in] BSTR signalName,  
                  [in] BSTR pathToParent,  
                  [in] BSTR signalContent,  
                  [out, retval] BSTR* messages);
```

```
Function SetSignal(typeOfSignal As COpenIFSignalType, _  
                  signalName As String, _  
                  pathToParent As String, _  
                  signalContent As String) As String
```

Description

Sets the content of the signal specified by **signalName** and **pathToParent** using an XML-string since the parameter **signalContent**. You can change the name and other attributes of the signal using the XML. Signals are only available in applications, programs, and diagrams.

Parameters

Name	Type	Description
typeOfSignal	enum COpenIFSignalType	Determines what kind of signal to set.
signalName	String	Name of the signal to set information for.
pathToParent	String	Path to the object where the signal is declared. Example: "MyApp.MyProgram"
signalContent	String	The information to set for the selected signal as an XML-string. See signal XML example .

Return value

A [message bucket](#) is returned.

5.1.34.5 DeleteSignal

```
HRESULT DeleteSignal([in] enum COpenIFSignalType typeOfSignal,  
                    [in] BSTR signalName,  
                    [in] BSTR pathToParent);
```

```
Sub DeleteSignal(typeOfSignal As COpenIFSignalType, _  
                signalName As String, _  
                pathToParent As String)
```

Description

Deletes the signal specified by **signalName** and **pathToParent** from the project.

Parameters

Name	Type	Description
typeOfSignal	enum COpenIFSignalType	Determines what kind of signal to delete.
signalName	String	Name of the signal to delete.
pathToParent	String	Path to the object where the signal is declared. Example: "MyApp.MyProgram"

5.1.35 Methods for CodeBlocks

There are several types of code blocks that can be used. To select what kind of code blocks to use, the first parameter named **typeOfCodeBlock** of the type **enum CBOpenIFCodeBlockType** can have one of the following values:

- OI_FBD (Not in Diagram Types and Diagrams)
- OI_IL (Not in Diagram Types and Diagrams)
- OI_LD (Not in Diagram Types and Diagrams)
- OI_SFC
- OI_ST
- OI_FD (Only in Diagram Types and Diagrams)

5.1.35.1 CodeBlocks XML example

XML example:

```
<?xml version="1.0" encoding="UTF-16"?>
<STCodeBlock xmlns="CBOpenIFSchema3_0" Name="CodeBlock2">
  <ST_Code>
    Var1:=Var2+1;
  </ST_Code>
</STCodeBlock>
```

For further information see the [XML Schema](#) and [semantics for omitted attributes and elements](#).

See also [What XML elements and attributes are needed by Set and New – methods?](#)

5.1.35.2 NewCodeBlock

```
HRESULT NewCodeBlock([in] enum CBOpenIFCodeBlockType typeOfCodeBlock,  
                    [in] BSTR codeBlockName,  
                    [in] BSTR pathToParent,  
                    [in] BSTR codeBlockContent,  
                    [out, retval] BSTR* messages);
```

```
Function NewCodeBlock(typeOfCodeBlock As CBOpenIFCodeBlockType, _  
                    codeBlockName As String, _  
                    pathToParent As String, _  
                    codeBlockContent As String) As String
```

Description

Creates a new code block with the name **codeBlockName** of the type **typeOfCodeBlock**. The parameter **codeBlockContent** is optional. It sets the content of the code block using an XML-string. If it is empty, an empty code block will be created. If content is provided the “Name” attribute in the XML-string is ignored since the parameter **codeBlockName** is used as the code block name.

Parameters

Name	Type	Description
typeOfCodeBlock	enum CBOpenIFCodeBlockType	Determines what kind of code block to create.
codeBlockName	String	The name of the code block to create.
pathToParent	String	Path to the object where to put the new code block. Example: “MyLib.MyFBType”
codeBlockContent	String	Optional parameter. A string containing an XML description of the new code block. See code block XML example . If this string is empty, an empty code block will be created.

Return value

A [message bucket](#) is returned.

5.1.35.3 GetCodeBlock

```
HRESULT GetCodeBlock([in] BSTR codeBlockPath,  
                     [out, retval] BSTR* codeBlockContent);
```

Function GetCodeBlock(codeBlockPath As String) As String

Description

Returns the content of the code block specified by **codeBlockPath**, as an XML-string.

Parameters

Name	Type	Description
codeBlockPath	String	Path to the code block to retrieve information from. Example: “MyLibrary.MyFBType.MyCode”.

Return value

A string containing an XML description of the code block is returned. See [code block XML example](#).

5.1.35.4 SetCodeBlock

```
HRESULT SetCodeBlock([in] BSTR codeBlockPath,  
                    [in] BSTR codeBlockContent,  
                    [out, retval] BSTR* messages);
```

```
Function SetCodeBlock(codeBlockPath As String, _  
                    codeBlockContent As String) As String
```

Description

Sets the content of the code block specified by **codeBlockPath** using an XML-string as the parameter **codeBlockContent**. You can change the name of the code block using the XML.

Parameters

Name	Type	Description
codeBlockPath	String	Path to the code block to set information for. Example: "MyLibrary.MyFBType.MyCode"
codeBlockContent	String	The information to set for the selected code block as an XML-string. See code block XML example .

Return value

A [message bucket](#) is returned.

5.1.35.5 DeleteCodeBlock

```
HRESULT DeleteCodeBlock([in] BSTR codeBlockPath);
```

```
Sub DeleteCodeBlock(codeBlockPath As String)
```

Description

Deletes the code block specified by **codeBlockPath** from the project.

Parameters

Name	Type	Description
codeBlockPath	String	Path to the code block to delete. Example: “MyLibrary.MyFBType.MyCode”

5.1.36 Methods for CMConnections

5.1.36.1 CMConnection XML example

XML example:

```
<?xml version="1.0" encoding="UTF-16"?>
<CMConnection xmlns="CBOpenIFSschema3_0" Name="Par1" ActualParameter="GlobVar1"
GraphicalConnection="0"/>
```

For further information see the [XML Schema](#) and [semantics for omitted attributes and elements](#).

See also [What XML elements and attributes are needed by Set and New – methods?](#)

5.1.36.2 GetCMConnection

```
HRESULT GetCMConnection([in] BSTR connectionPath,
                        [out, retval] BSTR* connectionContent);
```

```
Function GetCMConnection(connectionPath As String) As String
```

Description

Returns the content of the control module connection, specified by **connectionPath**, as an XML-string.

Parameters

Name	Type	Description
connectionPath	String	Path to the connection to retrieve information from. Example: "MyApplication.MyCM.MyConnection". <i>Note! The path can contain several instance levels.</i> <i>Example: "MyApplic.CMT1.CMInst1.CMInst2.Par1"</i>

Return value

A string containing an XML description of the instance is returned. See [control module connection XML example](#).

5.1.36.3 SetCMConnection

```
HRESULT SetCMConnection([in] BSTR connectionPath,  
                        [in] BSTR connectionContent,  
                        [out, retval] BSTR* messages);
```

```
Function SetCMConnection(connectionPath As String, _  
                        connectionContent As String) As String
```

Description

Sets the content of the control module connection specified by **connectionPath** using an XML-string as the parameter **connectionContent**.

Parameters

Name	Type	Description
connectionPath	String	Path to the connection to set information for. Example: "MyApplication.MyCM.MyConnection" <i>Note! The path can contain several instance levels.</i> <i>Example:</i> <i>MyApplic.CMT1.CMInst1.CMInst2.Par1"</i>
connectionContent	String	The information to set for the selected connection as an XML-string. See control module connection XML example .

Return value

A [message bucket](#) is returned.

5.1.37 Methods for FDConnections

5.1.37.1 FDConnection XML example

XML example:

```
<?xml version="1.0" encoding="UTF-16"?>
<FDConnection xmlns="CBOpenIFSchema3_0" Name="QuotaExc" ActualParameter="BoolVar1"/>
```

Attribute	Type	Description
Name	String	Name of the parameter.
ActualParameter	String	Name of a variable, parameter or port connected to the parameter defined by Name above.

For further information see the [XML Schema](#) and [semantics for omitted attributes and elements](#).

See also [What XML elements and attributes are needed by Set and New – methods?](#)

5.1.37.2 GetFDConnection

```
HRESULT GetFDConnection([in] BSTR connectionPath,
                        [out, retval] BSTR* connectionContent);
```

```
Function GetFDConnection(connectionPath As String) As String
```

Description

Returns the content of the connection in an FD codeblock, specified by **connectionPath**, as an XML-string.

Parameters

Name	Type	Description
connectionPath	String	Path to the connection to retrieve information from. Example: "MyApplication.MyDiagram.MyInstance.MyParameter". <i>Note! The path can contain several instance levels.</i> <i>Example: "MyApplic.Diagram1.CMInst1.CMInst2.Par1"</i>

Return value

A string containing an XML description of the instance is returned. See [Function Diagram connection XML example](#).

5.1.37.3 SetFDConnection

```
HRESULT SetFDConnection([in] BSTR connectionPath,  
                        [in] BSTR connectionContent,  
                        [out, retval] BSTR* messages);
```

```
Function SetFDConnection(connectionPath As String, _  
                        connectionContent As String) As String
```

Description

Sets the content of the connection in an FD codeblock specified by **connectionPath** using an XML-string as the parameter **connectionContent**.

Parameters

Name	Type	Description
connectionPath	String	Path to the connection to set information for. Example: "MyApplication.MyDiagram..MyInstance.MyParameter" <i>Note! The path can contain several instance levels.</i> <i>Example:</i> <i>MyApplic.Diagram1.CMInst1.CMInst2.Par1"</i>
connectionContent	String	The information to set for the selected connection as an XML-string. See Function Diagram connection XML example .

Return value

A [message bucket](#) is returned.

5.1.38 Methods for Connected Libraries

5.1.38.1 Connected Libraries XML example

XML example:

```
<?xml version="1.0" encoding="UTF-16"?>
<ConnectedLibraries xmlns="CBOpenIFSSchema3_0">
  <ConnectedLibrary Name="MyLibrary" MajorVersion="1" MinorVersion="0" Revision="0"/>
  <ConnectedLibrary Name="ControlStandardLib" MajorVersion="1" MinorVersion="0"
Revision="0"/>
</ConnectedLibraries>
```

For further information see the [XML Schema](#) and [semantics for omitted attributes and elements](#).

See also [What XML elements and attributes are needed by Set and New – methods?](#)

5.1.38.2 GetConnectedLibraries

```
HRESULT GetConnectedLibraries([in] BSTR appOrLibName,
                              [out,retval] BSTR*
                              connectedLibrariesContent);
```

```
Function GetConnectedLibraries(appOrLibName As String) As String
```

Description

Returns the connected libraries for the application or library specified by **appOrLibName**, as an XML-string.

Parameters

Name	Type	Description
appOrLibName	String	Name of the application or library to retrieve information from. Example: "MyApplication".

Return value

A string containing an XML description of the connected library is returned. See [connected libraries XML example](#).

5.1.38.3 SetConnectedLibraries

```
HRESULT SetConnectedLibraries([in] BSTR appOrLibName,  
                              [in] BSTR connectedLibrariesContent,  
                              [out,retval] BSTR* messages);
```

```
Function SetConnectedLibraries (appOrLibName As String, _  
                               connectedLibrariesContent As String _  
                               ) As String
```

Description

Sets the connected libraries for the application or library specified by **appOrLibName** using an XML-string as the parameter **connectedLibrariesContent**.

Parameters

Name	Type	Description
appOrLibName	String	Name of the application or library to set the connected libraries for. Example: "MyApplication"
connectedLibrariesContent	String	The information to set for the selected application or library as an XML-string. See connected libraries XML example .

Return value

A [message bucket](#) is returned.

5.1.38.4 ConnectLibrary

```
HRESULT ConnectLibrary([in] BSTR appOrLibName,  
                        [in] BSTR libraryToConnect);
```

```
Sub ConnectLibrary(appOrLibName As String, libraryToConnect As  
String)
```

Description

Connects the library **libraryToConnect** to **appOrLibName**.

Parameters

Name	Type	Description
appOrLibName	String	Name of the application or library to add the connection to. Example: "MyApplication"
libraryToConnect	String	The name and version of the library to connect. The version number of the library must be specified. Example: "Library1 1.2-3"

5.1.38.5 DisconnectLibrary

```
HRESULT DisconnectLibrary([in] BSTR appOrLibName,
                          [in] BSTR libraryToDisconnect);

Sub DisconnectLibrary(appOrLibName As String, _
                      libraryToDisconnect As String)
```

Description

Disconnects the library **libraryToDisconnect** from **appOrLibName**.

Parameters

Name	Type	Description
appOrLibName	String	Name of the application or library to remove the connection from. Example: “MyApplication”
libraryToConnect	String	The name and version of the library to disconnect. The version number of the library is optional. Example: “Library1”

5.1.38.6 ReplaceConnectedLibrary

```
HRESULT ReplaceConnectedLibrary([in] BSTR controllerName,  
                                [in] BSTR connectedLibraryName,  
                                [in] BSTR replacingLibraryName);
```

```
Sub ReplaceConnectedLibrary(controllerName As String, _  
                            connectedLibraryName As String, _  
                            replacingLibraryName As String)
```

Description

Replaces the library **connectedLibraryName**, which is connected to the controller **controllerName**, with the library **replacingLibraryName**.

Parameters

Name	Type	Description
controllerName	String	Name of the controller in which to replace a connected library. Example: "MyController"
connectedLibraryName	String	The name and version of the connected library to be replaced. The version number of the library is mandatory. Example: "MyLibrary 1.0-0"
replacingLibraryName	String	The name and version of the replacing library. The version number of the library is mandatory. Example: "MyLibrary 1.0-1"

5.1.39 Methods for Connected Hardware Libraries

5.1.39.1 Connected Hardware Libraries XML example

XML example:

```
<?xml version="1.0" encoding="UTF-16"?>
<ConnectedHardwareLibraries xmlns="CBOpenIFSchema3_0">
  <ConnectedHardwareLibrary Name="BasicHWLib" MajorVersion="1" MinorVersion="0"
Revision="0"/>
  <ConnectedHardwareLibrary Name="MyHWLibrary" MajorVersion="1" MinorVersion="0"
Revision="0"/>
</ConnectedHardwareLibraries>
```

For further information see the [XML Schema](#) and [semantics for omitted attributes and elements](#).

See also [What XML elements and attributes are needed by Set and New – methods?](#)

5.1.39.2 GetConnectedHardwareLibraries

```
HRESULT GetConnectedHardwareLibraries([in] BSTR controllerName,
                                       [out, retval] BSTR*
connectedHardwareLibrariesContent);
```

```
Function GetConnectedHardwareLibraries(controllerName As String _
                                       ) As String
```

Description

Returns the connected hardware libraries for the controller specified by **controllerName**, as an XML-string.

Parameters

Name	Type	Description
controllerName	String	Name of the controller to retrieve information from. Example: "MyController".

Return value

A string containing an XML description of the connected hardware library is returned. See [connected hardware libraries XML example](#).

5.1.39.3 SetConnectedHardwareLibraries

```
HRESULT SetConnectedHardwareLibraries([in] BSTR controllerName,  
[in] BSTR  
  
connectedHardwareLibrariesContent,  
[out,retval] BSTR* messages);
```

```
Function SetConnectedHardwareLibraries (controllerName As String, _  
connectedHardwareLibrariesContent As String  
—  
) As String
```

Description

Sets the connected hardware libraries for the controller specified by **controllerName** using an XML-string as the parameter **connectedHardwareLibrariesContent**.

Parameters

Name	Type	Description
controllerName	String	Name of the controller to set the connected hardware libraries for. Example: "MyController"
connectedHardwareLibrariesContent	String	The information to set for the selected controller as an XML-string. See connected libraries XML example .

Return value

A [message bucket](#) is returned.

5.1.39.4 **ConnectHardwareLibrary**

```
HRESULT ConnectHardwareLibrary([in] BSTR controllerName,  
                               [in] BSTR hardwareLibraryToConnect);  
  
Sub ConnectHardwareLibrary(controllerName As String, _  
                           hardwareLibraryToConnect As String)
```

Description

Connects the hardware library **hardwareLibraryToConnect** to the controller **controllerName**.

Parameters

Name	Type	Description
controllerName	String	Name of the controller to add the connection to. Example: "MyController"
hardwareLibraryToConnect	String	The name and version of the hardware library to connect. The version number of the hardware library must be specified. Example: "HWLibrary1 1.2-3"

5.1.39.5 DisconnectHardwareLibrary

```
HRESULT DisconnectHardwareLibrary([in] BSTR controllerName,  
                                   [in] BSTR  
                                   hardwareLibraryToDisconnect);  
  
Sub DisconnectHardwareLibrary(controllerName As String, _  
                               hardwareLibraryToDisconnect As String)
```

Description

Disconnects the hardware library **hardwareLibraryToDisconnect** from the controller **controllerName**.

Parameters

Name	Type	Description
controllerName	String	Name of the controller to remove the connection from. Example: "MyController"
hardwareLibraryToConnect	String	The name and version of the hardware library to disconnect. The version number of the hardware library is optional. Example: "HWLibrary1"

5.1.39.6 ReplaceConnectedHardwareLibrary

```
HRESULT ReplaceConnectedHardwareLibrary([in] BSTR controllerName,
                                         [in] BSTR
                                         connectedLibraryName,
                                         [in] BSTR
                                         replacingLibraryName);

Sub ReplaceConnectedHardwareLibrary(controllerName As String, _
                                   connectedHardwareLibraryName As String,
                                   _
                                   replacingHardwareLibraryName As
                                   String)
```

Description

Replaces the hardware library **connectedHardwareLibraryName**, which is connected to the controller **controllerName**, with the hardware library **replacingHardwareLibraryName**.

Parameters

Name	Type	Description
controllerName	String	Name of the controller in which to replace a connected hardware library. Example: "MyController"
connectedHardwareLibraryName	String	The name and version of the connected hardware library to be replaced. The version number of the hardware library is mandatory. Example: "MyHWLibrary 1.0-0"
replacingHardwareLibraryName	String	The name and version of the replacing hardware library. The version number of the hardware library is mandatory. Example: "MyHWLibrary 1.0-1"

5.1.40 Methods for Version and State

5.1.40.1 SetLibraryVersion

```
HRESULT SetLibraryVersion([in] BSTR libraryName,  
                           [in] LONG newMajorVersion,  
                           [in] LONG newMinorVersion,  
                           [in] LONG newRevision);
```

```
Sub SetLibraryVersion(libraryName As String, _  
                      newMajorVersion As Long, _  
                      newMinorVersion As Long, _  
                      newRevision As Long)
```

Description

Sets a new version for the library **libraryName**. The new version must have a higher version number than the old one.

Parameters

Name	Type	Description
libraryName	String	Name of the library to set a new version for. Example: "MyLibrary"
newMajorVersion	Long	The new major version number. (1 to 32767)
newMinorVersion	Long	The new minor version number. (0 to 255)
newRevision	Long	The new revision number. (0 to 255)

5.1.40.2 SetHardwareLibraryVersion

```
HRESULT SetHardwareLibraryVersion([in] BSTR hardwareLibraryName,  
                                   [in] LONG newMajorVersion,  
                                   [in] LONG newMinorVersion,  
                                   [in] LONG newRevision);
```

```
Sub SetHardwareLibraryVersion(hardwareLibraryName As String, _  
                              newMajorVersion As Long, _  
                              newMinorVersion As Long, _  
                              newRevision As Long)
```

Description

Sets a new version for the hardware library **hardwareLibraryName**. The new version must have a higher version number than the old one.

Parameters

Name	Type	Description
hardwareLibraryName	String	Name of the hardware library to set a new version for. Example: "MyHWLibrary"
newMajorVersion	Long	The new major version number. (1 to 32767)
newMinorVersion	Long	The new minor version number. (0 to 255)
newRevision	Long	The new revision number. (0 to 255)

5.1.40.3 SetApplicationVersion

```
HRESULT SetApplicationVersion([in] BSTR applicationName,  
                             [in] LONG newMajorVersion,  
                             [in] LONG newMinorVersion,  
                             [in] LONG newRevision);
```

```
Sub SetApplicationVersion(applicationName As String, _  
                          newMajorVersion As Long, _  
                          newMinorVersion As Long, _  
                          newRevision As Long)
```

Description

Not implemented yet.

Parameters

Name	Type	Description
applicationName	String	Name of the application to set a new version for. Example: "MyApplication"
newMajorVersion	Long	The new major version number. (1 to 32767)
newMinorVersion	Long	The new minor version number. (0 to 255)
newRevision	Long	The new revision number. (0 to 255)

5.1.40.4 SetControllerVersion

```
HRESULT SetControllerVersion([in] BSTR controllerName,  
                             [in] LONG newMajorVersion,  
                             [in] LONG newMinorVersion,  
                             [in] LONG newRevision);
```

```
Sub SetControllerVersion(controllerName As String, _  
                        newMajorVersion As Long, _  
                        newMinorVersion As Long, _  
                        newRevision As Long)
```

Description

Not implemented yet.

Parameters

Name	Type	Description
controllerName	String	Name of the controller to set a new version for. Example: "MyController"
newMajorVersion	Long	The new major version number. (1 to 32767)
newMinorVersion	Long	The new minor version number. (0 to 255)
newRevision	Long	The new revision number. (0 to 255)

5.1.40.5 GetLibraryState

```
HRESULT GetLibraryState([in] BSTR libraryName,  
                        [out,retval] BSTR* libraryState);
```

```
Function GetLibraryState(libraryName As String) As String
```

Description

Returns the state of the library **libraryName**. The state is one of three values: Open, Closed or Released.

- Open – The library and its content is open to changes.
- Closed – The library and its content can not be changed, but it can be reopened again.
- Released – The library and its content can no longer be changed, and can not be reopened.

Parameters

Name	Type	Description
libraryName	String	Name of the library to change state of. Example: "MyLibrary".

Return value

A string containing the state of the library.

5.1.40.6 **SetLibraryState**

```
HRESULT SetLibraryState([in] BSTR libraryName,  
                        [in] BSTR libraryState);  
  
Sub SetLibraryState(libraryName As String, libraryState As String)
```

Description

Sets the state of the library named **libraryName** to the state **libraryState**.

Parameters

Name	Type	Description
libraryName	String	Name of the library to change state of. Example: “MyLibrary”
libraryState	String	The new state to set the library to. Valid values are: Open, Closed and Released

5.1.40.7 GetHardwareLibraryState

```
HRESULT GetHardwareLibraryState([in] BSTR hardwareLibraryName,  
                                [out,retval] BSTR*  
hardwareLibraryState);  
  
Function GetHardwareLibraryState(hardwareLibraryName As String _  
                                ) As String
```

Description

Returns the state of the hardware library **hardwareLibraryName**. The state is one of three values: Open, Closed or Released.

- Open – The hardware library and its content is open to changes.
- Closed – The hardware library and its content can not be changed, but it can be reopened again.
- Released – The hardware library and its content can no longer be changed, and can not be reopened.

Parameters

Name	Type	Description
hardwareLibraryName	String	Name of the hardware library to change state of. Example: "MyHWLibrary".

Return value

A string containing the state of the hardware library.

5.1.40.8 SetHardwareLibraryState

```
HRESULT SetHardwareLibraryState([in] BSTR hardwareLibraryName,  
                                [in] BSTR hardwareLibraryState);
```

```
Sub SetHardwareLibraryState(hardwareLibraryName As String, _  
                            hardwareLibraryState As String)
```

Description

Sets the state of the hardware library named **hardwareLibraryName** to the state **hardwareLibraryState**.

Parameters

Name	Type	Description
hardwareLibraryName	String	Name of the hardware library to change state of. Example: "MyHWLibrary"
hardwareLibraryState	String	The new state to set the hardware library to. Valid values are: Open, Closed and Released

5.1.41 Methods for Refresh

5.1.41.1 RefreshProject

HRESULT RefreshProject() ;

Sub RefreshProject()

Description

Refreshes the entire project from source code. This operation can take a long time to complete.

Parameters

None.

5.1.41.2 RefreshLibrary

HRESULT RefreshLibrary(BSTR libraryName);

Sub RefreshLibrary(libraryName As String)

Description

Refreshes the entire library named **libraryName** from source code. This operation can take a long time to complete.

Parameters

Name	Type	Description
libraryName	String	Name of the library to refresh. Example: "MyLibrary"

5.1.41.3 RefreshHardwareLibrary

```
HRESULT RefreshHardwareLibrary(BSTR hardwareLibraryName) ;
```

```
Sub RefreshHardwareLibrary(hardwareLibraryName As String)
```

Description

Refreshes the entire hardware library named **hardwareLibraryName** from source code. This operation can take a long time to complete.

Parameters

Name	Type	Description
hardwareLibraryName	String	Name of the hardware library to refresh. Example: "MyHWLibrary"

5.1.41.4 RefreshApplication

```
HRESULT RefreshApplication(BSTR applicationName) ;
```

```
Sub RefreshApplication(applicationName As String)
```

Description

Refreshes the entire application named **applicationName** from source code. This operation can take a long time to complete.

Parameters

Name	Type	Description
applicationName	String	Name of the application to refresh. Example: "MyApplication"

5.1.41.5 RefreshController

```
HRESULT RefreshController(BSTR controllerName) ;
```

```
Sub RefreshController(controllerName As String)
```

Description

Refreshes the entire controller named **controllerName** from source code. This operation can take a long time to complete.

Parameters

Name	Type	Description
controllerName	String	Name of the controller to refresh. Example: "MyController"


5.1.42 Methods for writing in the CBM Session log

5.1.42.1 WriteInformation

```
HRESULT WriteInformation([in] BSTR message) ;
```

```
Sub WriteInformation(message As String)
```

Description

	ABB AB	Doc. no.	3BSE033313	Lang.	en	Rev. ind.	Q	Page	240

The method prints an Information message in the CBM session log. WriteInformation is used for informative messages, e.g. the CBM has downloaded an application to a controller. The layout of a message will be “I Date_and_time: message\n”.

Parameters

Name	Type	Description
message	String	The message to be printed

5.1.42.2 WriteWarning

```
HRESULT WriteWarning([in] BSTR message);
Sub WriteWarning(message As String)
```

Description

The method prints a Warning message in the CBM session log. WriteWarning is used when something unexpected occurred. A warning is an indication that something went wrong, but we will try to recover and continue executing.

The layout of a message will be “W Date_and_time: message\n”.

Parameters

Name	Type	Description
message	String	The message to be printed

5.1.42.3 WriteError

```
HRESULT WriteError([in] BSTR message);
Sub WriteError(message As String)
```

Description

The method prints an Error message in the CBM session log. WriteError is used when an unexpected error occurred. The CBM will probably stop executing after this. The layout of a message will be “E Date_and_time: message\n”.

Parameters

Name	Type	Description
message	String	The message to be printed

5.1.43 XML Examples for code blocks written in various languages

5.1.43.1 Code blocks written in structured text (ST)

Structured text is described as plain text embedded as a node value of a `<ST_Code>` XML element.

Structured text XML example:

```
<CodeBlocks>
  <STCodeBlock Name="CodeBlock1">
    <ST_Code>
      if Var1 &gt; 0 then
        FBInst1( Parl := Var1 );
        FBInst2( Parl := Var2 );
      end_if;
    </ST_Code>
  </STCodeBlock>
  <STCodeBlock Name="CodeBlock2">
    <ST_Code>
      Var1:=Var2+1;
    </ST_Code>
  </STCodeBlock>
</CodeBlocks>
```

For further information see the [XML Schema](#) and [semantics for omitted attributes and elements](#).

5.1.43.2 Code blocks written in sequential function chart (SFC)

Sequential function chart is described by various XML elements according to the example below.

```
<CodeBlocks>
  <SFCCodeBlock Name="Code" SeqControl="1" StepElapsedTime="0" SFCViewerAspect="0">
    <SFCStep Name="S1" InitialStep="1">
      <Pl_Action>
        <ST_Code>a:=1+1;</ST_Code>
      </Pl_Action>
    </SFCStep>
    <SFCTransition Name="Tr1">True</SFCTransition>
    <SFCSimultaneous>
      <SFCBranch>
        <SFCStep Name="S2" InitialStep="0"/>
      </SFCBranch>
      <SFCBranch>
        <SFCStep Name="S3" InitialStep="0"/>
      </SFCBranch>
    </SFCSimultaneous>
    <SFCTransition Name="Tr2">True</SFCTransition>
    <SFCStep Name="S4" InitialStep="0"/>
    <SFCSelection>
      <SFCBranch>
        <SFCTransition Name="Tr3">True</SFCTransition>
      </SFCBranch>
      <SFCBranch>
        <SFCTransition Name="Tr4" Dest="S1">True</SFCTransition>
      </SFCBranch>
    </SFCSelection>
    <SFCSubSequence SubSeqName="Subseq">
      <SFCStep Name="S5" InitialStep="0"/>
      <SFCTransition Name="Tr5">True</SFCTransition>
    </SFCSubSequence>
  </SFCCodeBlock>
</CodeBlocks>
```

For further information see the [XML Schema](#) and [semantics for omitted attributes and elements](#).

The example above corresponds to the following SFC code block displayed in a POU Editor:

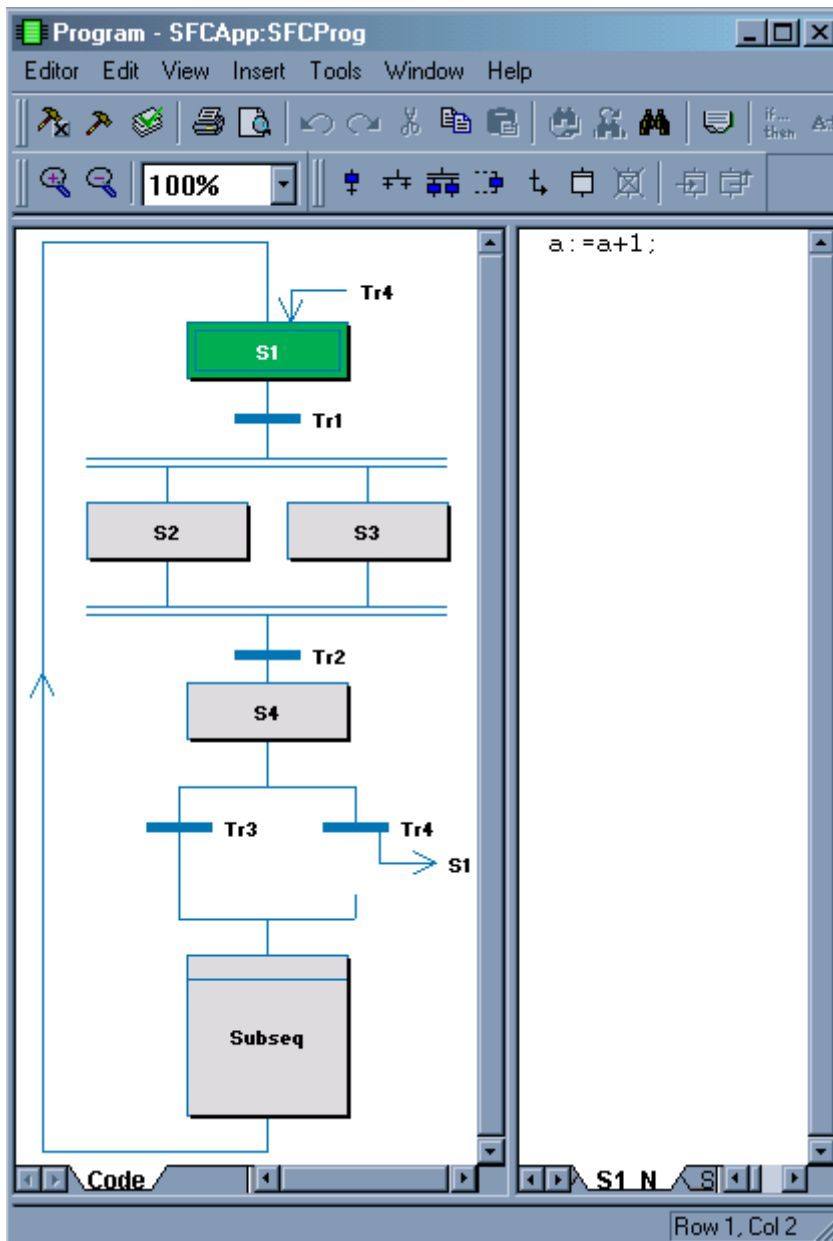


Figure 13

5.1.43.3 Code blocks written in instruction list (IL)

Instruction list is described by various XML elements according to the example below.

Instruction list XML example:

```
<CodeBlocks>
  <ILCodeBlock Name="Code">
    <ILRow>
      <Instruction>LD</Instruction>
      <Operand>Level</Operand>
      <Description>Load Level and</Description>
    </ILRow>
    <ILRow>
      <Instruction>GT</Instruction>
      <Operand>1000</Operand>
      <Description>test if Level &gt; 1000</Description>
    </ILRow>
    <ILRow>
      <Instruction>JMPC</Instruction>
      <Operand>HighLevel</Operand>
      <Description>Jump to HighLevel if so</Description>
    </ILRow>
    <ILRow>
      <Instruction>LD</Instruction>
      <Operand>True</Operand>
      <Description>Load True</Description>
    </ILRow>
    <ILRow>
      <Instruction>ST</Instruction>
      <Operand>OpenValve</Operand>
      <Description>Store as OpenValve</Description>
    </ILRow>
    <ILRow>
      <Instruction>JMP</Instruction>
      <Operand>AfterAlarm</Operand>
    </ILRow>
    <ILRow>
      <Label>HighLevel</Label>
      <Instruction>LD</Instruction>
      <Operand>True</Operand>
      <Description>Load True</Description>
    </ILRow>
    <ILRow>
      <Instruction>ST</Instruction>
      <Operand>LevelTooHigh</Operand>
    </ILRow>
    <ILRow>
      <Label>AfterAlarm</Label>
    </ILRow>
  </ILCodeBlock>
</CodeBlocks>
```

For further information see the [XML Schema](#) and [semantics for omitted attributes and elements](#).

The example above corresponds to the following IL code block displayed in a POU Editor.

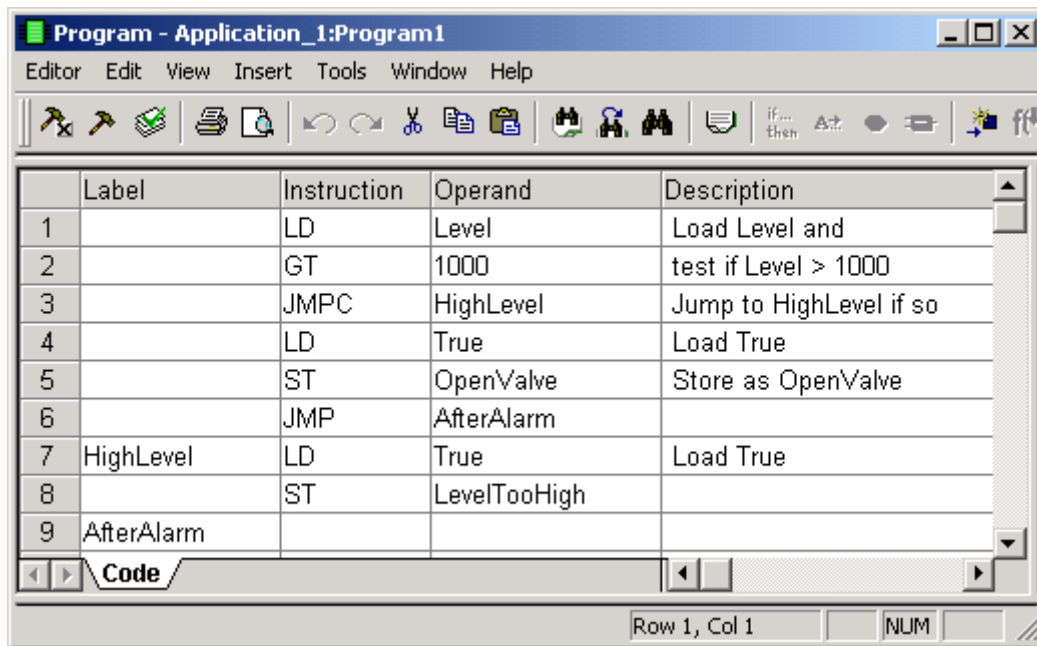


Figure 14

5.1.43.4 Code blocks written in function block diagram (FBD)

Function block diagram is described as plain text embedded as a node value of a **<ST_Code>** element where the father element is a **<FBDCodeBlock>** element.

Note! A higher level of XML description is desirable because it is not possible to understand the “plain ST-code text with comments” according to the example below. I hope we can provide a better XML description in a future version!

FBD XML example:

```
<?xml version="1.0" encoding="UTF-16"?>
<Program xmlns="CBOpenIFSchema3_0" Name="FBDProg" TaskConnection="NewCont.Slow"
SILLevel="NonSIL" SimulationMark="0">
  <Variables>
    <Variable Name="a" Type="bool" Attribute="retain" InitialValue=""
ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.bool"/>
    <Variable Name="b" Type="bool" Attribute="retain" InitialValue=""
ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.bool"/>
    <Variable Name="c" Type="bool" Attribute="retain" InitialValue=""
ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.bool"/>
    <Variable Name="d" Type="bool" Attribute="retain" InitialValue=""
ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.bool"/>
    <Variable Name="t2" Type="time" Attribute="retain" InitialValue=""
ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.time"/>
    <Variable Name="t1" Type="time" Attribute="retain" InitialValue=""
ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.time"/>
    <Variable Name="__FBD_Code_7" Type="bool" Attribute="" InitialValue=""
ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.bool"/>
    <Variable Name="__FBD_Code_9" Type="bool" Attribute="" InitialValue=""
ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.bool"/>
    <Variable Name="__FBD_Code_11" Type="bool" Attribute="" InitialValue=""
ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.bool"/>
  </Variables>
  <FunctionBlocks>
    <FunctionBlock Name="Ton_1" Type="Ton" TaskConnection=""/>
  </FunctionBlocks>
  <CodeBlocks>
    <FBDCodeBlock Name="FBD_Code">
      <ST_Code>
        (*$VER 1 *)
        (*$L F7 Plain A4 Portrait English*)
        __FBD_Code_9(*$AC 1*)(*$ID 32769*) := (*$P 1 *)
        (*$CP bool,2*)
        (*$ID 3*) AND (IN1 := a,
IN2 := b);
        __FBD_Code_11(*$AC 1*)(*$ID 32770*) := (*$CP bool,2*)
        (*$ID 4*) AND (IN1 := c,
IN2 := d);
        __FBD_Code_7(*$AC 1*)(*$ID 32771*) := (*$CP bool,2*)
        (*$ID 2*) OR (IN1 := __FBD_Code_9,
IN2 := __FBD_Code_11);
        (*$ID 1*) Ton_1(In := __FBD_Code_7,
PT := t1,
```

```

ET => t2) ;
</ST_Code>
</FBDCodeBlock>
</CodeBlocks>
</Program>

```

Note! Newline characters are significant within the `<ST_Code>` text.

For further information see the [XML Schema](#) and [semantics for omitted attributes and elements](#).

The example above corresponds to the following FBD code block displayed in a POU Editor.

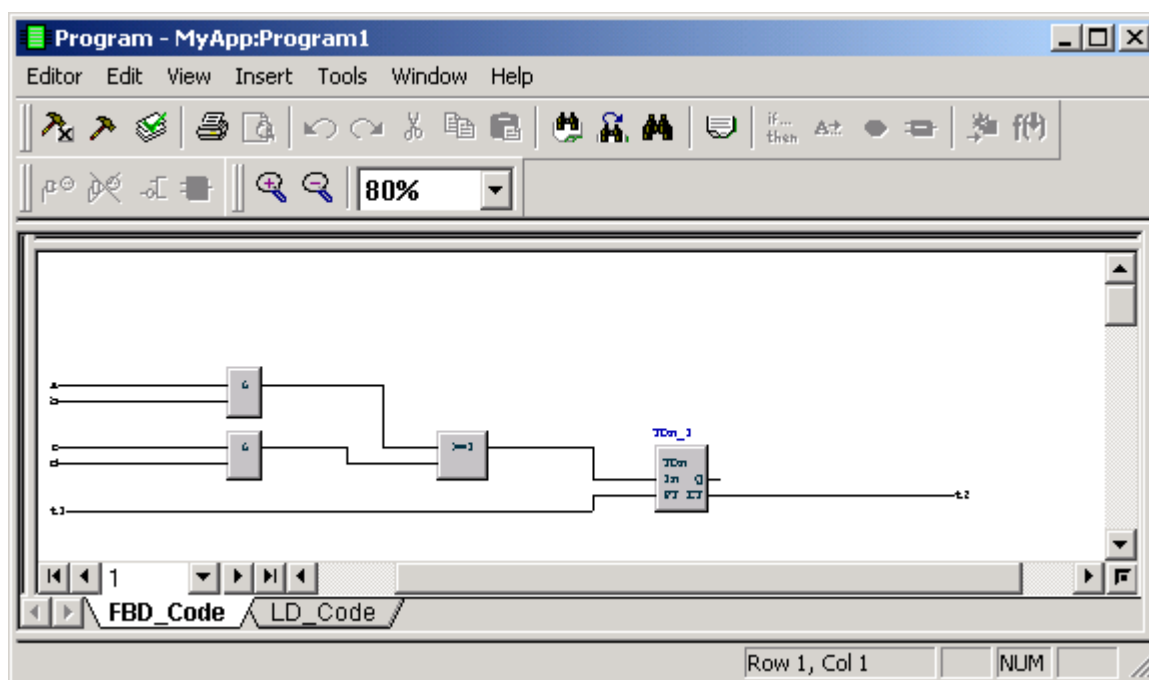


Figure 15

5.1.43.5 Code blocks written in ladder diagram (LD)

Ladder diagram is described as plain text embedded as a node value of a `<ST_Code>` element where the father element is a `<LDCodeBlock>` element.

Note! A higher level of XML description is desirable because it is not possible to understand the “plain ST-code text with comments” according to the example below. I hope we can provide a better XML description in a future version!

LD XML example:

```
<?xml version="1.0" encoding="UTF-16"?>
<Program xmlns="CBOpenIFSschema3_0" Name="LDProg" TaskConnection="" SILLevel="NonSIL"
SimulationMark="0">
  <Variables>
    <Variable Name="a" Type="bool" Attribute="retain" InitialValue=""
ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.bool"/>
    <Variable Name="b" Type="bool" Attribute="retain" InitialValue=""
ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.bool"/>
    <Variable Name="c" Type="bool" Attribute="retain" InitialValue=""
ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.bool"/>
    <Variable Name="d" Type="bool" Attribute="retain" InitialValue=""
ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.bool"/>
    <Variable Name="t2" Type="time" Attribute="retain" InitialValue=""
ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.time"/>
    <Variable Name="t1" Type="time" Attribute="retain" InitialValue=""
ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.time"/>
    <Variable Name="trig1" Type="bool" Attribute="retain" InitialValue=""
ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.bool"/>
    <Variable Name="trig2" Type="bool" Attribute="retain" InitialValue=""
ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.bool"/>
    <Variable Name="__LD_Code_1" Type="bool" Attribute="" InitialValue=""
ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.bool"/>
    <Variable Name="__LD_Code_2" Type="bool" Attribute="" InitialValue=""
ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.bool"/>
    <Variable Name="__LD_Code_6" Type="bool" Attribute="" InitialValue=""
ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.bool"/>
    <Variable Name="__LD_Code_5" Type="bool" Attribute="" InitialValue=""
ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.bool"/>
    <Variable Name="__LD_Code_3" Type="bool" Attribute="" InitialValue=""
ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.bool"/>
    <Variable Name="__LD_Code_7" Type="bool" Attribute="" InitialValue=""
ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.bool"/>
    <Variable Name="__LD_Code_12" Type="bool" Attribute="" InitialValue=""
ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.bool"/>
```

```

    <Variable Name="__LD_Code_14" Type="bool" Attribute="" InitialValue=""
ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.bool"/>
    <Variable Name="__LD_Code_10" Type="bool" Attribute="" InitialValue=""
ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.bool"/>
    <Variable Name="__LD_Code_4" Type="bool" Attribute="" InitialValue=""
ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.bool"/>
</Variables>
<FunctionBlocks>
    <FunctionBlock Name="Ton_2" Type="Ton" TaskConnection=""/>
</FunctionBlocks>
<CodeBlocks>
    <LDCodeBlock Name="LD_Code">
        <ST_Code>
            (*$VER 1 *)
            (*$L F7 Plain A4 Portrait English*)
            __LD_Code_2(*$AC 1*)(*$ID 32769*) := (*$P 1 *)
            (*$C LD *)
            (*$R 1 *)
            (*$ID 3*) MOVE ((*$IC INVISIBLE CONTACT*)IN1 := TRUE);
            __LD_Code_5(*$AC 1*)(*$ID 32770*) := (*$C LD *)
            (*$CP bool,2*)
            (*$ID 6*) AND (IN1 := __LD_Code_2,
            (*$IC INVISIBLE CONTACT*)IN2 := TRUE);
            __LD_Code_14(*$AC 1*)(*$ID 32771*) := (*$C LD *)
            (*$CP bool,2*)
            (*$ID 10*) AND (IN1 := __LD_Code_5,
            IN2 := (*$CCF CONTACT*)a);
            __LD_Code_3(*$AC 1*)(*$ID 32772*) := (*$C LD *)
            (*$CP bool,2*)
            (*$ID 5*) AND (IN1 := __LD_Code_14,
            IN2 := (*$CCF CONTACT*)b);
            __LD_Code_12(*$AC 1*)(*$ID 32773*) := (*$C LD *)
            (*$CP bool,2*)
            (*$ID 9*) AND (IN1 := __LD_Code_5,
            IN2 := (*$CCF CONTACT*)c);
            __LD_Code_7(*$AC 1*)(*$ID 32774*) := (*$C LD *)
            (*$CP bool,2*)
            (*$ID 8*) AND (IN1 := __LD_Code_12,
            IN2 := (*$CCF CONTACT*)d);
            __LD_Code_6(*$AC 1*)(*$ID 32775*) := (*$C LD *)
            (*$CP bool,2*)
            (*$ID 7*) OR (IN1 := __LD_Code_3,
            IN2 := __LD_Code_7);
            (*$CCF COIL*)trig1(*$AC 1*)(*$ID 32776*) := (*$C LD *)
            (*$ID 4*) MOVE (IN1 := __LD_Code_6);
            __LD_Code_1(*$AC 1*)(*$ID 32777*) := (*$C LD *)
            (*$R 2 *)
            (*$ID 1*) MOVE ((*$IC INVISIBLE CONTACT*)IN1 := TRUE);
            __LD_Code_4(*$AC 1*)(*$ID 32778*) := (*$C LD *)
            (*$CP bool,2*)
            (*$ID 11*) AND (IN1 := __LD_Code_1,
            IN2 := (*$CCF CONTACT*)trig1);
            (*$EN __LD_Code_4*)
            (*$ENO __LD_Code_10*)
            (*$ID 12*) Ton_2 (In := trig1,
            PT := t1,
            ET = > t2) ;
            (*$CCF COIL*)trig2(*$AC 1*)(*$ID 32779*) := (*$C LD *)
            (*$ID 2*) MOVE (IN1 := __LD_Code_10);
        </ST_Code>
    </LDCodeBlock>

```

```
</CodeBlocks>
</Program>
```

Note! Newline characters are significant within the `<ST_Code>` text.
For further information see the [XML Schema](#) and [semantics for omitted attributes and elements](#).

The example above corresponds to the following LD code block displayed in a POU Editor.

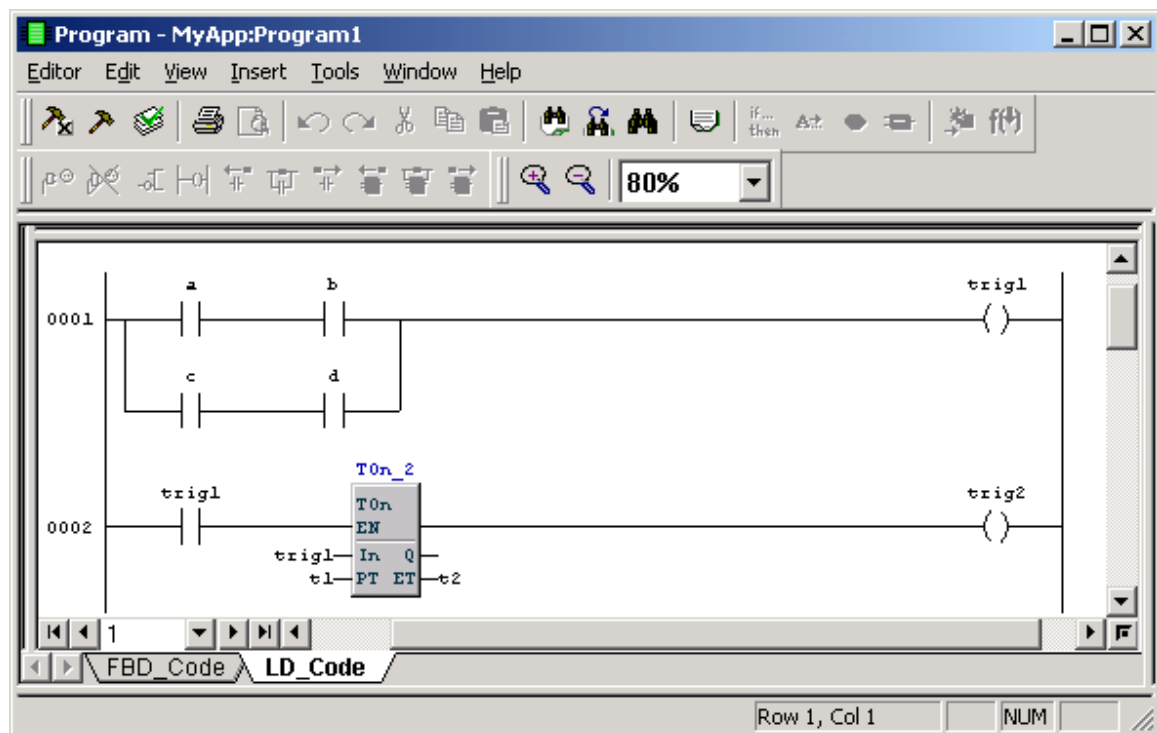


Figure 16

5.1.44 Some additional remarks

5.1.44.1 Project Explorer update

The tree view in the Project Explorer will be updated with new or deleted items. However no attempt will be made to keep an expanded branch correctly updated. The user may have to collapse and expand the tree to see the changes.

5.1.44.2 Dialogs

All Control Builder dialogs will normally be suppressed during an Open Interface method call. ***But some rare error cases can display dialogs.*** Such examples are:

- File Organization Unit methods ***may*** issue an error dialog on out of disk space or other disk write problems (e.g. write protected). If this happens the COM call will hang, and requires manual intervention.
- If the Control Builder Professional runs out of memory during a call, an error dialog will be displayed. This will make the COM call hang, and requires manual intervention.

5.1.44.3 File Organization Units

Examples of “File Organization Units” are Project, Libraries, Applications and Controllers. That is a unit, which can be saved on a file.

1. All File Organization Unit methods will remove leading and trailing spaces, tabs and newline characters from directory specifications.
2. It is possible to use an UNC path as directory specification. However there is currently a limitation that means that the directory must be created manually, before it is used.
3. When calling New-methods on OpenIF in CB Professional (where all FOU:s are stored in PPA) directoryPath can be left blank since it is irrelevant in that case.
4. When calling Insert or Open-methods on OpenIF in CB Professional replace the filepath with the GUID of the object to insert or open.

5.1.44.4 The Protected attribute of POU types and Data types

POU types and Data types have a *protected* attribute. If the attribute is *set* the following applies:

- **Delete** methods, for a protected type, will fail and the type will not be deleted
- **Set** methods, for a protected type, will fail and the type will be unchanged.
- **Get** methods, for a protected type, will not return a full description of the type. Instead an XML string describing the external interface (general information + parameters + extensible parameters) of the type is returned. See the example below!

Example of a returned XML description for a function block type with the protection attribute set:

```
<?xml version="1.0" encoding="UTF-16"?>
<FunctionBlockType xmlns="CBOpenIFSchema3_0" Name="MyProtectedFBType" Protected="1"
Hidden="0" Scope="public" InteractionWindow="" AlarmOwner="0" SILLevel="NonSIL"
SimulationMark="0">
  <Parameters>
    <Parameter Name="Par1" Type="dint" Attribute="retain" Direction="in"
InitialValue="" ReadPermission="" WritePermission="" AuthenticationLevel="None"
TypePath="System.dint"/>
  </Parameters>
  <ExtensibleParameters/>
  <Description>Description of the Protected FBType</Description>
</FunctionBlockType>
```

Compare with the XML description if the protected attribute is *not* set. See [function block XML example](#).

Note! An interactive user can override the protection. In this case Open Interface methods will behave just as if the protection attribute was *not* set but **Get methods** will return the correct value of the property `Protected="1"` in the XML description. See example above.

5.1.44.5 Password for application units and library units

Applications and libraries can be password protected. This functionality cannot be modeled by the means of Open Interface. This section discusses what will happen if an interactive user manually sets a password, for an application or library, and the interactive user doesn't enter the password:

- *Delete* methods, for types and instances, will fail and the type (or the instance) will not be deleted
- *Set* methods will fail and the type (or the instance) will be unchanged.
- *Get* methods will return a full description of the type (or the instance).
- *New* will fail.

5.1.44.6 What XML elements and attributes are needed by Set and New – methods?

The [XML Schema](#) contains all necessary information but I will give you some hints:

1. Attributes can be specified to be **required** (use="required") or **optional**. Optional attributes can be omitted in the XML document (string).
2. Attributes can be specified to have **default** values. Example: **default="retain"**. Such attributes can be omitted, in the XML document (string), if you accept the default value.
3. Elements can be specified to be optional through the schema setting **minOccurs="0"**. Such elements can be omitted.

Assume that a client would like to specify an XML document (string) according to the previous [function block type XML example](#) and use this XML-string as an actual parameter to the **NewFunctionBlockType** or **SetFunctionBlockType** methods. See also [semantics for omitted attributes and elements](#).

The following shorter XML document is equivalent to the previous mentioned example:

```
<?xml version="1.0" encoding="UTF-16"?>
<FunctionBlockType Name="MyFBType" xmlns="CBOpenIFSchema3_0">
  <Parameters>
    <Parameter Name="Par1" Type="dint" InitialValue="7">
      <Description>Description of Par1</Description>
    </Parameter>
    <Parameter Name="Par2" Type="CalendarStruct" Direction="in_out"/>
  </Parameters>
  <Variables>
    <Variable Name="Var1" Type="real" InitialValue="7.0">
      <Description>Description of Var1</Description>
    </Variable>
  </Variables>
  <ExternalVariables>
    <ExternalVariable Name= »ExtVar1 » Type= »dint » Attribute= »constant »/>
  </ExternalVariables>
  <FunctionBlocks>
    <FunctionBlock Name="FBInst1" Type="AnotherFBType" TaskConnection="NewCont.Fast">
      <Description>Description of the instance</Description>
    </FunctionBlock>
  </FunctionBlocks>
  <CodeBlocks>
    <STCodeBlock Name="MySTCodeBlock">
      <ST_Code>if Par1>0 then FBInst1( ParA := ExtVar1 ); end_if;</ST_Code>
    </STCodeBlock>
  </CodeBlocks>
  <Description>Description of the FBType: "MyFBType"</Description>
</FunctionBlockType>
```

5.1.44.7 Semantics for omitted attributes and elements

The chapter “[What XML elements and attributes are needed by Set and New – methods?](#)” discussed XML attributes having **default** values and optional XML Elements (**minOccurs=“0”**).

Assume a **New** or a **Set** method is called and the supplied XML document (String) contains omitted attributes (with default values) and/or omitted XML Elements. The normal followin of the Control Builder in this case is as follows:

Case	Control Builder action
Omitted XML attribute	The Control Builder will store the default value, according to the XML Schema , into its internal object storage.
Omitted XML Element	The Control Builder will remove the corresponding stuff from its internal object storage. Example: if the method SetFunctionBlockType is called and the XML-string doesn't contain any <Variables> element then all previous declared variables are discarded.

There are some exceptions to the rules above:

Case	Control Builder action
Omitted GUID attribute (or empty string value)	The Control Builder will retain the old value (Set methods) or automatically create a new GUID value (New –methods).
Omitted attributes and/or element for HWUnits	See the SetHardwareUnit method
Omitted “VAType” attribute	See the SetAccessVariables method

5.1.44.8 XML character stuffing

Some characters are in conflict with the extensible markup language (XML). Open Interface will therefore replace such characters according to the following tables.

Characters in XML Element node values will be replaced according to the following table:

Character	Replaced by
&	&
<	<
>	>

Characters in XML attribute values will be replaced according to the following table:

Character	Replaced by
&	&
<	<
>	>
”	"
,	'

5.1.44.9 Interaction with open editors

Open Interface “Set” methods can affect data that is also modifiable by the interactive editors in the Control Builder. If there is one or more open editor that edits the same data that is updated by a “Set” method, the editors will *not* be updated automatically.

Open Interface “Delete” methods will automatically *close* any editors displaying the deleted item and potential “not applied” content of the editors are lost.

Open Interface “Delete” methods for source code units (DeleteApplication, DeleteLibrary, DeleteController) will automatically *close* any editors displaying any item contained in the deleted source code unit. Potential “not applied” content of such editors are lost.

5.1.44.10 Available methods in Online and TestMode mode

Only “Get” methods and the “Offline” method will work in *Online* and *TestMode* mode. Other methods will return an error (HRESULT = OI_E_MODE) if they are called when the Control Builder is in Online or TestMode mode.

5.1.44.11 Security handling

Open Interface supports the access management of the Control Builder Professional.

The Open Interface methods act as the user currently using the Control Builder Professional.

5.1.44.12 Multiple client access

Multiple clients can use the “CB Open Interface” simultaneously although this is not recommended. Every method call is queued in the STA’s (Single Treaded Apartments) Windows message queue. Since the method calls are read and dispatched from the queue one at a time, this ensures that access is from a single tread at any one time.

Multiple simultaneously clients can of course destroy for each other. One client can, for instance, delete an application inserted by another client and so on. There is currently no lock mechanism implemented.

5.2 The XML Schema

The XML Schema contains the grammatical rules for XML elements and attributes defined for the Control Builder. All incoming strings containing XML documents are validated against the schema. Invalid XML documents are discarded.

The schema used by “CB Open Interface” is according to the W3C XML Schema standard (approved by W3C 2001-05-02).

Note! Earlier revisions of this document, and Alpha versions of ATLAS 0.44, use a so-called XDR (XML Data Reduced) schema. XDR is an early schema proposal submitted to W3C by Microsoft and other companies. The W3C XML Schema is incompatible with the XDR schema.

I strongly recommend you to study a book (or article) about XML Schema before you read the schema. One book “Professional XML 2nd Edition, ISBN 1-861005-05-9” is mentioned in the reference list.

The chapters “[What XML elements and attributes are needed by Set and New – methods?](#)” and [semantics for omitted attributes and elements](#) gives some hints about writing XML document for the Control Builder.

The XML Schema was included in previous versions of this document. Due to the large size of the schema file we have decide to not include the schema in this document. Instead a user-friendly HTML project named “CBOpenIFSchema3_0.html” is included on the [Control IT Installation media](#). See “[CBOpenIFSchema3_0.html](#)”.

5.2.1 The HTML project “CBOpenIFSchema3_0.html”

The figure below is a snapshot of the HTML project “CBOpenIFSchema3_0.html” included on the [Control IT Installation media](#).

The screenshot shows a web browser window with the address bar displaying the file path: `file:///C:/root/Atlas/Workplace/v045_SB31/ManuallyManaged/CBOpenIFSamples/CBOpenIFSchema3_0/...`. The browser window has a menu bar with File, Edit, View, Favorites, Tools, and Help. The main content area displays the project information for the 'DataType' element.

element DataType

diagram

namespace CBOpenIFSchema3_0

children **Components Description**

used by element **DataTypes**

attributes

Name	Type	Use	Default	Fixed	Annotati
Name	xs:string	required			
Protected	xs:boolean		0		
Hidden	xs:boolean		0		
Scope	xs:NMTOKEN		public		
Guid	xs:string				
ReservedByFunction	xs:string				

source

```
<xs:element name="DataType">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Components" minOccurs="0"/>
      <xs:element ref="Description" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="Name" type="xs:string" use="required"/>
    <xs:attribute name="Protected" type="xs:boolean" default="0"/>
    <xs:attribute name="Hidden" type="xs:boolean" default="0"/>
    <xs:attribute name="Scope" default="public"/>
    <xs:simpleType>
      <xs:restriction base="xs:NMTOKEN">
        <xs:enumeration value="private"/>
        <xs:enumeration value="public"/>
      </xs:restriction>
    </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="Guid" type="xs:string" default="" />
    <xs:attribute name="ReservedByFunction" type="xs:string" default="" />
  </xs:complexType>
</xs:element>
```

Figure 17

5.3 A Visual Basic 6.0 application sample using the CB Open Interface

I strongly recommend you to study a book (or article) about the XML DOM parser before you try to use the parser. One book “Professional XML 2nd Edition, ISBN 1-861005-05-9” is mentioned in the reference list.

Note! The [CB Object model](#) relieves the client from the knowledge of XML and XML DOM programming.

Step by step instruction

1. Start the Control Builder Professional.
2. Start Visual Basic 6.0 IDE.
3. Create a new “Standard EXE” application (File->New Project command).
4. On the Project menu, click the References command. Scroll down until you see the name “ControlBuilder 2.0 Type Library”. Click the check box next to this name and click Ok.
5. On the Project menu, click the References command. Scroll down until you see the name “Microsoft XML, v6.0”. Click the check box next to this name and click Ok.
6. On the Project menu, click the References command. The dialog should now look like figure V1. Then click on the Cancel button.

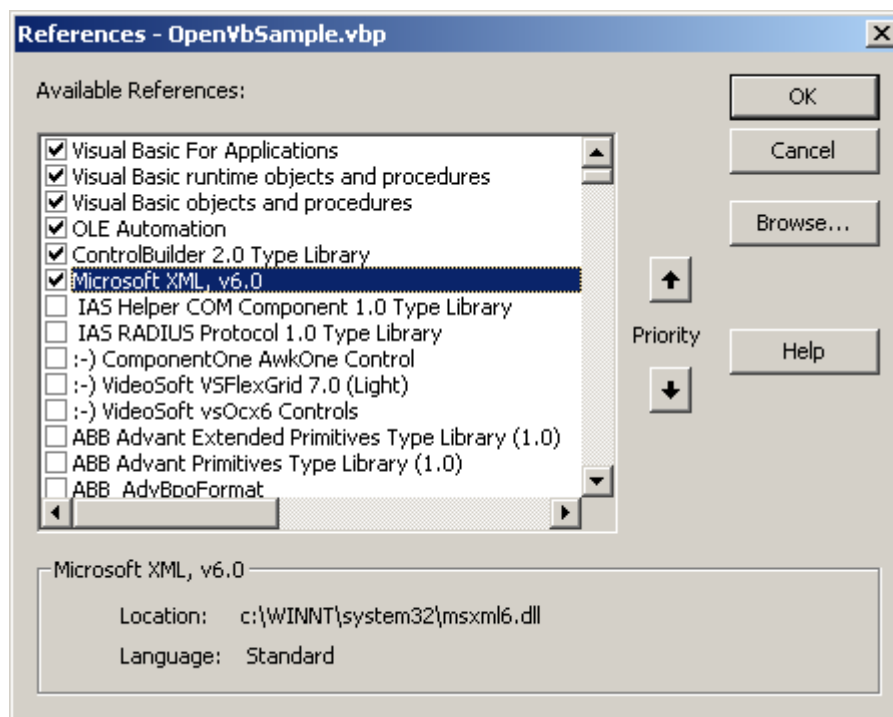


Figure V1.

7. Now the Visual Basic IDE has knowledge about the “Open Interface” and the Microsoft XML parser (MSXML). This can be proven by means of invoking the “object browser”.
8. Press the F2 key. Select “CONTROLBUILDERLib” in the top left drop down box. Click on “CBOpenIF” in the left pane and select a method in the right pane. The “object browser” should look like figure V2.

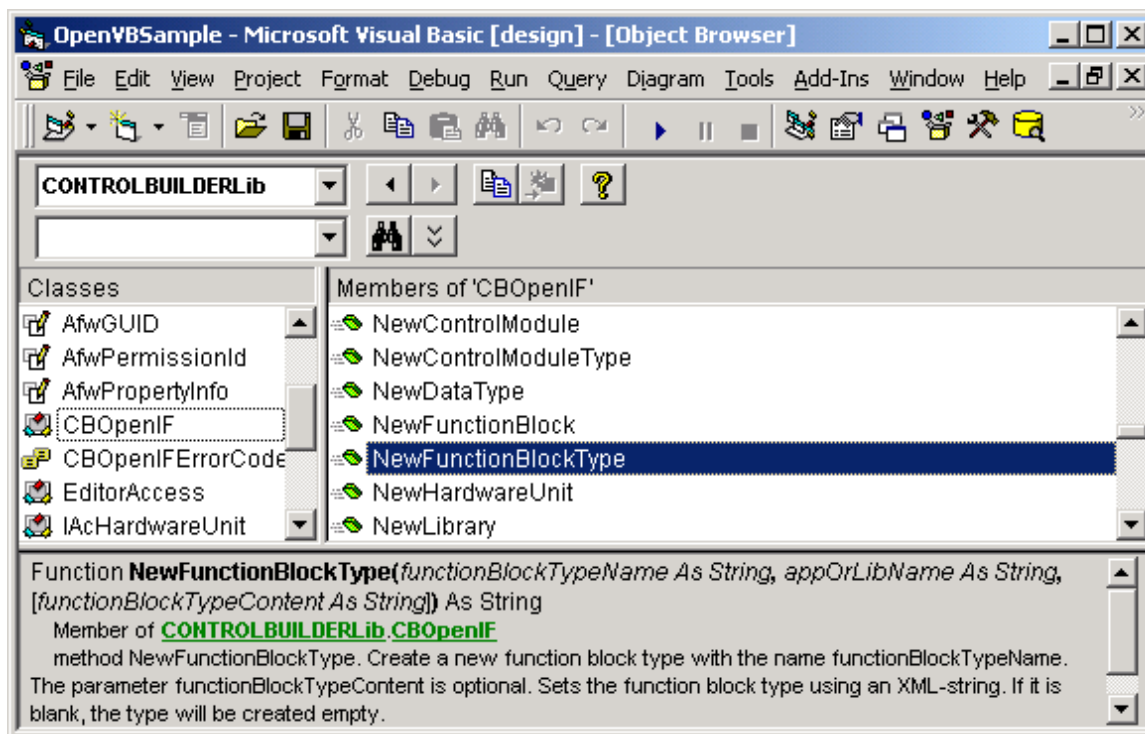


Figure V2.

9. The task is now to implement an application according to figure V3 and table T1.

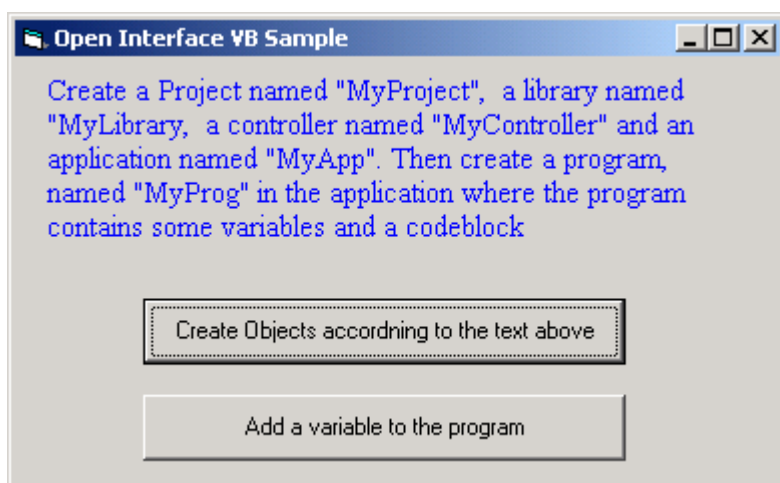


Figure V3.

Object	Name	Property	Settings
Form	frmOpenVBSample	Caption	Open Interface VB Sample
Label	lblText	Caption	Create a Project named “MyProject”, a library named “MyLibrary, a controller named “MyController” and an

			application named "MyApp". Then create a program, named "MyProg" in the application where the program contains some variables and a codeblock
CommandButton	cmdCreateObjects	Font ForeColor Caption	Times New Roman, Size = 11 Blue Create Objects according to the text above
CommandButton	cmdAddVariable	Caption	Add a variable to the program

Table T1.

10. Implement the following code in frmOpenVBSample.

```

Option Explicit
Private XMLDoc As MSXML2.DOMDocument60
Private SchemaCache As MSXML2.XMLSchemaCache60
Private cb As CONTROLBUILDERLib.CBOpenIF

Private Sub Form_Load()
    On Error GoTo someerr
    txtDataTypeName.Text = "MyDataType2"

    'Make an instance of the parser!
    Set XMLDoc = New MSXML2.DOMDocument60
    Set SchemaCache = New MSXML2.XMLSchemaCache60
    Call SchemaCache.Add("CBOpenIFSchema3_0", "CBOpenIFSchema3_0.xsd")
    Set XMLDoc.schemas = SchemaCache
    XMLDoc.async = False
    XMLDoc.validateOnParse = True 'Validate against the XML Schema
    XMLDoc.setProperty "SelectionLanguage", "XPath"
    XMLDoc.setProperty "SelectionNamespaces", "xmlns:cb=""CBOpenIFSchema3_0""

    'Make an instance of the CB Open Interface (co)class
    Set cb = New CONTROLBUILDERLib.CBOpenIF
    Exit Sub

someerr:
    MsgBox Err.Description, , "An Error I"
End Sub

Private Sub Form_Unload(Cancel As Integer)

```

```

Set cb = Nothing
Set XMLDoc = Nothing
Set SchemaCache = Nothing
End Sub

Private Sub cmdCreateObjects_Click()
    Dim XMLStr As String
    Dim Bucket As String

    On Error GoTo someerr
    Call cb.NewProject("MyProject")
    Call cb.NewLibrary("MyLibrary")
    Call cb.NewApplication("MyApp")
    Call cb.NewController("MyController", "AC 800M")

    XMLStr = "<?xml version='1.0' encoding='UTF-16' ?>" & _
        "<Program Name='MyProgram' xmlns='CBOpenIFSchema3_0'>" & _
        "<Variables>" & _
        "<Variable Name='Var1' Type='dint' InitialValue='7' />" & _
        "<Variable Name='Var2' Type='dint' />" & _
        "</Variables>" & _
        "<CodeBlocks>" & _
        "<STCodeBlock Name='CodeBlock1'>" & _
        "<ST_Code>Var2 := Var1 +1;</ST_Code>" & _
        "</STCodeBlock>" & _
        "</CodeBlocks>" & _
        "</Program>"

    Bucket = cb.NewProgram("MyProg", "MyApp", XMLStr)
Exit Sub

someerr:
    MsgBox Err.Description, , "An Error occurred"
End Sub

```


11. On the Run menu, click the Start command in order to test the VB-application. Press the button “Create Objects according to the text above “ and study the result in the Control Builder. The project explorer and Program editor should look like figure V4 and V5.

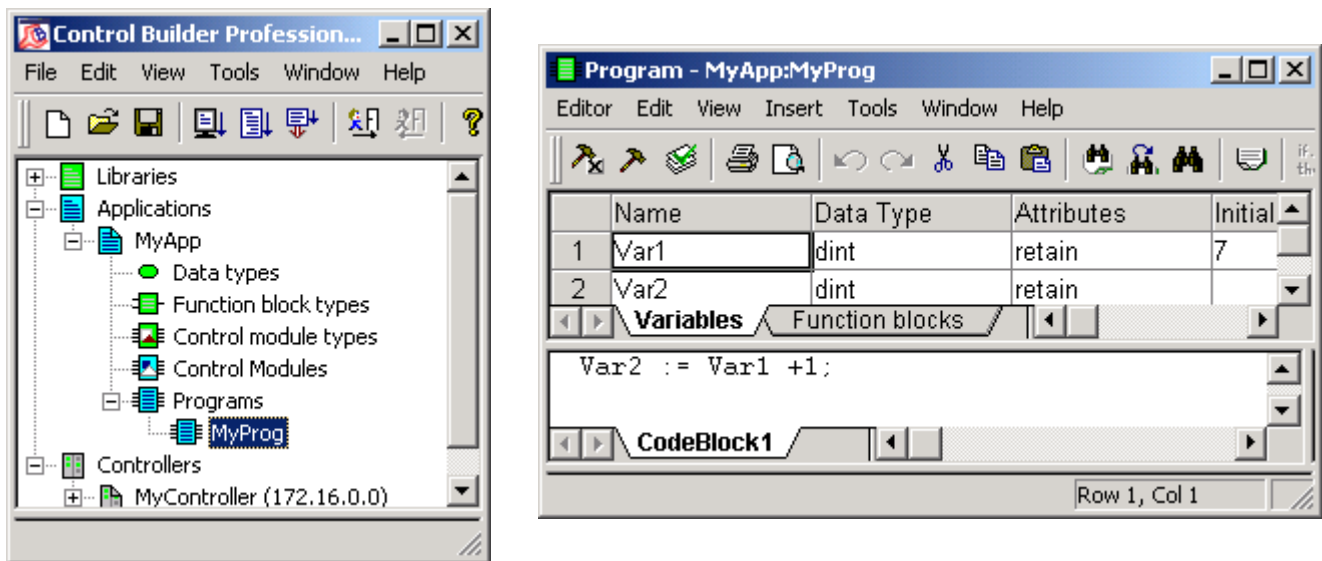


Figure V4 and figure V5.

12. Save the project and the form file in a directory, for instance in the C:\OpenVBSample directory.

We will now proceed and implement the code behind the “*Add a variable to the program*” button. See figure V3. The idea is:

- Fetch the current content of the program “MyApp.MyProg”. The Open Interface method **GetProgram** returns a string containing an XML description of the program’s content.
- Prompt the user for the name and type of the new variable i.e. the variable that will be added to the program.
- Use the XML parser MSXML in order to transform the XML-string into a tree of XMLNodes.
- Locate the `<Variables>` node in the tree of XMLNodes.
- Create a new `<Variable Name="NewName" Type="NewType" />` node.
- Insert the `<Variable>` node as a child to the `<Variables>` node.
- Transform the tree of XMLNodes to a new XML-string.
- Call the Open Interface method **SetProgram** using the new XML-string as actual parameter.
- Implement the 265 following code in frmOpenVBSample.

```
Private Sub cmdAddVariable_Click()
    Dim ProgramContentAsXML As String
    Dim Bucket As String
    Dim Ok As Boolean

    On Error GoTo someerr
```

```

`Fetch the current content of the program "MyApp.MyProg".
`The Open Interface method GetProgram returns a string containing an
`XML description of the program's content
ProgramContentAsXML = cb.GetProgram("MyApp.MyProg")

`Prompt the user for the name and type of the new variable i.e.
`the variable that will be added to the program.
Dim VariableName As String
Dim VariableType As String
VariableName = InputBox("Specify the new variable's name:", "Specify name")
VariableType = InputBox("Specify the new variable's type:", "Specify type")

`Use the XML parser MSXML in order to transform the XML-string into a tree
`of XMLNodes. First make an instance of the parser!
Ok = XMLDoc.loadXML(ProgramContentAsXML)
If Ok Then
    `A tree of XML Nodes according to the XML string "ProgramContentAsXML"
    `has been created by the parser
    `Locate the <Variables> node in the tree of XMLNodes
    Dim VariablesNode As MSXML2.IXMLDOMNode
    Set VariablesNode = XMLDoc.selectSingleNode("//cb:Variables")
    `Create a new <Variable Name="NewName" Type="NewType" /> node
    Dim NewVariableNode As MSXML2.IXMLDOMElement
    Set NewVariableNode = XMLDoc.createElement(NODE_ELEMENT, _
                                                "Variable", VariablesNode.namespaceURI)

    `Set the Attributes for the new Variable element
    Call NewVariableNode.setAttribute("Name", VariableName)
    Call NewVariableNode.setAttribute("Type", VariableType)

    `Insert the <Variable> node as a child to the <Variables> node
    Call VariablesNode.appendChild(NewVariableNode)

    `The property XMLDoc.xml is a string representation of the tree of XMLNodes
    `Call the Open Interface method SetProgram using the new XML-string as actual
    `parameter
    Bucket = cb.SetProgram("MyApp.MyProg", XMLDoc.xml)
End If
Exit Sub

someerr:
MsgBox Err.Description, , "An Error occurred"

```

End Sub

13. On the Run menu, click the Start command in order to test the VB-application. Press the button “*Add a variable to the program*” button, specify the new variable’s name and type, and study the result in the Control Builder. Bring up the Program editor and verify that the new variable has been added to the program.

5.4 A Visual C++ 6.0 application sample using the CB Open Interface

5.4.1 Some advice for Visual C++ programmers using the CB Open Interface

If you use the VC++ #import preprocessor directive the code will be easier to write and resemble the [Visual Basic code](#) presented in the last chapter.

Example of the import directive:

- #import "msxml6.dll" rename_namespace("OpenIF") using namespace OpenIF;

Use the #import directive in order to create wrapper classes.

- "Wrapper" classes have the following advantages:
 - ◆ Throw errors instead of returning HRESULT's
 - ◆ Functions can return values directly
 - ◆ Hides "AddRef" and "Release" (reference counting)
 - ◆ Allow **property** methods to be accessed like data members
 - ◆ Replaces certain types (VARIANTs and BSTRs) with classes

[Gå till första sidan](#) 

For further information see reference [8], ISBN 1-861001-20-7 Beginning ATL3 COM Programming.

5.4.2 A Visual C++ 6.0 application sample

Below I present a snippet of a C++ application. This application solves the same task as the Visual Basic application in the previous chapter.

First the h-file:

```
#import "msxml6.dll" rename _namespace("MSXML6")
using namespace MSXML6;

#import "C:\root\atlas\workplace\current\products\sl\WinNT\x86\debug\exe\Main.exe"
rename _namespace("CBProf")
using namespace CBProf;

#ifdef _DEBUG
#include "debug\msxml6.tlh"
#include "debug\Main.tlh"
#else
#include "release\msxml6.tlh"
#include "release\Main.tlh"
#endif

class COpenCPPSampleView : public CformView
{
private:
    ICBOpenIFPtr m_ICBOpenIFPtr;
    IXMLDOMDocument2Ptr m_pXMLDocument;
    IXMLDOMSchemaCollectionPtr m_pSchemaCache;
};
```

... and then a snippet from the CPP-file:

```
COpenCPPSampleView::COpenCPPSampleView()
    : CformView(COpenCPPSampleView::IDD)
{
    try
    {
        // Make an instance of the parser and the Schema Cache.!
        HRESULT hr = m_pXMLDocument.CreateInstance(__uuidof(DOMDocument60));
        HRESULT hr2 = m_pSchemaCache.CreateInstance(__uuidof(XMLSchemaCache60));
        HRESULT hr3 = m_pSchemaCache->add(_T(„CBOpenIFSchema3_0“),
            _variant_t(_T(„CBOpenIFSchema3_0.xsd“)));

        if (hr != S_OK || hr2 != S_OK || hr3 != S_OK)
        {
            AfxMessageBox(„Unable to instantiate the MSXML Parser“);
        }

        // Connect SchemaCache to the XML DOM Parser
        VARIANT varValue;
        varValue.vt = ::VT_DISPATCH;
        varValue.pdispVal = m_pSchemaCache;
        m_pXMLDocument->269utrefy_schemas(varValue);

        m_pXMLDocument->put_async(VARIANT_FALSE);
        //Validate against the XML Schema
```

```

m_pXMLDocument->validateOnParse = VARIANT_TRUE;
m_pXMLDocument->setProperty("SelectionLanguage", "Xpath");
m_pXMLDocument->setProperty("SelectionNamespaces", "xmlns:cb=\"CBOpenIFSchema3_0\"");

// Make an instance of the CB Open Interface (co)class
hr = m_ICBOpenIFPtr.CreateInstance(__uuidof(CBOpenIF));
if (hr != S_OK)
{
    AfxMessageBox("Unable to connect to CB Open Interface");
}
}
catch (_com_error ce)
{
    _bstr_t ErrMsg("Some Error Occured: ");
    ErrMsg += ce.Description();
    AfxMessageBox(ErrMsg);
}
}

//-----

COpenCPPSampleView::~COpenCPPSampleView()
{
    if (m_pXMLDocument != NULL)
        m_pXMLDocument.Release();
    if (m_pSchemaCache != NULL)
        m_pSchemaCache.Release();
    if (m_ICBOpenIFPtr != NULL)
        m_ICBOpenIFPtr.Release();
}

//-----

void COpenCPPSampleView::OnCreateObjects_Click()
{
    _bstr_t XMLStr;
    _bstr_t Bucket;

    try
    {
        m_ICBOpenIFPtr->NewProject("MyProject", "", "");
        m_ICBOpenIFPtr->NewLibrary("MyLibrary", "", "");
        m_ICBOpenIFPtr->NewApplication("MyApp", "", "");
        m_ICBOpenIFPtr->NewController("MyController", "AC 800M", "", "");

        XMLStr = "<?xml version='1.0' encoding='UTF-16' ?>";
        XMLStr += "<Program Name='MyProgram' xmlns='CBOpenIFSchema3_0'>";
        XMLStr += "<Variables>";
        XMLStr += "<Variable Name='Var1' Type='dint' InitialValue='7' />";
        XMLStr += "<Variable Name='Var2' Type='dint' />";
        XMLStr += "</Variables>";
        XMLStr += "<CodeBlocks>";
        XMLStr += "<STCodeBlock Name='CodeBlock1'>";
        XMLStr += "<ST_Code>Var2 := Var1 +1;</ST_Code>";
        XMLStr += "</STCodeBlock>";
        XMLStr += "</CodeBlocks>";
        XMLStr += "</Program>";
    }
}

```

```

    Bucket = m_ICBOpenIFPtr->NewProgram("MyProg", "MyApp", XMLStr);
}
catch (_com_error ce)
{
    _bstr_t ErrMsg("Some Error Occured: ");
    ErrMsg += ce.Description ();
    AfxMessageBox(ErrMsg);
}
}

//-----

void COpenCPPSampleView::OnAddVariable()
{
    _bstr_t ProgramContentAsXML;
    _bstr_t Bucket;

    try
    {
        // Fetch the current content of the program "MyApp.MyProg".
        // The Open Interface method GetProgram returns a string containing an
        // XML description of the program's content
        ProgramContentAsXML = m_ICBOpenIFPtr->GetProgram("MyApp.MyProg");

        // Use the XML parser MSXML in order to transform the XML-string into a tree
        // of XMLNodes.
        If (m_pXMLDocument->loadXML(ProgramContentAsXML))
        {
            // A tree of XML Nodes according to the XML string "ProgramContentAsXML"
            // has been created by the parser
            // Locate the <Variables> node in the tree of XMLNodes
            IXMLDOMNodePtr VariablesNode = m_pXMLDocument->selectSingleNode("//cb:Variables");

            // Create a new <Variable Name="NewName" Type="NewType" /> node
            short NodeType = NODE_ELEMENT;
            _variant_t nodeType(NodeType);
            IXMLDOMElementPtr NewVariableNode = m_pXMLDocument->createNode(nodeType, "Variable",
                VariablesNode->namespaceURI);

            // Set the Attributes for the new Variable element
            NewVariableNode->setAttribute("Name", "Kalle"); // Hard coded in order to make the example simple
            NewVariableNode->setAttribute("Type", "Real"); // Hard coded in order to make the example simple

            // Insert the <Variable> node as a child to the <Variables> node
            VariablesNode->appendChild(NewVariableNode);

            // The property m_pXMLDocument->xml is a string representation of the tree of XMLNodes
            // Call the Open Interface method SetProgram using the new XML-string as actual parameter
            Bucket = m_ICBOpenIFPtr->SetProgram("MyApp.MyProg", m_pXMLDocument->xml);
        }
    }
    catch (_com_error ce)
    {
        _bstr_t ErrMsg("Some Error Occured: ");
        ErrMsg += ce.Description ();
        AfxMessageBox(ErrMsg);
    }
}

```

}
}

5.5 The CB Object model

There is an OLE automation compatible object model (COM-DLL) available to use on the top of the open interface. The purpose of the object model is to make it easy for a client to use the interface. The model relieves the client from the knowledge of XML.

Note! The object model will be fully specified in another specification, see [ref \[9\]](#), and is not a part of this document. However, this subchapter is submitted so that the reader can get an idea of what the object model would look like.

The figure below shows the model for a functionblocktype.

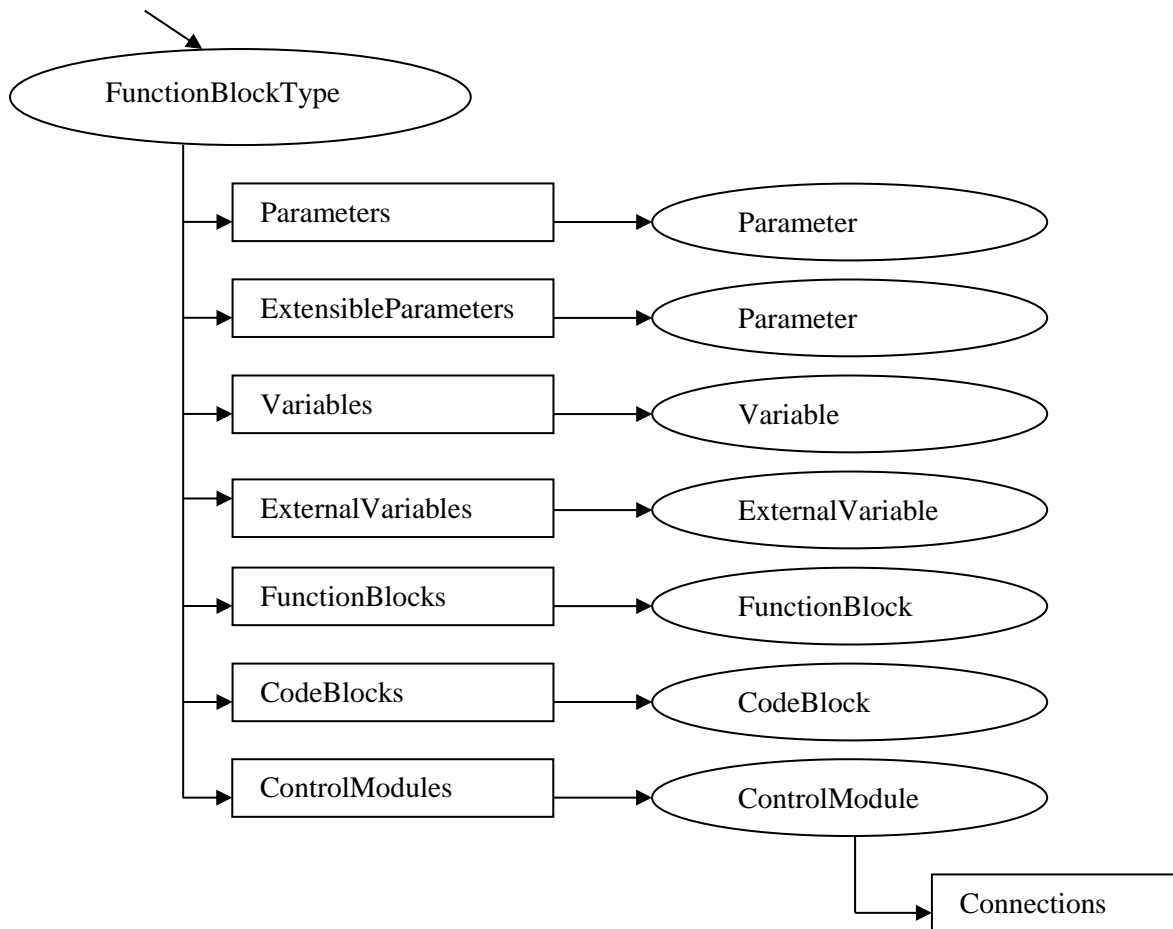


Figure 10.Example of the object model for a Function block type.

The official name of the object model is the “CB Open Interface Helper”. The “CB Open Interface Helper” is a COM-DLL providing the following services:

- The object model is able to deserialize an XML String to a hierarchy of objects, see figure 10. The objects are created in the client’s memory and have well known names such as FunctionBlockType, Variable, Parameter and so on.
- The object model is able to serialize a hierarchy of objects (representing for instance a FunctionBlockType) to an XML String.
- The “CB Open Interface” is used in conjunction with the object model. The methods of the “CB Open Interface” are used in order to get or set XML Strings from/to the Control Builder.

The following Visual Basic code shows how to use the object model.

Example 1. The example shows how to create a new FunctionBlockType using the object model and how to add new variables to an existing FunctionBlockType.

Option Explicit

```
Private cb As CONTROLBUILDERLib.CBOpenIF

Private Sub Form_Load()
    Set cb = New CONTROLBUILDERLib.CBOpenIF
    Call cb.NewProject("TestProject")
    Call cb.NewApplication("MyApp")
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Set cb = Nothing
End Sub

Private Sub cmdCreateNewFBType_Click()
    On Error GoTo errhandler
    'Create an empty Function Block Type in the client's process memory
    Dim FBType As FunctionBlockType
    Set FBType = New FunctionBlockType
    'Give it a name and set some properties!
    FBType.Name = "MyFBType"
    FBType.Description = "a description of MyFBType"
    FBType.AspectObject = True

    'Add some Variables, Parameters, FunctionBlocks, Codeblocks and so on .....
    Call FBType.Parameters.Add1("ParA", "dint")
    Call FBType.Parameters.Add2("ParB", "dint", "coldretain", cbOut, _
                                "123", "", "", "Description of ParB")
    Call FBType.Variables.Add1("Var1", "dint")
    Call FBType.Variables.Add1("Var2", "dint")
    Call FBType.FunctionBlocks.Add1("MyFunctionBlock", "RTC")

    Dim CodeStr As String
    CodeStr = "ParB:= ParA + Var1 + Var2;" & vbCrLf & _
              "Var1:= Var1+ Var2 + 1;"
    Call FBType.CodeBlocks.AddSTCodeBlock2("MyCode", CodeStr)

    'A FunctionBlockType, with content according to the code above,
    'does now exists in the client's process memory.
    'Next step is to Serialize the objects into an XMLString
    Dim XMLStr As String
    XMLStr = FBType.Serialize()

    'Finally call the "CB Open Interface" method "NewFunctionBlockType"
    'in order to create a corresponding FunctionBlockType in the
    '"ControlBuilder EXE"
```

```

    Call cb.NewFunctionBlockType("MyFBType", "MyApp", XMLStr)
Exit Sub

errhandler:
    Call MsgBox("An Error Occured " & Err.Description, , "Error Occured")
    Err.Clear
End Sub

Private Sub cmdAddVarToFBType_Click()
    On Error GoTo errhandler
    'Prompt the user for the name and type of the new variable i.e.
    'the variable that will be added to the FunctionBlockType.
    Dim VariableName As String
    Dim VariableType As String
    VariableName = InputBox("Specify the new variable's name:", "Specify name")
    VariableType = InputBox("Specify the new variable's type:", "Specify type")

    'Create an empty Function Block Type in the client's process memory
    Dim FBType As FunctionBlockType

    'Get a copy of the type, as an XMLString, from the "ControlBuilder EXE"
    'Deserialize the received XMLString to objects (i.e. Variables, Parameters,...)
    Dim ObjFactory As ObjectFactory
    Set ObjFactory = New ObjectFactory
    Set FBType = ObjFactory.DeserializeFunctionBlockType _
    (cb.GetFunctionBlockType("MyApp.MyFBType"))

    'Add a new variable
    Call FBType.Variables.Add1(VariableName, VariableType)

    'Serialize the objects to an XMLString and check in the result in
    'the "ControlBuilder EXE"
    Call cb.SetFunctionBlockType("MyApp.MyFBType", FBType.Serialize())
Exit Sub

errhandler:
    Call MsgBox("An Error Occured " & Err.Description, , "Error Occured")
    Err.Clear
End Sub

```

5.6 The Control IT Installation media

The installation program shall

- Install and register an appropriate XML parser i.e. **msxml6.dll** (installed with 3rd party installation)
- Install the XML Schema “**CBOpenIFSchema3_0.xsd**” in the **C:\Windows\System32** folder (32-bit windows installation) or in the **C:\Windows\SysWOW64** folder (64-bit windows installation)

In addition, the production installation media (Control IT installation media) shall contain a folder named “CBOpenInterface”. This folder shall contain the following items:

- Visual Basic 6.0 sample applications
- Visual C++ 6.0 sample applications
- A user manual i.e. this document
- A copy of the XML Schema “**CBOpenIFSchema3_0.xsd**”
- A HTML project named “[CBOpenIFSchema3_0.html](#)” showing the XML Schema in a user-friendly way