Application for Communication

applications & TOOLS

# Individual Visualization with OPC based on Microsoft® .NET and the Programming Language C#

SIEMENS

## Warranty, Liability and Support

We do not accept any liability for the information contained in this document.

Any claims against us - based on whatever legal reason - resulting from the use of the examples, information, programs, engineering and performance data etc., described in this document shall be excluded. Such an exclusion shall not apply in the case of mandatory liability, e.g. under the German Product Liability Act ("Produkthaftungsgesetz"), in case of intent, gross negligence, or injury of life, body or health, guarantee for the quality of a product, fraudulent concealment of a deficiency or breach of a condition which goes to the root of the contract ("wesentliche Vertragspflichten"). However, claims arising from a breach of a condition which goes to the root of the contract shall be limited to the foreseeable damage which is intrinsic to the contract, unless caused by intent or gross negligence or based on mandatory liability for injury of life, body or health. The above provisions does not imply a change in the burden of proof to your detriment.

The Application Examples are not binding and do not claim to be complete regarding the circuits shown, equipping and any eventuality. They do not represent customer-specific solutions. They are only intended to provide support for typical applications. You are responsible in ensuring that the described products are correctly used. These Application Examples do not relieve you of the responsibility in safely and professionally using, installing, operating and servicing equipment. When using these Application Examples, you recognize that Siemens cannot be made liable for any damage/claims beyond the liability clause described above. We reserve the right to make changes to these Application Examples at any time without prior notice. If there are any deviations between the recommendations provided in these Application Examples and other Siemens publications - e.g. Catalogs - then the contents of the other documents have priority.

**Copyright© 2005 Siemens A&D. It is not permissible to transfer or copy these Application Examples or excerpts of them without first having prior authorization from Siemens A&D in writing.**

For questions about this document please use the following e-mail-address:

online-support.automation@siemens.com

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

# Foreword

## Why OPC?

Process data exchange between PLCs and computers via OPC is a core element within automation technology, as OPC offers the following advantages and possibilities:

- Provision of standardized interfaces
- Openness regarding the Windows world
- Producing interoperability between heterogeneous automation systems.

## Objectives of the application

The application on hand shows a possible procedure for independently generating individual OPC clients using the programming language C# by Mircrosoft.

For this purpose, a visualization of a conveyor belt simulation is realized in C#. The programming paradigm ".NET“ by Microsoft is used here.

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

## Main contents of this application

The following main points are discussed in this application:

- Generating and using a special and reusable .NET-Control in C#
- Procedure for connecting existing software components (ActiveX-Controls) in .NET applications
- Demonstration of handling COM components, especially the OPC server, with .NET:
  - Procedure for connecting to the OPC server
  - Memory management when using COM components
- Demonstration of the most important OPC DA 2.05 methods for reading SIMATIC process variables:
  - Synchronous writing and reading
  - Asynchronous writing
  - Monitoring of process variables
- Demonstration of error handling with .NET
- Internationalizing (dynamic language selection during runtime) with .NET

## Delimitation

This application does not contain a complete description

- of the .NET framework,
- of C#,
- of the OPC specification as well as
- deeper level COM mechanisms

Therefore, basic knowledge in the area of object oriented programming, as well as in the COM field, is expected. Further knowledge in UML (Unified Modelling Language) is an advantage.

Previous knowledge in this field is expected.

## Structure of the document

The documentation of this application is divided into the following main parts.

| Part | Description |
|---|---|
| Application Description | Provides a general overview of the contents. You will learn about the components used (standard hardware and software components and the specially created software). |
| Function principles and program structures | Discusses the detailed function processes of the involved hardware and software components, the solution structures, and the concrete implementation of this application.<br>This part is necessary if you want to learn about the interaction of the solution components, for example in order to use them as the basis for own development. |
| Structure, Configuration and Operation of the Application | This part leads you step by step through the structure, important configuration steps, commissioning and operation of the application. |
| Appendix | Here you find further information, such as bibliography, glossary etc.. |

## Reference for Automation and Drives Service & Support

This entry originates from the internet application portal of the A&D Service and Support. The following link takes you directly to the download page of this document.

http://support.automation.siemens.com/WW/view/en/21043779

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

# Table of Contents

**Application Description ..... 7**

**1 Automation Task ..... 7**
1.1 Overview ..... 7
1.2 Requirements ..... 8

**2 Automation Solution ..... 9**
2.1 Overview of complete solution ..... 9
2.2 Description of the core functionality ..... 10
2.3 Required hardware and software components ..... 12
2.4 Alternative solutions ..... 14

**Function Principles and Program Structures ..... 17**

**3 General Function Mechanisms ..... 17**
3.1 Principal application model of the OPC DA application ..... 17
3.2 Differences between synchronous and asynchronous read and write jobs ..... 21
3.3 Dividing OPC items into OPC groups ..... 25
3.4 Identifying OPC items created in an OPC client ..... 26
3.5 Function mechanisms of .NET and inclusion of components of the previous programming world ..... 28

**4 Function Mechanisms of this Application ..... 33**
4.1 Logic data interface between controller and OPC client ..... 33
4.2 Structure of the OPC Client ..... 41
4.3 The .NET UserControl ..... 43
4.4 Establishing the connection to the OPC server and creating the OPC items ..... 48
4.5 Problem circles during creating of OPC clients and solutions in this application ..... 50

**5 Data flow models of this application ..... 58**
5.1 Asynchronous write procedure using .NET control ..... 58
5.2 Monitoring a box on the conveyor belt (.NET-Control) ..... 62
5.3 Synchronous read process with the ActiveX-Control ..... 65

**6 Explanations on the Simulation Program ..... 67**
6.1 Basic structures of the simulation ..... 67
6.2 Process of simulation ..... 69

**7 Code Fragments of the OPC Clients ..... 71**
7.1 Introduction and basics ..... 71
7.2 Connecting to the OPC server ..... 74
7.3 Reading and Writing OPC Items ..... 80
7.4 Release of references on the OPC server ..... 85
7.5 Internationalization ..... 87

**8 Brief Introduction into Using .NET ..... 90**

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

8.1 Including the OPC-DA-RCW interfaces ..... 90
8.2 Installing a.NET-UserControl ..... 93
8.3 Installing an ActiveX -UserControls ..... 95

**Structure, Configuration and Operation of the Application ..... 97**

**9 Installation and Commissioning ..... 97**
9.1 Installation of Hardware and Software ..... 97
9.2 Installation of the application software ..... 98

**10 Configuration Process ..... 100**
10.1 Configuring the PC station ..... 100
10.2 Configuration of the OPC server ..... 104

**11 Operating the Application ..... 111**

**Appendix and List of Further Literature ..... 116**

**12 Glossary ..... 116**

**13 Literature ..... 117**

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

# Application Description

## Content

Here an overview of the application is given. You will learn about the components used (standard hardware and software components and the specially created software).

The basic performance data show how powerful this application is.

# 1 Automation Task

## Here you will find information on …

the automation task discussed in the documentation on hand.

## Discussed topics


| Chap. | Title | Page |
|---|---|---|
| 1.1 | Overview | 7 |
| 1.2 | Requirements | 8 |

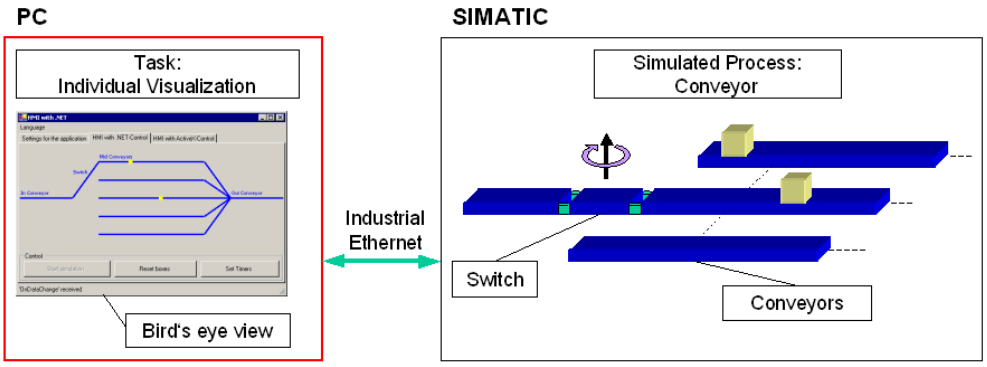

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

## 1.1 Overview

### Overview of the automation task

The following figure gives an overview of the automation task.

Figure 1-1




### Description of the Automation Task

Using an example from conveyor technology, this application shows how - using a simulated process - the movement of boxes on conveyor belts can be visualized.

## 1.2 Requirements

**Requirements for operating and monitoring modules for visualizing**

These modules must be reusable. They must also comply with the following requirements:

- Using existing .NET-Controls for controlling the simulation
- Generating an individual .NET-Control for depicting the boxes on the conveyor belts.
- Using ActiveX-Controls for depicting the conveyor speed

**Requirements to the data interface between visualizing and controlling**

Process data are exchanged via the standardized OPC-DA interface:

- Connection to the process data via Industrial Ethernet as well as the SIMATIC NET OPC Server V6.2.
- Using the OPC-DataAccess Custom interfaces V2.05
- Symbolic and absolute addressing of process data
- Synchronous reading and writing of process data
- Asynchronous writing of process data
- Implementing a structure for error handling

**Requirements to the development environment to be used**

The current Windows development environment is to be used:

- Using Microsoft Visual Studio® .NET 2003
- Using the .NET programming language Visual C#

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

# 2 Automation Solution

**Here you will find information on …**

the solution selected for the automation task.

**Discussed topics**


| Chap. | Title | Page |
|---|---|---|
| 2.1 | Overview of complete solution | 9 |
| 2.2 | Description of the core functionality | 10 |
| 2.3 | Required hardware and software components | 12 |
| 2.4 | Alternative solutions | 14 |

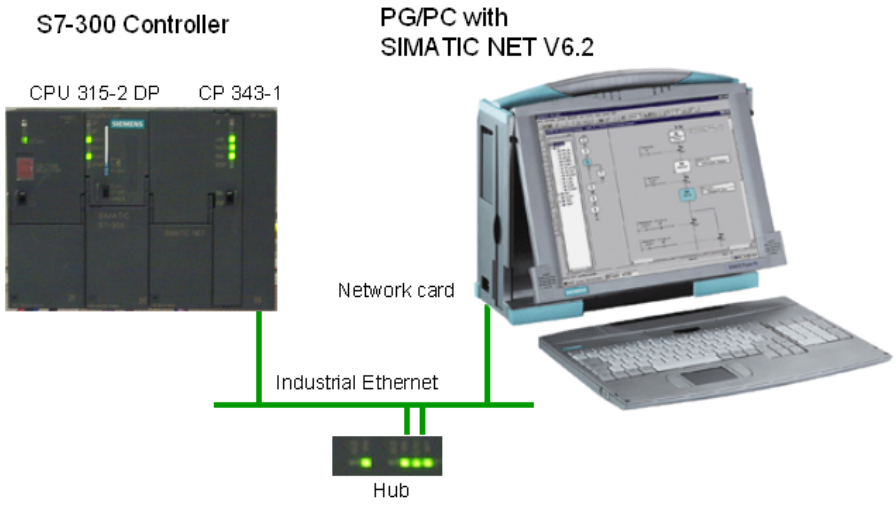

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

## 2.1 Overview of complete solution

### Display

The following figure displays the most important components of the solution:

Figure 2-1




### Controlling

On the controlling side, there is a SIMATIC S7-300 with the CP 343-1 as well as a CPU 315-2 DP.

### PG/PC

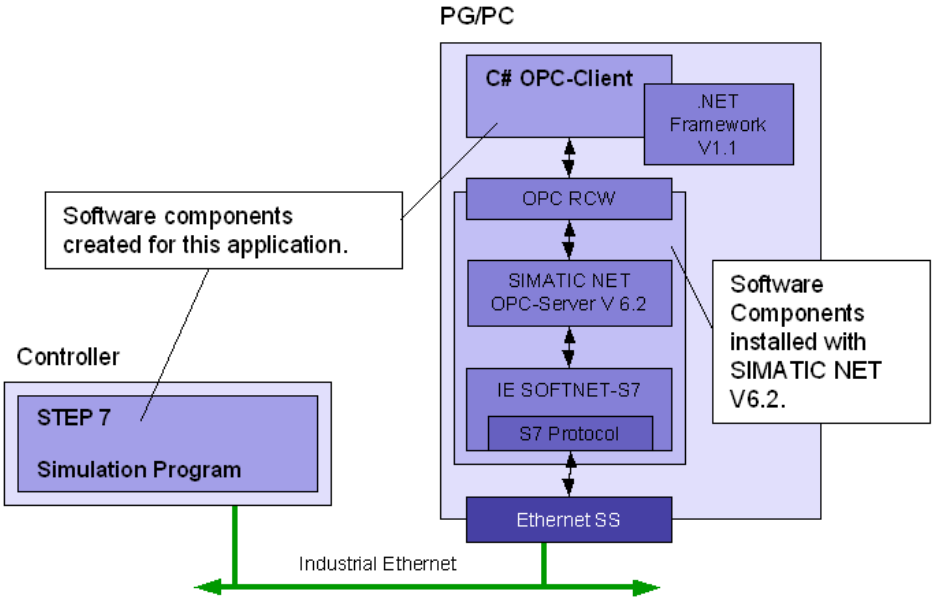
A PG/PC is connected to an S7-300 controller via hub. Thereby a standard Ethernet network card is used (already integrated in a SIMATIC Power PG).

## 2.2 Description of the core functionality

### Software components involved

The following figure shows the involved software components:

Figure 2-2




### PC/PG

A C# OPC client has been realized on the PC/PG for visualizing.

For connecting to the process , the OPC client uses the OPC Runtime Callable Wrapper (abbrev.: RCW) of the SIMATIC NET OPC-Servers V6.2. The SIMATIC NET OPC server establishes the connection to the controller via the SIMATIC NET SOFTNET-S7 connection.

### Controller

The controller provides the data to be visualized.

An S7 program for simulating the movement of boxes on a conveyor belt has been implemented for this.

### Generated software components

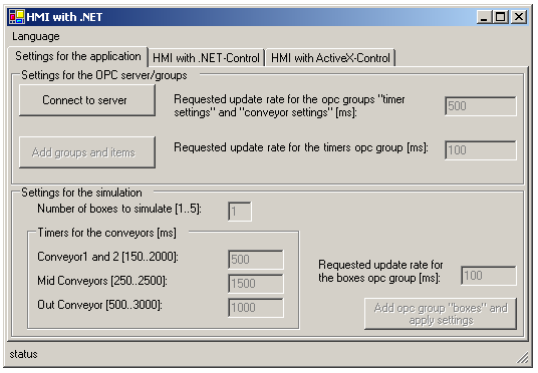
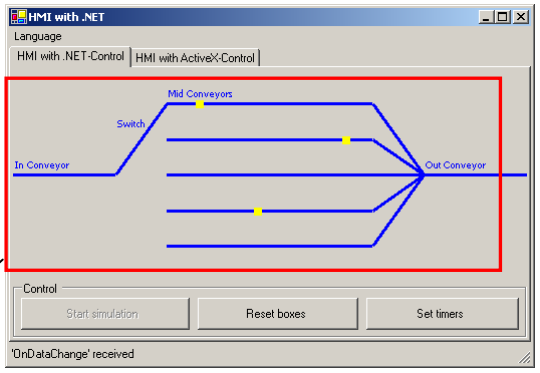
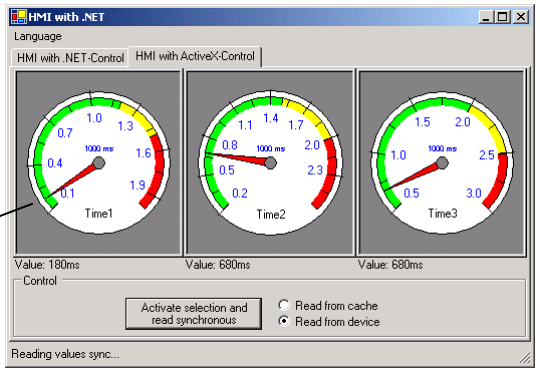
- C# OPC client
- STEP 7 simulation program

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

## General application procedure

The application consists of a visualization user-interface with three sections:

Table 2-1

| No. | Instructions | User interface |
|---|---|---|
| 1 | Settings:<br><br>Connection to the OPC server:<br>• Connecting with the OPC server<br>• Creating groups and items<br><br>Setting the process simulation:<br>• Conveyor speed<br>• Number of simulated boxes |  |
| 2 | Operator control and monitoring of variables with individually created .NET-UserControl<br><br>• Display of the position of individual boxes.<br>• Changing the switch position by means of mouse-click.<br><br>Using an individually generated Conveyor Control |  |
| 3 | Mere monitoring with ActiveX-Controls<br><br>Display of the simulated conveyor speeds.<br><br>Using ActiveX-Controls |  |

**Note** The actions are performed via the operator elements of the user interface.

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

## 2.3 Required hardware and software components

### Hardware components for the PC

Table 2-2

| Component | No. | MLFB / Order number | Note |
|---|---|---|---|
| PS307 5A | 1 | 6ES7307-1EA00-0AA0 | |
| CPU 315-2DP | 1 | 6ES7315-2AG10-0AB0 | |
| CP 343-1 | 1 | 6GK7343-1EX20-0XE0 | |
| Standard hub | 1 | Depending on product | |

### Hardware components for the PC

Table 2-3

| Component | No. | MLFB / Order number | Note |
|---|---|---|---|
| Power PG | 1 | 6ES7751-0EA31-0LB3 | |
| NDIS-capable network card | 1 | Depending on product | Integrated in Power PG |

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

### Standard software components

Table 2-4

| Component | No. | MLFB / Order number | Note |
|---|---|---|---|
| STEP 7 V5.3 SP1 (or higher) | 1 | 6ES7810-4CC07-0YA5 | Integrated in Power PG |
| SCL V5.3 | 1 | 6ES7811-1CC05-0YA5 | Integrated in Power PG<br>Required for adaptations |
| SIMATIC NET IE SOFTNET-S7 | 1 | 6GK1704-1CW62-3AA0 | Contains:<br>• SIMATIC NET Software V6.2 SP1<br>• OPC-RCW<br>• OPC-Server |
| Visual Studio .NET 2003 with .NET Framework 1.1 or higher | 1 | Depending on product. To be ordered via administrator or at http://www.microsoft.com/ | Required for adaptations |
| .NET Framework 1.1 | 1 | Download free of charge at http://www.microsoft.com/ | Required for commissioning |

### Example files and projects

The following list contains all files and projects used in this example.

Table 2-5

| Component | Note |
|---|---|
| 21043779_C#_OPC_Client_STEP7_v10.zip | This file contains the archived STEP 7 simulation program |
| 21043779_C#_OPC_Client_SETUP_v10.zip | Setup for installation of the user interface (without .NET-Framework) |
| 21043779_C#_OPC_Client_CODE_v10.zip | Source code files of the user interface |
| 21043779_C#_OPC_Client_DOKU_v10_d/e.pdf | This document |

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

## 2.4 Alternative solutions

Here you are given information on existing alternatives for solving the automation problem and the characteristics of the alternatives.

**Using operator control and monitoring systems (HMI systems)**

Basically, standardized HMI systems such as WinCC or ProTool can be accessed.

This has the following advantages and disadvantages:

Table 2-6

| Advantages | Disadvantages |
|---|---|
| • The connection to the process variables can be configured<br>• Simple integration of existing components<br>• Most functionalities are “on board“ (e.g. message systems) | • Partly, extensive and expensive software packages are required<br>• Only components supporting the HMI system can be used (.NET components for examples cannot be easily integrated into WinCC V6)<br>• Integration into the MS-Office world not possible without difficulties<br>• Connecting own databases not possible without difficulties |

**Custom interface vs. automation interface**

Due to the data structures defined in the OPC specification, the OPC interface has been implemented as custom interface.

In order to also be able to use OPC with programming languages not enabling Custom interface connection (VBA, VB V6.0 or older), an automation interface was implemented which depicts the more complex structures as simpler structures.

Due to the additional management required (→ additional wrapper, see Glossary) a disadvantage regarding performance must be taken into account.

**Note** For high-performance requirements in particular, work should exclusively be performed with the Custom interface, which excludes using VB 6.0 or VBA.

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

### Decision criteria for using different programming languages

The following table shows you the most important decision criteria which are decisive for selecting the used programming language.

Table 2-7

| Criterion | Description |
|---|---|
| Custom interface | Is it possible to use the custom interface of the OPC DA interface ? |
| Automation interface | Is it possible to use the automation interface of the OPC DA interface ? |
| Simple implementation | How much is the language suitable for relatively simply implementing code ? |
| Error handling | How well is the language suitable for realizing professional error handling ? |
| Performance | How well is the language suitable for developing performant OPC applications ? |

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

### Comparing different programming languages

The following table compares the different programming languages using the above described criteria.

Table 2-8

| Criterion | VBA (MSOffice) | VB V6.0 | VC++ | .NET-languages (managed) |
|---|---|---|---|---|
| Custom interface | - | - | ✓ | ✓ (with RCW) |
| Automation interface | ✓ | ✓ | ✓ (*) | ✓ (*) (with RCW) |
| Simple implementation | ++ | ++ | - (**) | + (**) |
| Error handling | - | - | + (**) | ++ (**) |
| Performance | - | - | ++ (**) | + (**) |

(*) should not be used

(**) using the custom interface

### OPC under .NET and C++

Table 2-9

| Criterion | Description |
|---|---|
| Implementation | Implementation of OPC clients in .NET hardly differs from the implementation in C++, as the COM mechanisms must still be followed.<br>However, the implementation of a user interface with .NET is clearly faster than comparable implementation with C++. |
| Error handling | The most striking difference between .NET and C++ is probably the error handling.<br>Instead of polling “HRESULT“ as usual, i.e. COM return values (→ glossary), an exception handling (→ Glossary) is realized. This has the following advantages:<br>• No error will be overlooked<br>• Reading code more simple<br>• Uniform error handling within the OPC client<br>• Possible introduction of error hierarchies by means of implementing some Exception classes |
| Code performance | Current measurements yielded that the performance differences in OPC matters are hardly worth mentioning.<br>The behavior of the user interface is a different matter - here .NET is rather comparable with VB V6.0.<br>Generally, however, the bottleneck will not be the update rate of the user interface, but the connection to the OPC server or the connection between OPC server and controller.<br>Therefore, for OPC applications, .NET can be considered equal to VC++ V6.0. |

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

### Prospects

Microsoft is currently enforcing performance improvements of .NET applications. Performance boosts in this field are to be expected over the following months.

In order to reduce the concrete implementation work during using OPC with .NET, the OPC foundation (→ http://www.opcfoundation.org/) is currently working on simplifying the interfaces. Hereby, for example, new RCWs are implemented which make better use of the .NET possibilities.

It can be assumed, that in the future the OPC interfaces for .NET will be as easy to handle as in VB V6.0.

**Note** The RCW currently provided by the OPC foundation has not yet been approved by the Siemens AG.

# Function Principles and Program Structures

**Content**

Discusses the detailed function processes of the involved hardware and software components, the solution structures, and where sensible the concrete implementation of this application.

Read this part if you want to know how the individual solution components interact.

# 3 General Function Mechanisms

**Here you will find information on …**

the function mechanisms applying regarding OPC and which require particular consideration with .NET.

**Discussed topics**

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc


| Chap. | Title | Page |
|---|---|---|
| 3.1 | Principal application model of the OPC DA application | 17 |
| 3.2 | Differences between synchronous and asynchronous read and write jobs | 21 |
| 3.3 | Dividing OPC items into OPC groups | 25 |
| 3.4 | Identifying OPC items created in an OPC client | 26 |
| 3.5 | Function mechanisms of .NET and inclusion of components of the previous programming world | 28 |

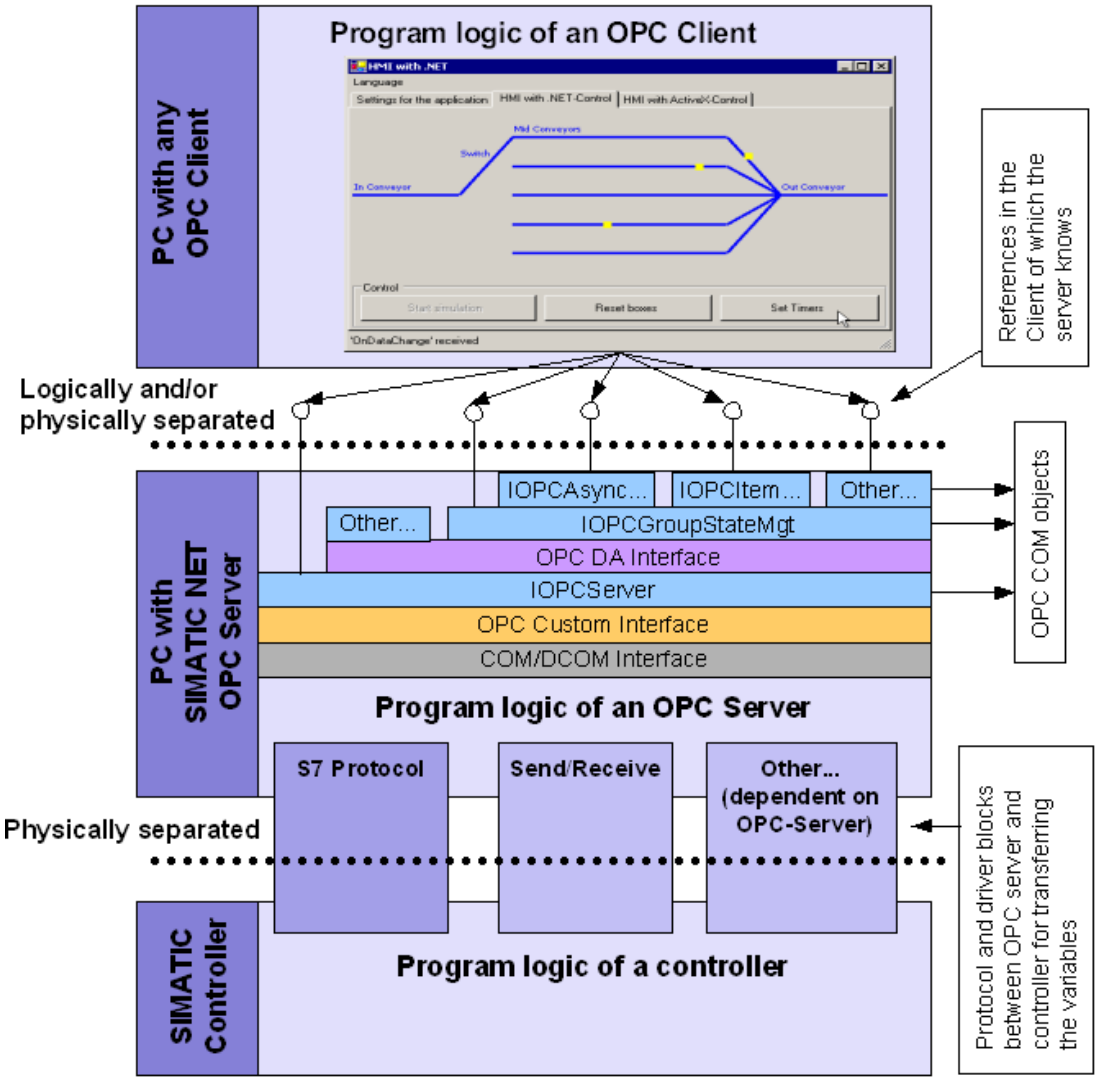

## 3.1 Principal application model of the OPC DA application

The application model of the OPC DA interface is briefly discussed now. A detailed description of the object model is available in the OPC DA specification \1\.

First you see a graphical overview of the application model, then the individual sections of the overview are discussed in detail.

## Overview

Figure 3-1




## OPC client

By generating references in the client to the respective interfaces, the OPC server is informed of client access to a certain variable of the configured controller being desired.

Further interfaces available via the “IOPCGroupStateMgt“ Group(s) interface enable read and write access to these variables. The following access methods are possible:

- Synchronous reading / writing
- Asynchronous reading / writing
- Monitoring variables (reporting value changes)

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

**Note**

Single threaded OPC clients are blocked at synchronous read / write jobs.
For asynchronous read / write jobs, the OPC client remains operable ("responsive").

Further information see chapter 3.2.

### OPC server

The OPC server is the central communication unit between an OPC client and the respective controller.

Via COM/DCOM mechanisms, it provides interfaces standardized towards the OPC client which allow each COM capable application to access variables of any controller.

In this application the DataAccess interfaces for generating the logic connections between OPC items and process variables are decisive.

The OPC server is connected to the respective controller via the implementation of the communication protocols.

### Connection between OPC server and controller

The values of the variables are read out via the connections configured in the OPC server using the respective protocol blocks. These connections are configured using the NetPro tool (see chapt. 10.2).

Hereby it must be noted that the protocol blocks are not a standardized interface, but proprietary protocols implemented for the respectively underlying controller.

### Reading variables from the controller

Reading and updating the variables can be performed in two ways:

Table 3-1

| Service | Description |
|---|---|
| Variable service | For variable services, one or several process variables are specified by means of absolute or symbolic addressing. Thereby, the variables to be monitored can be polled cyclically from the OPC server (→Polling, see Glossary). |
| Block service | If the variables to be monitored are transferred to the OPC server program controlled by means of larger data blocks, then this is referred to as block service. Thereby, not the process variables themselves are addressed, but rather the data areas such as a data block. A cyclic "monitoring" of the receive buffer by the OPC server only makes sense here for receive items (e.g. "receive"). |

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

**Notes on the services**

- In this application, only variable services are used.
- The absolute addressing only differs from the symbolic addressing by the ITEM_ID.
- The variable and block services are on the OPC client side only differentiated by its ITEM_ID.
- Here, “block services“ only refers to the services (“SEND/RECEIVE“, “BSEND/BRECEIVE“) provided by the SIMATIC NET OPC Server V6.2.

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

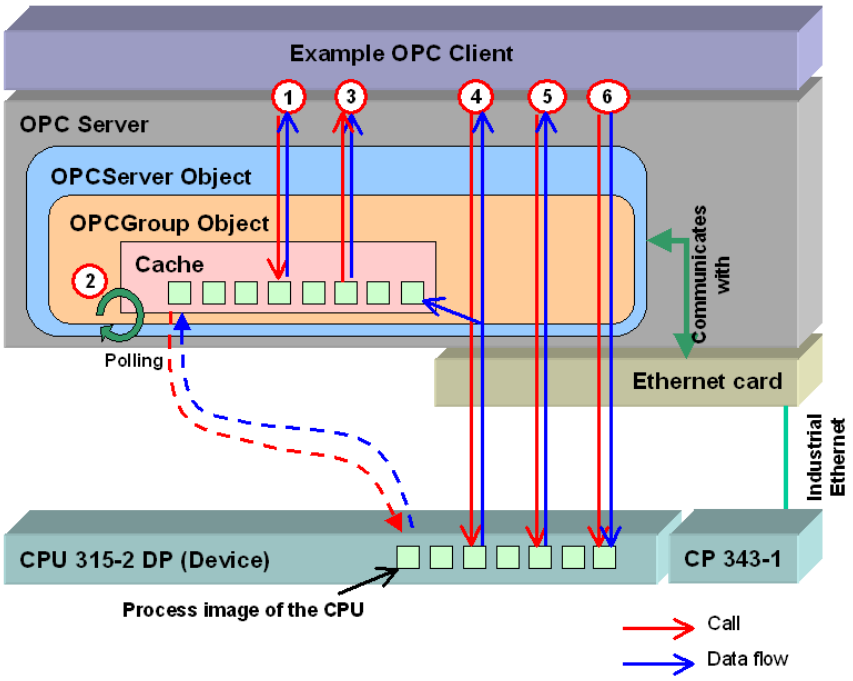
## 3.2 Differences between synchronous and asynchronous read and write jobs

The differences between synchronous and asynchronous reading and writing of OPC items is discussed below. Hereby, differences between accesses to the “DEVICE“ and the “CACHE“ are explained in detail.

### Function principle of synchronous / asynchronous read and write jobs

The following image gives you an overview of the function principles of the respective calls:

Figure 3-2




### Explanations on the figure

- The figure shows the call direction (read arrows) and the resulting data flow (blue arrows).
- Data is exchanged between the OPC server and the controller using an Industrial Ethernet card and a CP343-1.
- Data exchange between OPC client and OPC server occurs as inter-process communication (here: COM).
- "CACHE" refers to a temporary buffer which the OPC server generates for a particular group. It contains a local image of the process variables defined for this group (which in return are managed by the group as OPC items).
- "DEVICE" refers to the process variables on the controller.

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

### Explanation of the processes

The following table explains the respective calls and data flows. Thereby, the following consequences are pointed out:

Table 3-2

| No. | Instructions | Note |
|---|---|---|
| 1 | Synchronous read job on the CACHE | The value of the OPC item is read with the value currently located in the CACHE. |
| 2 | Updating the CACHE | The OPC server updates the entire CACHE with the values from the DEVICE after the time period defined with “requestedUpdateRate“.<br>**Note:**<br>The requestedUpdateRate need not correspond to the actual update rate. Please note the update rate returned by the OPC server (“revisedUpdateRate“)! |
| 3 | OnDataChange-Event | For OPC items parameterized as "active", the OPC server has registered a data change within its CACHE. This is reported to the OPC client.<br>**Note:**<br>This mechanism is ideal for monitoring the process variables. The update rate corresponds to the actual "updateRate". |
| 4 | Asynchronous read job to the DEVICE | The OPC client starts an asynchronous read job via the OPC group.<br>The OPC server reads the requested process variables from the process image of the S7-CPU and delivers to the OPC client via the OnReadComplete event. It then writes the read values to its CACHE<br>**Note:**<br>The OPC client can be further operated (is "responsive") at asynchronous calls while the respective job is processed. |
| 5 | Synchronous read job to the DEVICE | The value of the OPC item is read with the value currently located in the process image of the S7-CPU.<br>**Note:**<br>This access may take several seconds depending on physics, data traffic and data volume.<br>During this time, the thread of the OPC client, which starts the synchronous read call, cannot perform any other tasks! Single threaded OPC clients are hence blocked during this time and cannot execute further operating and monitoring tasks. |

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

| No. | Instructions | Note |
|---|---|---|
| 6 | Synchronous or asynchronous write job | Write jobs are only executed to the DEVICE, irrespective of whether they are synchronous or asynchronous.<br>**Note:**<br>This access may take several seconds depending on physics and data volume.<br>If a synchronous write job has been cancelled, the same behavior as described in point 5 takes place; for an asynchronous call the OPC client behaves as described in point 4. |

### Access to CACHE and DEVICE

The following table summarizes the operations possible to "DEVICE" and "CACHE"

Table 3-3

| Operation | DEVICE | CACHE | Note |
|---|---|---|---|
| Synchronous reading | ✓ | ✓ | Reading from the CACHE requires the group and the respective item to be active. |
| Asynchronous reading | ✓ | - | The OPV server reads the variable from the DEVICE. This also updates its CACHE. |
| Synchronous writing | ✓ | - | Write jobs are always written to the DEVICE. |
| Asynchronous writing | ✓ | - | Write jobs are always written to the DEVICE. |
| Monitoring (event controlled) | - | ✓ | Monitoring of variables is only possible using the CACHE. |

**Note**

Apart from read and write jobs, further methods are also available (such as IOPCAsyncIO2::Refresh) which, however, were not used in this application.

For further information see OPC Specification \1\.

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

### Comparison of application cases

The following table contains a comparisons of different application cases and shows the recommended procedure:

Table 3-4

| Operation | Cyclic/ Acyclic | Data quantity | DataChange-mechanism | Sync | Async |
|---|---|---|---|---|---|
| Read | Cyclic monitoring | Large | + | * | * |
| | | Small | + | * | * |
| | Acyclic | Large | - | - | + |
| | | Small | - | + | - |
| Write | Acyclic | Large | - | - | + |
| | | Small | - | + | - |

(*) The OnDataChange mechanism is executed asynchronous to the CACHE

**Note** Large data volumes in the range of several kilo-bytes were to be read or written with the block services, e.g. “BSEND/BRCV“. Hereby the SIMATIC works as active communication partner.

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

## 3.3 Dividing OPC items into OPC groups

Here you learn about the criteria to be taken into consideration when dividing the OPC items to OPC groups.

### Motivation of dividing OPC items

As explained in chapter 3.2, certain ways of behavior result from certain call types.

Now it makes sense combining process variables or OPC items with a similar operating or monitoring behavior into groups.

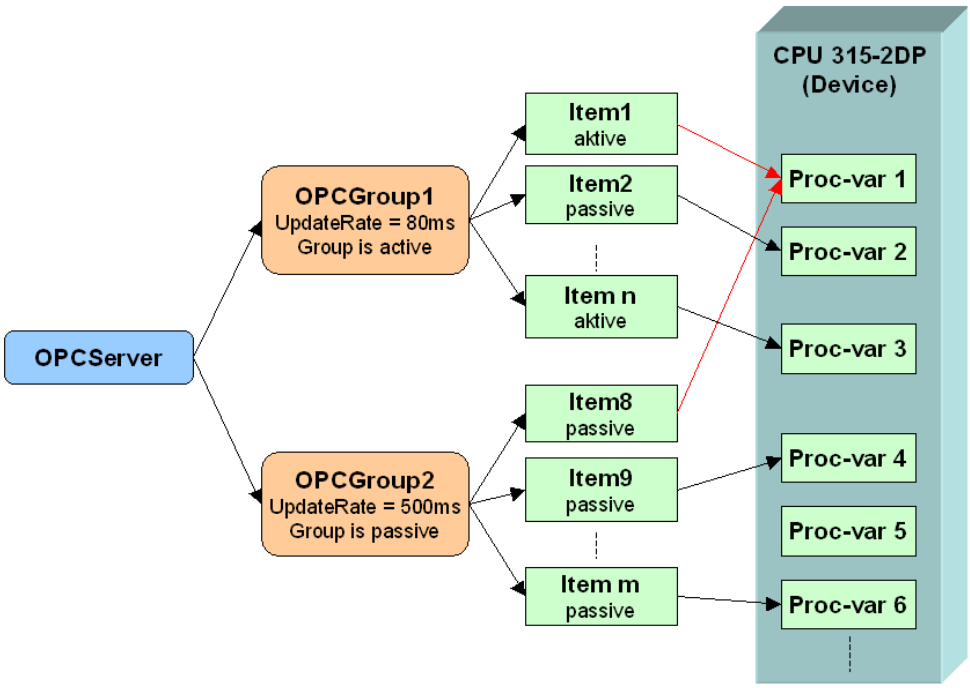
In some cases, however, it is desirable that a process variable is to be monitored at two different locations of the OPC client (such as for example in two different pages or dialogs). Another example for this is a process variable being monitored at one location, however only being sporadically (i.e. acyclically) read at an other location.

In these cases it makes sense creating the same process variable in several groups as OPC item.

### Dividing process variables to OPC items and OPC groups

The following figure illustrates this option with two different groups. Hereby the process variable 1 is contained in two OPC items which are stored in two groups. This enables generating different ways of behavior for a process variable in an OPC client.

Figure 3-3




Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

## 3.4 Identifying OPC items created in an OPC client

In order for the process variables mapped to OPC items being read or written by an OPC client, a definite assignment of OPC items to an OPC client must exist.

The concept of identifiers or handles is used here.

### Handle types

Two handle types are distinguished. This differentiation ensures that the OPC client as well as the OPC server can identify the respective OPC items.

Table 3-5

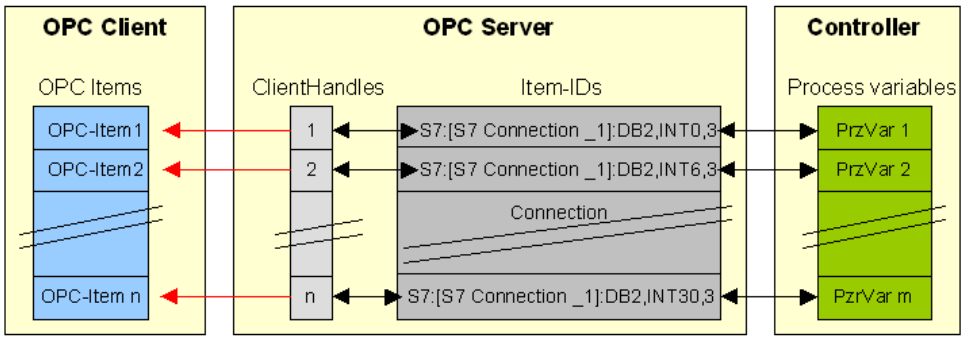
| Type | Description |
|---|---|
| Clienthandles | This handle servers for identification of an OPC item within the OPC client |
| Serverhandles | This handle servers for identification of an OPC item within the OPC server |

### Serverhandles

If the call direction is from OPC client to OPC server, the OPC client must transfer the respective server handles to the OPC server.

Example: write into the OPC items

Figure 3-4



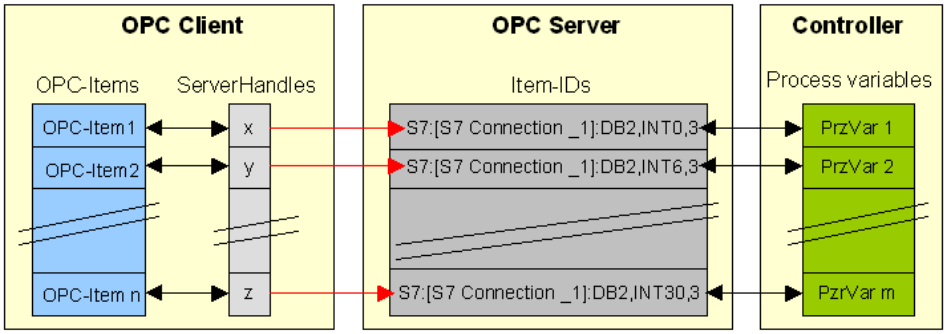

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

### Clienthandles

If the call direction is from OPC server to OPC client, the OPC client will receive the respective client handles from the OPC server.

Example: The OPC items in the CACHE have changed → the OnDataChange event is triggered

Figure 3-5




### Effects on the Client program

This structure implies that an OPC client must manage the server and/or client handles depending on the application case.

This way, the client handles may, for example, serve as an index of the respective OPC item in an OPC item array.

The server handles should either be encapsulated in a separate server handle array or in an own OPC item management class

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

## 3.5 Function mechanisms of .NET and inclusion of components of the previous programming world

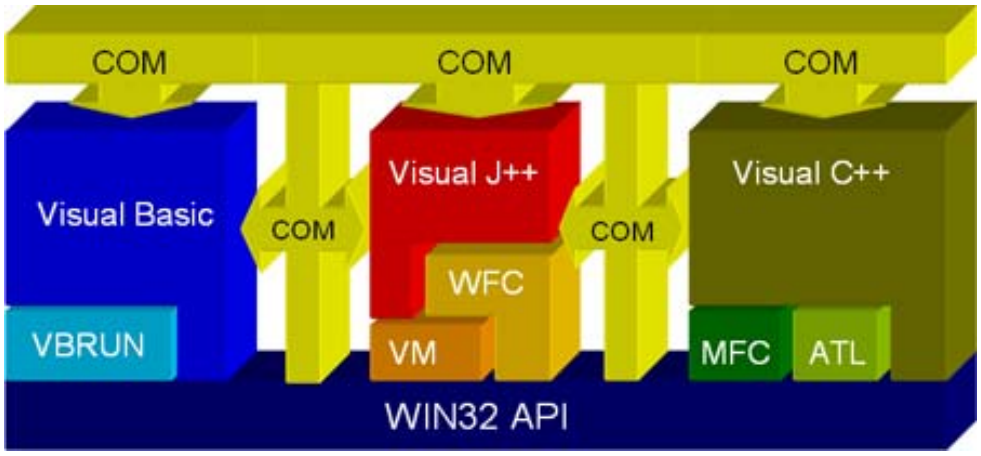
The previous as well as the latest Windows programming world will be discussed first now. Then the differences to be noted are explained.

It is only a brief introduction helpful for understanding the "how to program with .NET" as well as "how to combine the new and the old world?". Further information is available in the secondary literature \2\.

### 3.5.1 Programming model of the old world

**Overview**

Figure 3-6




**API**

The previous programming model of the Windows world builds on an **A**pplication **P**rogrammers **I**nterface (API). The API provides a lot of C-functionalities for accessing the operating system resources and functionalities.

**Programming languages and libraries**

Using various programming languages with partially extensive libraries (**VBRun**time, **M**icrosoft **F**oundation **C**lasses, **A**ctive **T**emplate **L**ibrary) it was possible to implement Windows programs. Hereby the libraries respectively encapsulated parts of the API functions.

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

### Interacting of the languages with COM

In order for the developed program components to be able to work with each other, **C**omponent **O**bject **M**odel (COM) by Microsoft was introduced. Using standardized interface definitions enabled establishing a "connecting glue" between the individual components. The concept of **D**istributed **C**omponent **O**bject **M**odel (DCOM) was developed in order to expand the cross-language and cross-component interaction to a cross-computer interaction.

**Note**

The OPC server works as COM component.

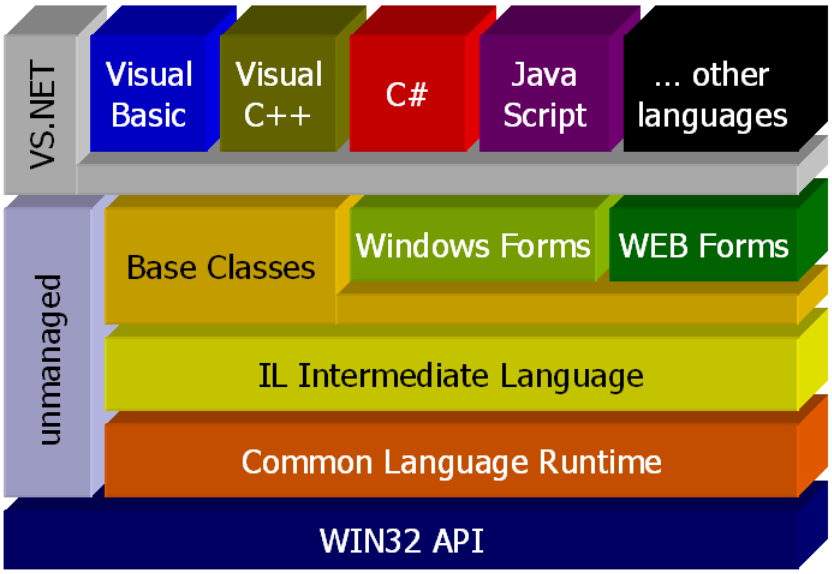
→ OPC clients access the OPC server via COM mechanisms.

## 3.5.2 Programming model of the new world

### Overview

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

Figure 3-7




### API

The basis of the model is Win32 API like in the previous Microsoft programming world.

### CLR

Based on API follows the **C**ommon **L**anguage **R**untime (CLR). It constitutes a runtime environment comparable with a Java Virtual Machine model. It compiles **J**ust **I**n **T**ime (JIT) a type of byte code in X86 code (depending on the respective processor architecture); Microsoft speaks of IL code (**I**ntermediate **L**anguage).

### Intermediate Language, Garbage Collector and unmanaged Code

The advantage of the IL code is the fact that it is independent of the platform. It is also possible to execute this code on Linux (Unix) systems, if a CLR has been implemented there (i.e. http://www.mono-project.com/).

The CLR does not only perform the JIT compiling of the IL code, but also the entire memory management. Similar to Java, a **G**arbage **C**ollector (GC) was developed, who releases un-referenced memory areas within the CLR after an unspecified time. Code subjected to the access of the Garbage Collector is also referred to as “managed Code“.

In order for it to be possible in certain cases to still control the release of memory areas manually, the developed code can be declared as "unmanaged". This enables protecting certain code areas or entire programs explicitly from the access of the Garbage Collector.

### Base Classes

Regarding the libraries of partly varying extend of the old world, Microsoft has now implemented a uniform base class library which can be accessed independently from the .NET programming language.

### Integrator Visual Studio .NET

Visual Studio .NET (VS .NET) takes on the task of combining the different .NET languages with each other.

This enables realizing .NET components in different languages. Interaction between the .NET components is possible with Visual Studio .NET without greater efforts.

Mainly the easy to handle integration options within the .NET components enable a faster development.

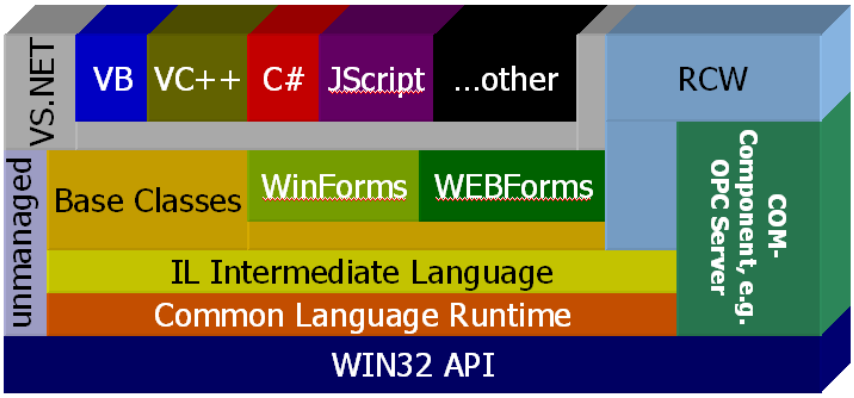
Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

### 3.5.3 Integration of COM components in .NET applications

**Overview**

Using a COM component in a .NET application/component:

Figure 3-8




**Runtime Callable Wrapper (RCW) and the relationship with the COM component**

Installing a COM component generated for the previous Windows world within a .Net application, requires a "wrapper", a type of a shell which encapsulates all interface definitions for the .NET application.

This is necessary as the previous interface definitions located in so-called IDL files (**I**nterface **D**efinition **L**anguage, see Glossary) are not supported by .NET. This wrapper is also referred to as **R**untime **C**allable **W**rapper (RCW).

For COM components offering the automation interface, Visual Studio can generate these wrappers automatically.

For COM component providing the custom interface, such wrappers must be generated manually.

**Note** The differences between automation and custom interface are not discussed here in greater detail. This requires secondary literature \2\.

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

### Memory management when using COM components

Memory management for .NET applications is performed by the Garbage Collector, however, COM components require an explicit memory management. Therefore, the following points must be considered for data exchange between both components:

Table 3-6

| Direction | Description |
|---|---|
| .NET zu COM | In .NET, all variables are objects. Due to the fact that generally speaking, COM servers do not synchronize with COM clients, transfer values should be protected from the access of the Garbage Collector ("pinning" of objects see chapter 7.3). |
| COM zu .NET | COM components deliver return values in form of COM pointers. Due to the fact that in the managed code of .NET clients such pointers do not exist, the return values must first be saved into .NET objects. This is performed via the .NET data type "IntPtr" and methods / objects of the "System.Marshal" classes. |

**Note** Chapter **7** contains code fragments on this topic.

### RCW and OPC

Due to the OPC specification, OPC components offer a Custom interface, i.e. an RCW must be generated manually for performant applications.

An RCW is delivered for the SIMATIC NET OPC server for the DataAccess interface V2.05, which must only be integrated into the respective Visual Studio project. (Chapter 8.1)

If the automation interface of OPC is used with .NET, then the RCW is automatically generated by VS .NET as soon as a reference to the OPC automation interface is included into the .NET project. The resulting double encapsulation of the OPC custom interface however, may cause a reduction in performance.

**Note** This application deals with the usage of the custom interface, as a performant connection is the focus of this application.

**Note** The reverse application case – using a .NET component as COM component – is possible using a **C**om **C**allable **W**rapper (CCW). This however, is not further discussed within the framework of this document, as it is not necessary for comprehension of this application.

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

# 4 Function Mechanisms of this Application

**This chapter explains ...**

how the classes in the OPC client interact with each other and which tasks the most important components fulfill.

The implementation of the simulation on the controller side is not discussed here in greater detail, as this is not the core focus of this application.

**Discussed topics**


| Chap. | Title | Page |
|---|---|---|
| 4.1 | Logic data interface between controller and OPC client | 33 |
| 4.2 | Structure of the OPC Client | 41 |
| 4.3 | The .NET UserControl | 43 |
| 4.4 | Establishing the connection to the OPC server and creating the OPC items | 46 |
| 4.5 | Problem circles during creating of OPC clients and solutions in this application | 50 |


Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

**Structure of the chapter**

First the logic data connection between controller and OPC client is illustrated. Hereby, the corresponding data structures in controller and OPC client are listed.

Subsequently, the relationships within the OPC client are illustrated in a class diagram, and the tasks and function principle of the OPCClienForm class and the ConveyorGraphControl component are explained.

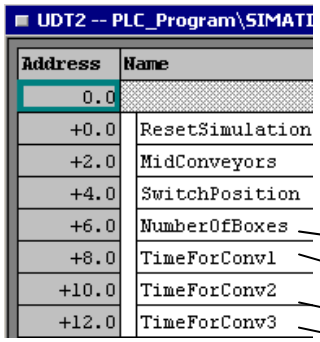
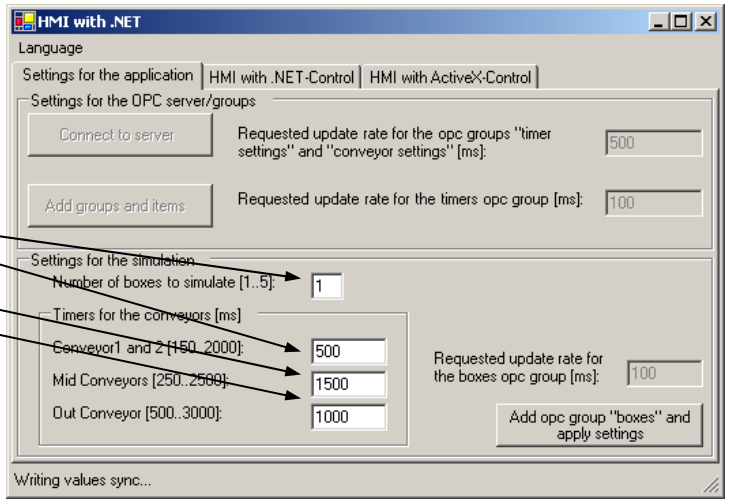
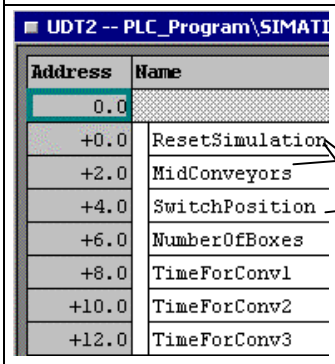
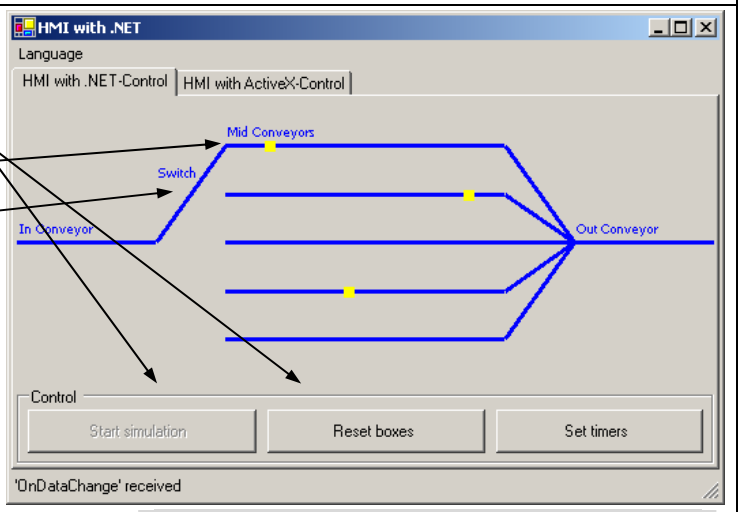
## 4.1 Logic data interface between controller and OPC client

This chapter first deals with which process variables are connected with which visualization elements. Subsequently, the relationship between the simulated box movements in controller and its display in the OPC client are explained.

### 4.1.1 Connecting the process variables with the visualization elements

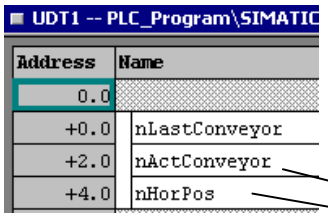
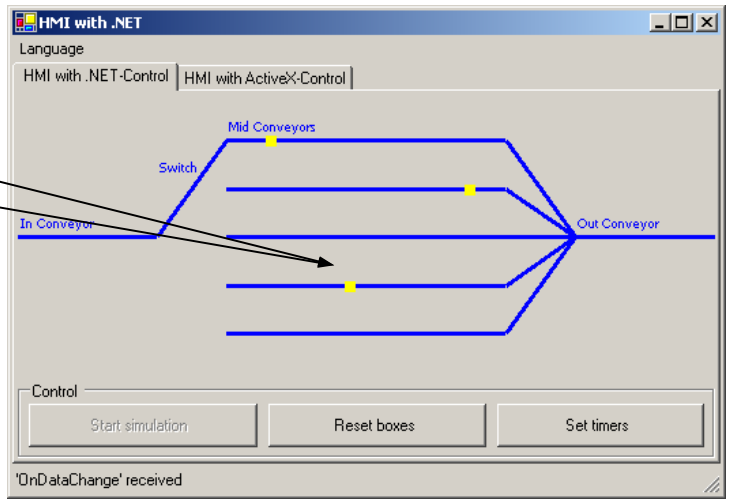
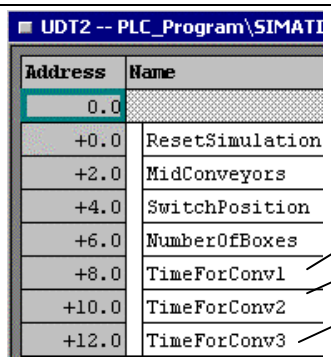
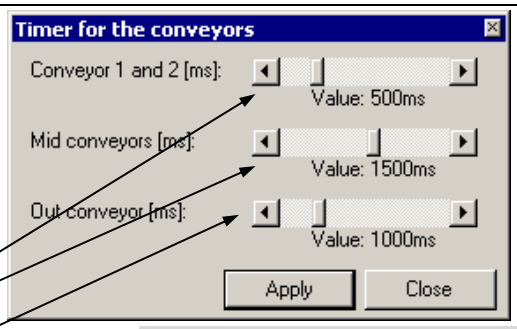
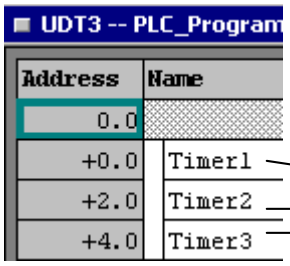
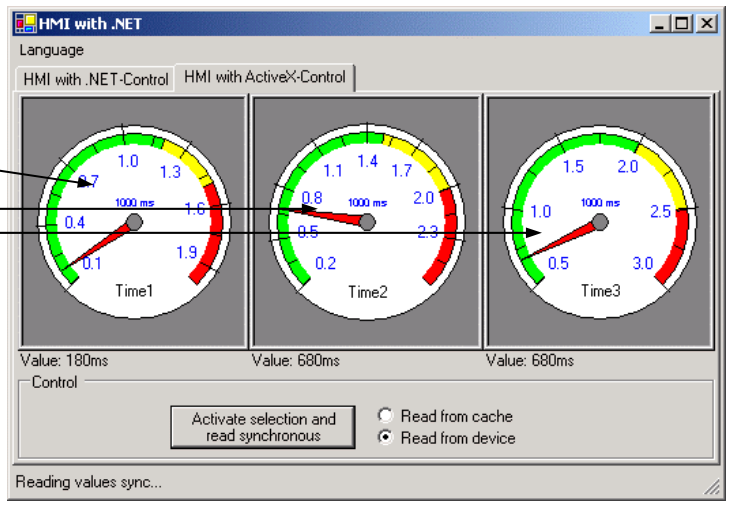
The following table shows the allocations:

Table 4-1

| | Process variables | Visualization element |
|---|---|---|
| Simulation settings |  **Note:** TimeForConv1..3 correspond to a measure for conveyor speed. |  |
| |  |  **Note** The parameter “MidConveyors“ (number of mid conveyors) is defined at the time of design. |

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

| | Process variables | Visualization element |
|---|---|---|
| Box | <br>Each box is based on the above structure<br><br>**Note:**<br>`Set Timers` enables you to set the conveyor sped at any time. |  |
| Conveyor speec |  | <br>**Note** There is a read and a write access. |
| |  | <br>**Note** There is only a read access |

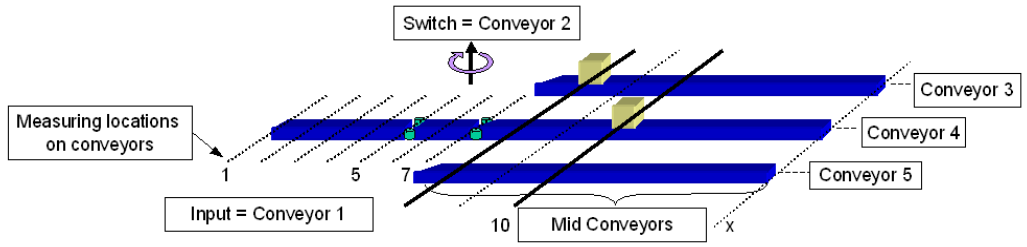
### 4.1.2 Data structure of boxes and conveyor belt in the controller

This chapter explains the correlation between the box data structure in the controller and the visualization.

**Plant overview (section)**

The following situation is assumed:

Figure 4-1




**Description of the plant overview**

Each conveyor belt has a certain number of measuring location assigned to it. The following situation results for the given section:

- The in conveyor has the measuring locations 1 to 5
- The switch has the measuring locations 5 to 6
- The mid conveyors have the measuring locations 7 to x
- Box 1 is on the uppermost conveyor belt (conveyor 3)
- Box 2 is on the mid conveyor belt (conveyor 4)

**Data structure of the boxes**

The data structure of a box is defined by an UDT in the controller. The simulation contains only an array of box structures.

In the above situation there are two boxes, which produces the following array:

Table 4-2

| Box | Data structure | Current value |
|---|---|---|
| 1 | Conveyor on which the box was | Conveyor 2 |
| | Conveyor on which the box is | Conveyor 3 |
| | Current horizontal position at which the box is located - this corresponds to the measuring location which last recorded the position | 8 |
| 2 | Conveyor on which the box was | Conveyor 2 |
| | Conveyor on which the box is | Conveyor 4 |
| | Current horizontal position at which the box is located - this corresponds to the measuring location which last recorded the position | 10 |

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

**Data structure of the conveyor**

The conveyor does not have an own data structure, as it is default - apart from the number of mid conveyors.

However, each conveyor section has a fixed number of "measuring locations". These measuring locations report the position of the boxes (see beginning of this chapter).

**Process of the simulation in the controller**

Timer are running in the controller in order to simulated the conveyor speeds.

Furthermore, a simulation block contains the array of the boxes to be simulated.

If a timer has terminated, the simulation block with the respective parameters is called. The block detects the position (measuring location) of the respective box on the conveyor and increments it. Should there be a conveyor transition, it will be processed (more information in chapter 6).

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

### 4.1.3 Data structure of the boxes in the OPC client

**Difference between data structure on the controller**

Due to the fact that the requirements for the data structure of a box within the visualization user interface differ from that of the controller, a partial amount of the data structure of the controller is used.

A box object for example must contain the information about its physical screen position, color, dimensions etc. It is not enough to only encapsulate the number of the conveyor and the horizontal position.

For that reason the box data from the controller must be written into the respective box object using an auxiliary function.

**Steps for transfer of box data**

The following steps reflect the rough procedure (chapter **5** informs you of the occurring data flow):

1. Fill control data into the respective box object.
2. Draw the box object on the respective conveyor.

The OPC client is hence only responsible for transfer of box data from the controller to the box objects of the conveyor. The boxes are drawn by the generated .NET-Control “ConveyorGraphControl“.

**Note** Data flow for changes of box position is depicted in chapter 5.2.

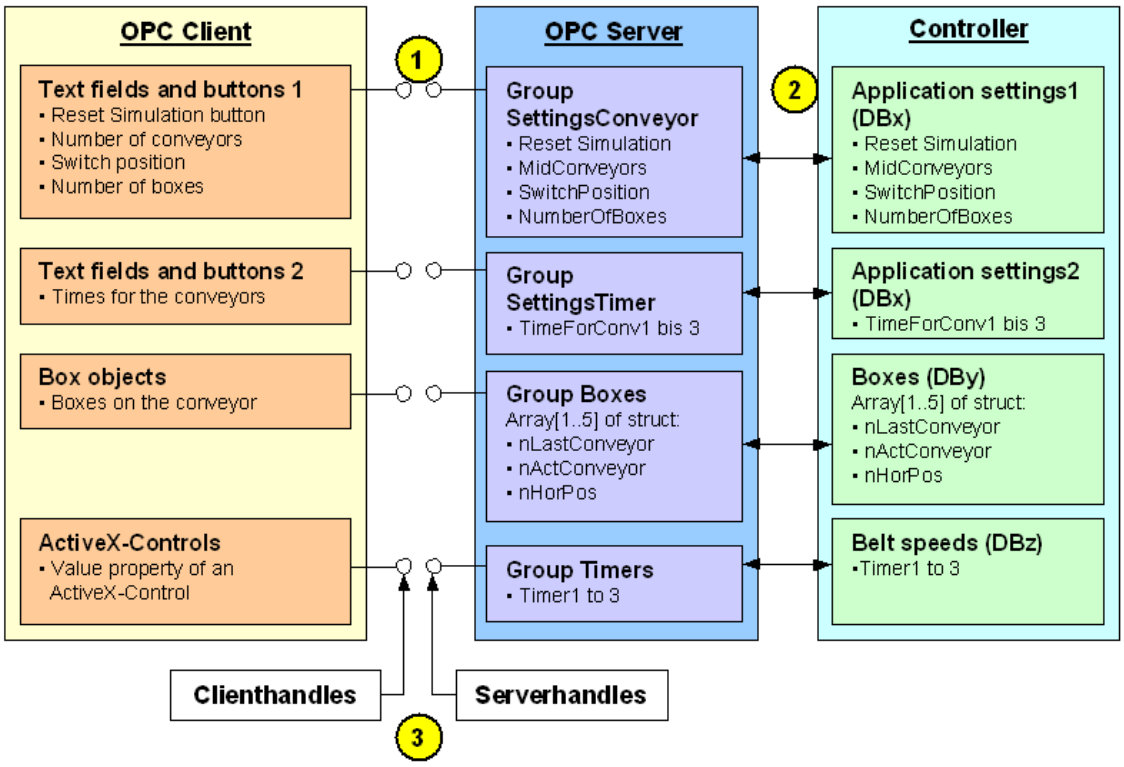
### 4.1.4 Selected depiction of the process variables to be visualized

As already explained in chapter 3.1, the process variables are depicted on OPC items, which in return are allocated to OPC groups.

**Overview**

The following overview shows the overall correlation. The three numbered sections are discussed below.

Figure 4-2




Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

### 1 – Assigning the process variables to OPC groups

The process variables to be visualized are here divided into 4 sections (see chapter4.1.1). A structuring criteria is the update rate, the property, whether process variables are to be monitored and whether reading synchronously and/or asynchronously. Furthermore, it is distinguished in how the OPCItems relate to the user interface. This results in creating 4 OPC groups.

Table 4-3

| Group | Characteristics of variables | Updata rate | To be monitored | OPC access method | Used in the user interface |
|---|---|---|---|---|---|
| SettingsConveyor | Variables serving for conveyor setting | Low | No | • Sync<br>• Async (switch) | Switch position only* |
| Settings Timer | Variables serving for setting conveyor speed | Low | No | Sync | SpeedsDialog |
| Boxes | Variables representing the boxes | High | Yes | OnDataChange | ConveyorGraph-Control |
| Timers | Variables depicting the conveyor speed | Average | No | Sync | ActiveX-Controls |

*) The remaining OPC items serve only for controlling/setting the simulation on the S7-CPU

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

### 2 – Assigning the process variables to OPC items

The respective process variables are now assigned to each group.

Table 4-4

| Group | Process variable |
|---|---|
| SettingsConveyor | • Reset simulation: Resets the application<br>• Mid Conveyors: number of mid conveyor belts<br>• SwitchPosition: switch position<br>• NumberOfBoxes: number of simulated boxes |
| Settings Timer | Continuously settable conveyor speeds |
| Boxes | Array[1..5] of boxes:<br>The boxes to be monitored/simulated |
| Timers | Timer1 to Timer3:<br>Timers displayed on the ActiveX-Controls |

### 3 – Addressing the process variables within SIMATIC

The assignment of OPC items to process variables occurs using so-called ItemIDs.

ItemIDs must be defined as data type “string“ and must follow a given syntax. This syntax depends on used OPC server and service (variables / block service).

The following applies for the SIMATIC NET OPC server V6.2 as well as the service used here (variable service):

<Protocol>:[<Connection name>],<Object>,<Type>,<Startindex>,<Number>

### Example for an OPC item

A variable is accessible via the S7 connection “S7 connection_1“ in DB2, DBW0. Then the ItemID is:

“S7:[S7 connection],DB2,INT,0,1“

**Note** Further information available in chapter 7.2 or OPC Specification / OPC Manual \1\.

### Managing the OPC Objects

This application saves all relevant data of the created OPC items in one or several arrays.

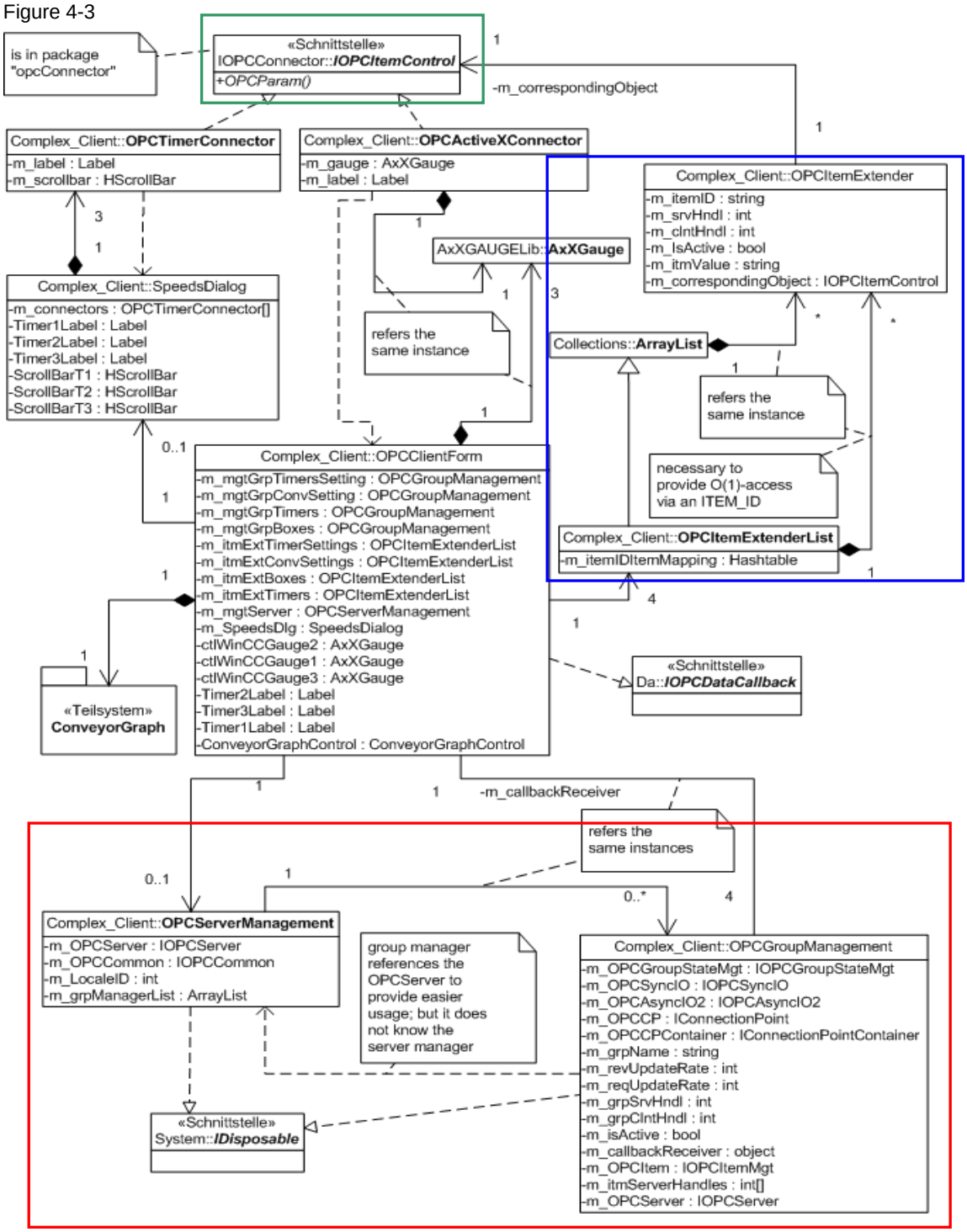
- Here the Client handles serve as index of an OPC item in the array
- The desired Server handles of OPCItems are returned from the corresponding array by specifying the respective ITEM_ID.

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

## 4.2 Structure of the OPC Client

### Class diagram of the OPC Client

Figure 4-3




Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

**Note**

Only the most important relationships and attributes are depicted.

Relationships with standard .NET controls or structures also necessary for this application are not depicted. This is referred to the commented source code.

### Explanations to the frame colors

Table 4-5

| Section | Explanation |
|---|---|
| Red | The classes framed in red encapsulate OPC specific functionalities, which can be simple adapted and reused.<br>These are mainly functionalities which in this application were necessary regarding the interfaces IOPCServer, IOPCGroupStateMgt and IOPCItemStateMgt, i.e. especially the data exchange via OPC-DA 2.05 (see chapter 4.5.1). |
| Blue | The classes framed in blue serve as usable logic interface for managing OPC items as simply as possible. This particularly simplifies reading and writing of OPC items to and from user interface elements. Core class is the “OPCItemExtenderList“ (see chapter 4.5.2). |
| Green | The area framed in green serves as interface for all user interface elements reading or writing data from OPC items using OPCItemExtender objects. |

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

### OPCClientForm – Tasks

This class represents the main dialog of the application. It has the following tasks:

- Receiving all OPC server events
- Setup and holding of the connection to the OPC server
- Contains all controls; amongst other things the conveyor control “ConveyorGraphControl“ (see chapter 4.3) and the ActiveX controls “xXGaugeLib.AxXGauge“ encapsulated by Visual Studio .NET
- Referencing all OPCGroupManagement objects and the OPCServerManagement objects (see chapter 4.5.1)
- Contains the lists which contain all of the required inforamtion On OPC items (see OPCItemExtender or OPCItemExtenderList in chapter 4.5.2 and 4.5.3)
- Control of the entire application
  - Forwarding control data to subordinate controls and vice versa
  - Error handling

The principle steps for OPC connection are given in chapter 4.4.

## 4.3 The .NET UserControl

### General

The .NET-Control “Conveyor“ (ConveyorGraph) is integrated into the main dialog.

The "conveyor" control works as independent component and can therefore be used in different .NET applications. It has an "internal" logic.

### Tasks

The control fulfills the following tasks:

- Managing the box objects
- Display of boxes on the conveyor belts
- Realizing a particular conveyor in form of an operable settable switch
- Providing public properties for manipulating the appearance of the conveyor within the .NET Designer
- Encapsulates the complete display and management functionality of the boxes and conveyors
- Providing a public box class (necessity, see chapter 4.1)
- Implementing a resizing functionality

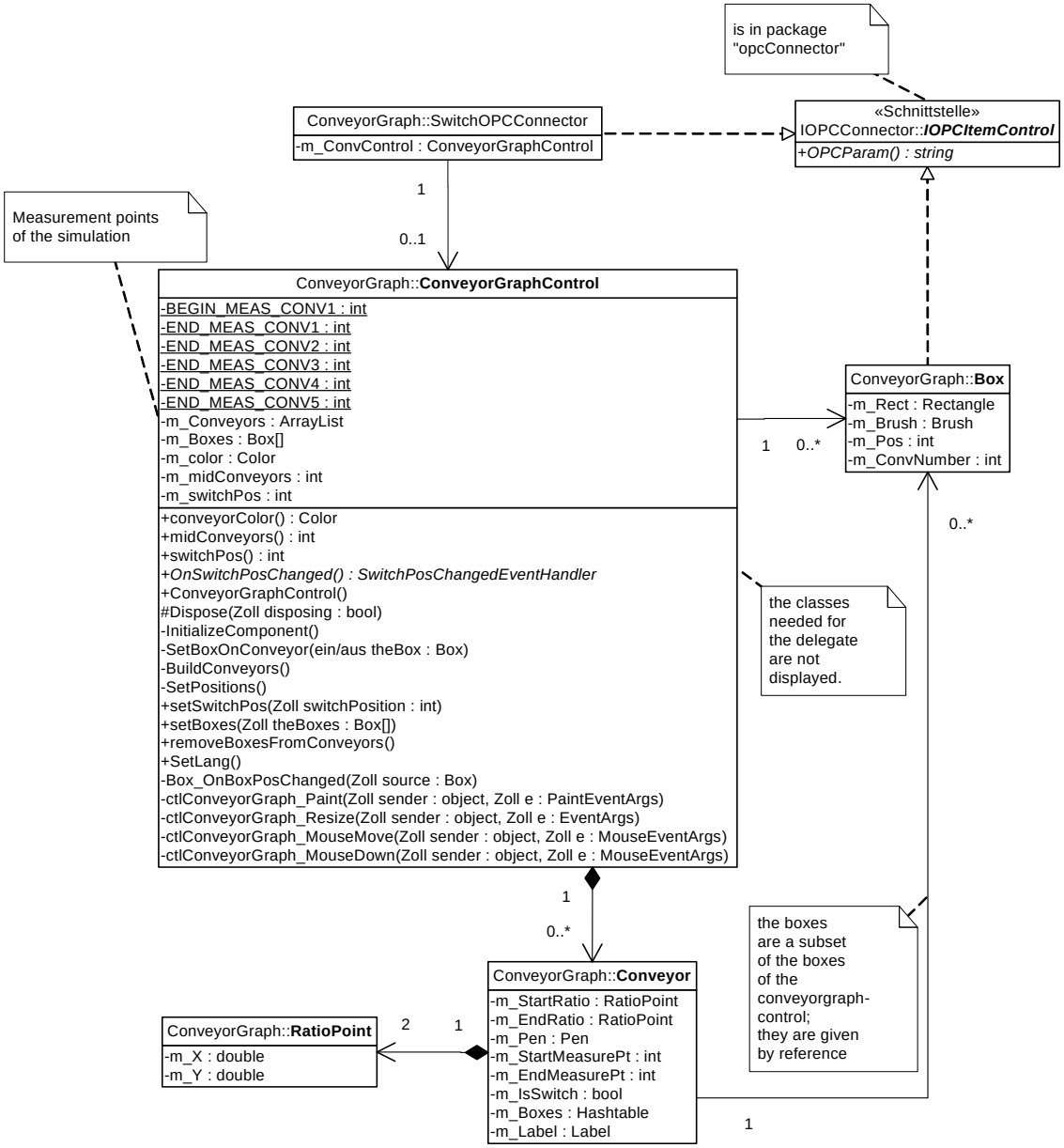
Outwardly the control only shows the public methods of the encapsulating class "ConveyorGraphControl" as well as the box class.

The following class diagram shows the internal structure of the control.

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

## Class diagram

Figure 4-4

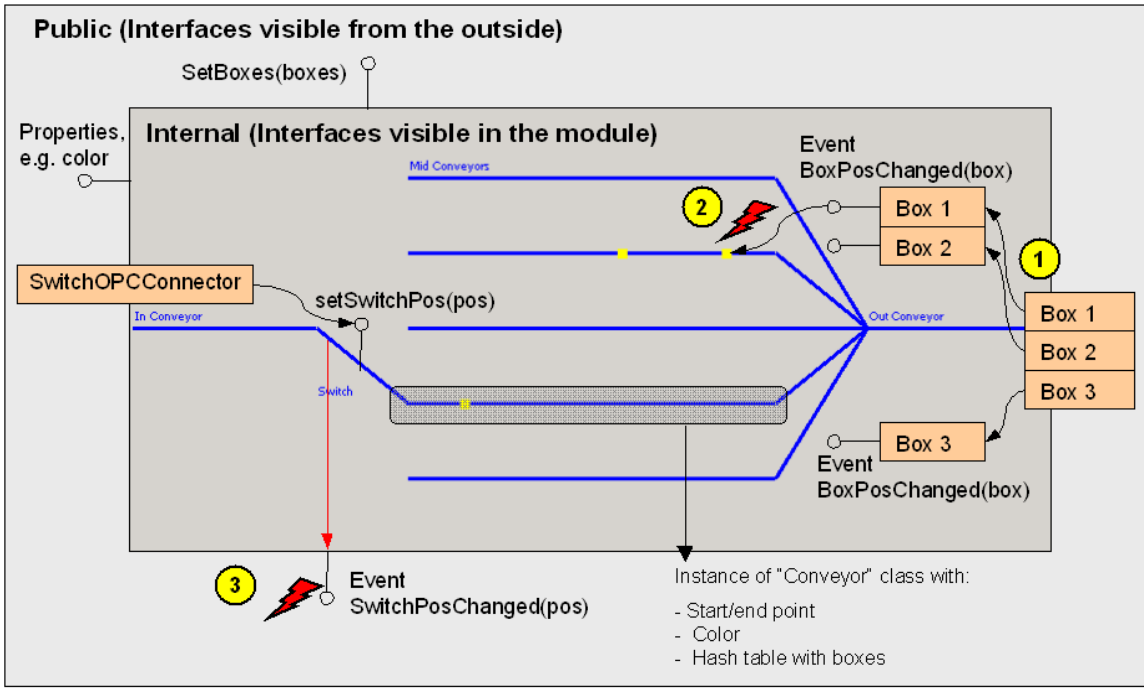



Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

## Overview of the function principle

The following figure shows the general function principle:

Figure 4-5




## Explanation of the function principle

Table 4-6

| No. | Explanation |
|---|---|
| 1 | The control consists of a collection with objects of the "Conveyor" type. Each "Conveyor" in return has a start and end point, a color (here: blue) and a number of measuring locations.<br>Furthermore, each "Conveyor" manages a hash table with boxes to be displayed. These boxes are copied by reference. They correspond to the boxes in the box array. |
| 2 | If the position of a box changes, then this is reported to the corresponding box via "OnDataChange". The box then notifies (BoxPosChanged) the ConveyGraphControl. Then this box is redrawn on the respective conveyor.^<br>**Note:**<br>For the OnDataChange event to be able to trigger the "BoxPosChanged" event, the "Box" class implements the "IOPCItemControl" interface. For further information see source code or chapter 4.5.3. |

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

| No. | Explanation |
|---|---|
| 3 | The “SwitchPosChanged“ event, which is triggered by mouse-click, informs the father class or the father container (here: OPCClientForm) that the switch position must be changed.<br>After the date has been written into the OPC item, the switch position is also changed in ConveyorGraphcontrol via the “SwitchOPCConnector“ class (data flows are described in chapter 5).<br>**Note:**<br>The switch position is set via an asynchronous write command. For changing the switch position with the OnWriteComplete event a “SwitchOPCConnector“ has been realized which implements the “IOPCItemControl“ interface. For further information see source code or chapter 4.5.3. |

**Note**

- The vertical position of the boxes is determined using the switch position, which was active at a certain box at position 7 (end position of the switch). The box must memorize the vertical position as the "switch" conveyor may have various positions.
- The positions of the box are identical with those stored in the controller.

For further information, please refer to the commented source code.

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

### The interfaces of the .NET controls

The following properties, methods and events are provided by the .NET control:

### "ConveyorGraphControl" class

Table 4-7

| Contents | Note |
|---|---|
| ConveyorColor | Color of the conveyors |
| MidConveyors | Number of conveyors in the mid section |
| SwitchPos | Position of the switch |

Table 4-8

| Method | Parameter | Note |
|---|---|---|
| Constructor | - | |
| SetBoxes | Box[] boxes | Transfers a box array to the Control |
| RemoveAllBoxes | - | Removes all boxes on the conveyor belts |
| SetLanguage | - | Sets the language of the Control to the language setting of the application |

Table 4-9

| Event | Parameter | Note |
|---|---|---|
| OnSwitchPos-Changed | • Object sender<br>• Int switchPos | The parameter "switchPos" refers to the position to which the switch is to be set. |

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

### "Box" class

Table 4-10

| Method | Parameter | Note |
|---|---|---|
| Constructor | Brush theColor | Color of the box |

### "SwitchOPCConnector" class

Table 4-11

| Method | Parameter | Note |
|---|---|---|
| Constructor | ConveyorGraphControl theConvControl | Required in order to call the internal setSwitchPos(pos) method. |

## 4.4 Establishing the connection to the OPC server and creating the OPC items

### Introduction

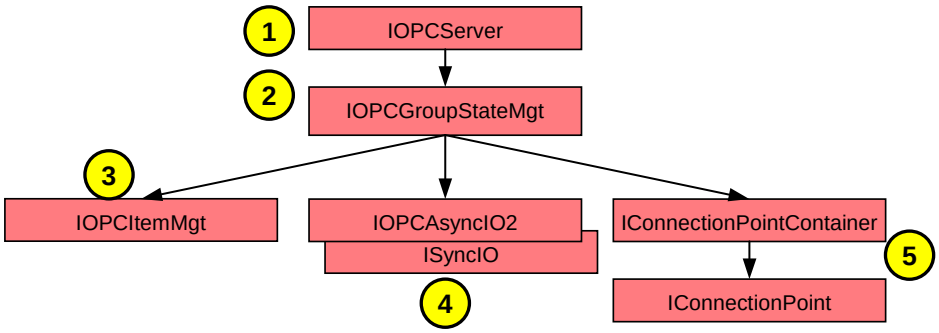
The procedure for connecting an .NET-OPC client to the OPC server as well as creating OPC items is identical to the procedure as with a C++ application.

In order to exchange data between OPC client and OPC server, a "Connection“ must first be established.

This occurs via the OPC DataAccess interface, whose objects and connections are displayed simplified and incomplete in the following figure.

### Objects of the OPC DataAccess interface

Figure 4-6




### Description of the graphic

The OPC client performs the respective actions by pressing the respective operator buttons. The dependencies displayed in the above figure must of course be taken into account. The actions to be performed are:

Table 4-12

| No. | Instructions | Note |
|---|---|---|
| 1 | Creating an instance of the "IOPCServer“ interface | This informs the OPC server that an OPC client needs to be served.<br>**Note:**<br>The OPCServer object is managed by the OPCServerManagement class. |
| 2 | “IOPCServer“ provides an interface for the “IOPCGroupStateMgt“ interface.<br>Now one or several OPC groups must be added via this interface (“IOPCGroupStateMgt“). | All OPC objects directly required in this application by the IOPCGroupStateMgt interface (see Figure 4-6), are encapsulated in the OPCGroupManagement class.<br>**Note** |

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

| No. | Instructions | Note |
|---|---|---|
| 3 | “IOPCGroupStateMgt“ provides an interface for the “IOPCItemMgt“ interface.<br>The OPC items are created using this interface. | |
| 4 | “IOPCGroupStateMgt“ provides an interface to the “IOPCAsyncIO2“ interface.<br>For enabling asynchronous calls, an instance must be created for this interface (“IOPCAsyncIO2“). | |
| 5 | The OPC server is to inform the OPC client actively when the files asynchronously written in the controller have arrived.<br>In order to do this the OPC client must log in at the respective interface as recipient of "Messages". This is performed via the "IConnectionPointContainer“ interface or respectively the "IConnectionPoint“. | Like all "active" COM components, the OPC server provides an interface to the "IConnectionPointContainer“. An instance to "IConnectionPoint“ finally provides the active connection. |

**Note**

- If variables / OPC items must continuously be monitored, the OPC group in which the variables/OPC items are located must be switched "active". Further respective information is available in the OPC Specification \1\ or in chapter 7.3.
- OPC items (and the respective group) to be read synchronously from the CACHE must be "active". Further respective information is available in the OPC Specification \1\ .
- Operating the user interface is introduced in chapter 11.
- The steps to be performed for connecting .NET and ActiveX-Controls are given in chapter 8.2 and 8.3.
- The respective OPC functionalities used regarding an OPC group are encapsulated in the OPCGroupManagement class.
  It is nevertheless explicitly pointed out, that it may become necessary, to adjust this class in an own application.
  A general application capability of this class can not be guaranteed. Only one possible way of implementation is illustrated here.
  With regards to this, an OPC wrapper class for .NET will be provided which considerably simplifies the implementation of OPC clients.

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

## 4.5 Problem circles during creating of OPC clients and solutions in this application

### Introduction

Problems occurring during creating of OPC clients and how to solve them will be discussed below.

For the following chapter it is recommended to have the class diagram (chapter 4.2) available.

### 4.5.1 Simple usage of the OPC RCW

### Problem

How can the implementation work load for OPC clients be reduced ?

The implementation work load for accessing OPC objects and their methods in .NET hardly differs from that under C++.
Additionally the required resource management (memory) contains also sources for errors.

### Solutions realized in this application

Generally the access to OPC objects by means of own classes should be encapsulated in order to facilitate access.

.NET provides an additional aid facilitating “cleaning up“ of obsolete objects. This is the “IDisposable“ interface, used for realizing the IDisposable pattern (see chapter 7.4).

Deriving the respective class from this interface guarantees a uniform method call, which enables releasing all obsolete resources.

The final release of all resources is down to the .NET-Framework of the GarbageCollector. In version V1.1 it cannot be predicted from when on a memory area selected for release can actually be released!

COM resources however must always be released explicitly, as this resource is located in the unmanaged Code and the GrabageCollector has no access to it.

The “OPCServerManagement“ and “OPCGroupManagement“ classes (blue frame in the class diagram of the client) encapsulate the access to the OPC objects and its methods. Additionally, these classes implement the “IDisposable“ interface and hence support the release of all obsolete resources.

Code fragments for connecting to the OPC are available in chapter 7.1 to 7.3 of the commented source code. The sequence diagram for running the IDisposable pattern is available in chapter 7.4.

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

### 4.5.2 Identification of OPC items via the respective handles

**Problem**

Which structure is suitable for managing client and server handles ?

As already explained in chapter 3.4,client and server handles are used for identifying OPC items. This is a quick and simple way of managing these handles:

- Storing the Serverhandles in an array:
  private int m_srvHndl[] = new int[4];
- Creating client handles as constants:
  private const int CLNTHNDL1 = 0;
  private const int CLNTHNDL2 = 1;
  private const int CLNTHNDL3 = 2;...

The following problem cases result from this "management structure":

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

**Receiving data via OnDataChange (code excerpt)**

Figure 4-7

```
public void OnDataChange(int dwTransid, int hGroup, int hrMasterquality, int hrMastererror,
    int dwCount, int[] phClientItems, object[] pvValues, short[] pwQualities,
    OpcRcw.Da.FILETIME[] pftTimeStamps, int[] pErrors)
{
    try
    {
        if(hGroup == GROUPSERVERHANDLE1)
        {
            if(hrMastererror != 0 || hrMasterquality != 0)
            {
                throw new Exception(errorStr + "; Quality: " + GetQualityText(pwQualities[0]));
            }

            for(int xCount = 0; xCount < dwCount; xCount++)
            {
                switch(phClientItems[xCount])
                {
                    case CLNTHNDL1:
                        m_myControl1.val = pvValues[xCount];
                        break;
                    case CLNTHNDL2:
                        m_myControl2.val = pvValues[xCount];
                        break;
                    case CLNTHNDL3:
```

**Explanation**

For the above method it is soon apparent that for many OPC items (e.g. several 1000 OPC items) the limits of clarity are quickly reached (switch-case instruction) and errors in programming become more and more likely.

Additionally, it is difficult to expand the above code fragment. It must be imagined that a further OPC item must be added to the runtime. This however must already be known at the time of implementation (due to the switch-case instruction in OnDataChange).

### Writing an OPC item via a synchronous write command (code excerpt)

Figure 4-8

```
private void setResetFlagForSimulation(int reset)
{
    // ************************************
    // set reset of simulation
    // ************************************
    int[] phServerHandle = new int[1];
    phServerHandle[0] = srvHndl[3];
    object[] phItemValue = new object[1];

    // here we don't read the item, because it is not connected with a
    // specific userinterface element
    phItemValue[0] = "0";

    int[] errors = null;

    try
    {
        m_mgtConvSetting.writeSync(phServerHandle, phItemValue, out errors);
    }
    catch (Exception)
    {
        throw;
    }
```

### Explanation

Here it becomes apparent, that the position of the server handle in the management array must for the respective action already be known at the time of implementation.
If at a later development time another OPC item is added, then the code of the “setResetFlagForSimulation“ code must be changed as the index of the server handle within the array has changed. Simple expandability cannot be ensured anymore.

### Solutions realized in this application

Encapsulating data, describing an OPC item, into its own class offers almost unlimited expandability. This has been realized by means of the “OPCItemExtender“ class.

All OPCItemExtender objects are saved in an object of the OPCItemExtenderList class, which in return is a specialization of the System.Collections.ArrayList class of the .NET frameworks.

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

### Conventions of the OPCItemExtender and OPCItemExtenderList objects

In order for the client and server handles not to be managed separately, the following conventions must be made:

- The client handle of an OPC item is the index of an OPCItemExtender object within the OPCItemExtenderList object.
- The server handle of an OPC item is also saved in the OPCItemExtender class.
- There are two ways to receive an OPCItemExtender object from the OPCItemExtenderList:
  - Index access to the list (index corresponds to the client handle)
  - Specifying the ITEM_ID, which should always be known in the application (z.B. in form of constants, #define directives, INI files)

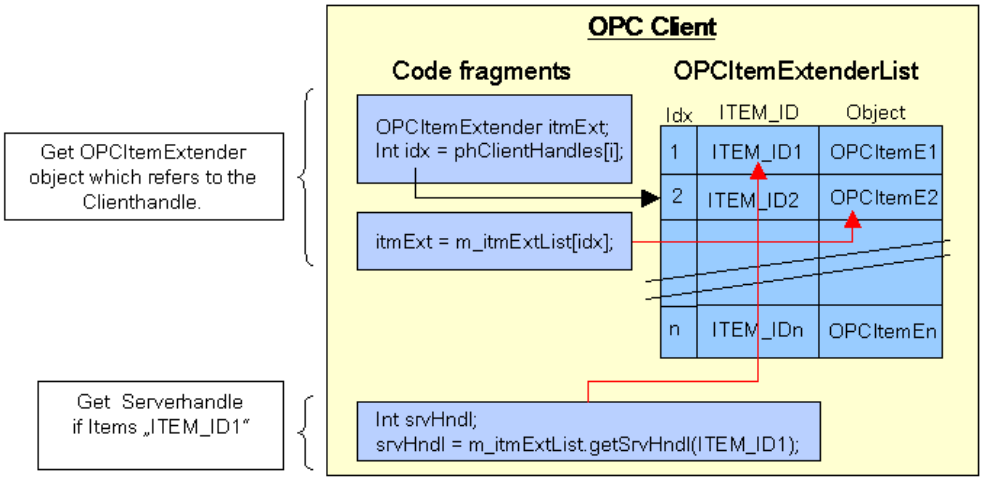
### The Hashtable of the OPCItemExtenderList

For also enabling performing the last point (accessing an OPCItemExtender object via its ITEM_ID) at over 1000 objects with acceptable performance, the OPCItemExtenderList contains a hash table which contains the ITEM_ID as keys and a reference to the respective OPCItemExtender object as value. This enables accessing any OPCItemExtender object in constant time (O(1) complexity).

### Access to a particular OPCItemExtender object via the hash table

The following figure schematically illustrates the above. This also enables code constructs discussed in the following blocks.

Figure 4-9




Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

### Receiving data via OnDataChange (code excerpt)

Figure 4-10

```
public void OnDataChange(int dwTransid, int hGroup, int hrMasterquality, int hrMastererror,
    int dwCount, int[] phClientItems, object[] pvValues, short[] pwQualities,
    OpcRcw.Da.FILETIME[] pftTimeStamps, int[] pErrors)
{
    // Data has been received
    statusBar.Text = m_ResMan.GetString("EXT_OPC_ONDATACHANGE_RECEIVED");
    string errorStr = "";
    string strValues = "";
    try
    {
        if(hGroup == m_mgtBoxes.srvHandle )
        {
            if(hrMastererror != 0 || hrMasterquality != 0)
            {
                throw new Exception(errorStr + "; Quality: " + GetQualityText(pwQualities[0]));
            }

            OPCItemExtender itm;
            for(int xCount = 0; xCount < dwCount; xCount++)
            {
                itm = (OPCItemExtender)(m_itmExtBoxes[phClientItems[xCount]]);
                itm.actValue = pvValues[xCount];
            }
        }
    }
    catch (Exception ex)
```

### Explanation

The value of the delivered OPC item is written into the OPCItemExtender object via an “actValue“ property, which is indexed over the respective client handle.

It becomes apparent, that only newly added OPC items require no immediate code adaptation of the OnDataChange method.

Another advantage of the here described solution is the fact, that the value of an OPC item is now available at any time, as it is stored in the respective OPCItemExtender object (which can be addressed via the known ITEM_ID).

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

### Writing an OPC item via a synchronous write command (code excerpt)

Figure 4-11

```
private void setResetFlagForSimulation(int reset)
{
    // ************************************
    // set reset of simulation
    // ************************************
    int[] phServerHandle = new int[1];
    // get the server handle of the item with item-id ITEM_RESET
    phServerHandle[0] = m_itmExtConvSettings.getSrvHandle(ITEM_RESET);
    object[] phItemValue = new object[1];

    // here we don't read the item, because it is not connected with a
    // specific userinterface element
    phItemValue[0] = reset.ToString();

    int[] errors = null;

    try
    {
        // write it down
        m_mgtConvSetting.writeSync(phServerHandle, phItemValue, out errors);
    }
    catch (Exception)
```

### Explanation

The code extract shows how the server handle for the OPC item “ITEM_RESET“ is read from the respective OPCItemExtenderList object. Irrespective of whether further OPC items are added in the later course, the correct OPC item is always written at this point. Further code extracts on “Reading and Writing of OPC Items“ are available in chapter 7.3.

**Note**

In order to fully provide the required expandability, respective methods must of course be implemented the OPCItemExtenderList class.

Only the methods necessary for this application have been implemented in this application. For example, it is not possible within this application to remove individual, already added OPC items during runtime, as the respective methods have not been implemented.

With the existing class structure however, there shouldn't be any larger problems regarding expandability of the classes.

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

### 4.5.3 Allocating the value of an OPC item to user-defined objects

**Problem**

The class structure explained in chapter 4.5.2 provides no solution to a specific problem:

Which user interface element is meant to indicate the value of a certain OPC item ?

At this point you are again dependent on a switch-case construct, as already illustrated in chapter 4.5.2. Often, certain user interface elements request the value to be displayed as String value, other elements require a Boolean value. At least a switch-case regarding the type of an OPC item is necessary.

The next question follows, if the value of an OPC item is not to be displayed at all, but for example to be saved in a database.

The respective question is: How can a connection of any objects to values of OPC items be realized ?

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

**Solutions realized in this application**

The answer to this questions is given by the object orientated languages: Polymorphism.

The “IOPCConnector::IOPCItemControl“ used in this application, declares a “OPCParam“ property which expects a string as a type.

**Note** In this application this was sufficient; for other OPC clients it may be useful to declare the type as object or instead a property using methods, which are overloaded accordingly.

If now a .NET class is derived from this interface (C# modifier must of course be considered), then this class of the property must implement this interface. Such a class is for example OPCActiveXConnector.

**Note** In this application, all classes implementing the “IOPCConnector::IOPCItemControl“ interface have the suffix “Connector“.

**Application example**

In the following, we assume that m_actXConn is an object of the OPCActiveXConnector class, to which an ActiveX control and a Label object is transferred (better: the reference to it).

If the “OPCParam“ property is now set by the m_actXConn object, then m_actXConn writes the transferred value into the ActiveX control and the label object.

### Using polymorphism

Due to the fact, however, that it is not desirable to memorize all of the connector objects separately, polymorphism occurs.

The OPCItemExtender class contains a member

private IOPCItemControl m_correspondingObject;

Any object of a class which implements the "IOPCConnector::IOPCItemControl" object. Due to the fact that each Connector class implements this interface, any object of any Connector class can also be specified here.

When setting the ("still abstract") property the type of object is detected via "late-binding", and the respective ("now concrete") property is set.

### Writing OPC item values into Connector objects (code extract)

The following code fragment of the OPCItemExtenderList class shows how simple writing of OPC item values into any Connector objects is. The "actValue" property of an OPCItemExtender object is hereby set in the OnDataChange method (see chapter 4.5.2 "OnDataChange" Code Fragment).

A code fragment indicating the generation of OPCItemExtender objects and Connector objects is available in chapter 7.2.

Figure 4-12

```
public string actValue
{
    // only returns the last written value! (not of
    // this is necessary to have the chance to have
    get{ return m_itmValue; }
    set
    {
        m_itmValue = value;
        if(m_correspondingObject != null)
            m_correspondingObject.OPCParam = value;
    }
}
```

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

# 5 Data flow models of this application

**This chapter explains ...**

how certain data flows of this application are organized.

**Discussed topics**


| Chap. | Title | Page |
|---|---|---|
| 5.1 | Asynchronous write procedure using .NET control | 58 |
| 5.2 | Monitoring a box on the conveyor belt (.NET-Control) | 62 |
| 5.3 | Synchronous read process with the ActiveX-Control | 65 |


## 5.1 Asynchronous write procedure using .NET control

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

**Frame conditions**

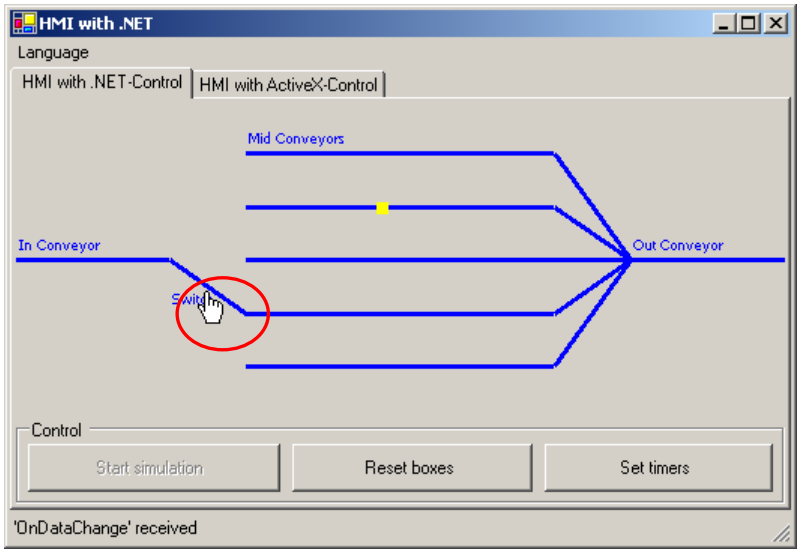
Now the processes for a switch change are described. The following general requirements apply:

- The OPC clients remains operable during writing the variables into the controller
- The user interface is only updated if the data has actually been written to the controller.

**Initial situation**

The following figure illustrates the initial situation:

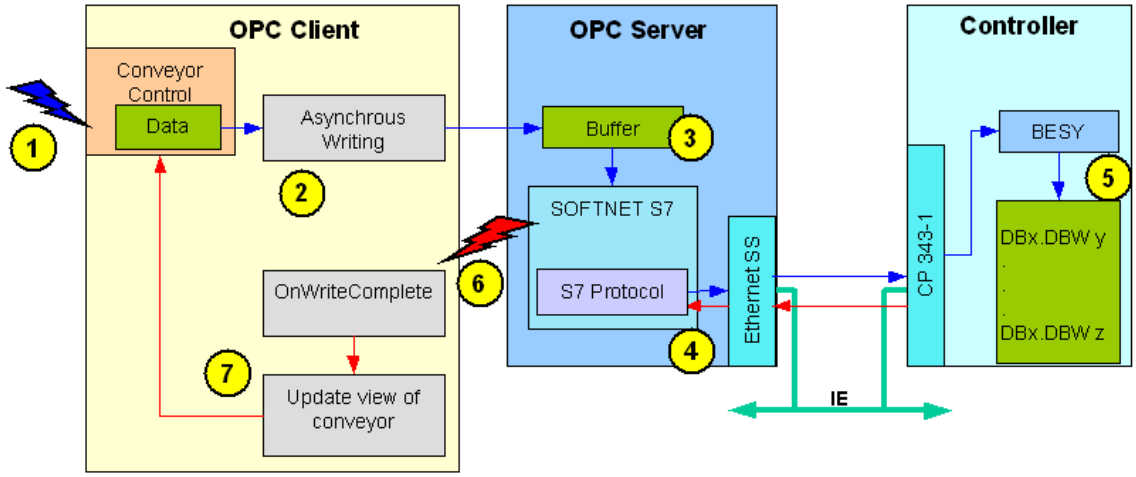
Figure 5-1



## Data flow model of the entire application

The following overview roughly illustrates the data flow of all involved components:

Figure 5-2




## Description

Table 5-1

| No. | Instructions | Note |
|---|---|---|
| 1 | Clicking the switch triggers an event. This prepares data for transfer to the OPC server. | |
| 2 | Data are transferred to the OPC server via an asynchronous write command. | Calling a function which executes the write command asynchronously, keeps the user interface operable. |
| 3 | Due to the fact, that several OPC clients might send data to the controller simultaneously, the data is buffered by the OPC server. Furthermore, the communication from the OPC server is optimized. | The data is forwarded to the S7 protocol as soon as possible. |
| 4 | Using the SIMATIC NET SOFTNET-S7, the data is sent to the controller with the S7 protocol via Ethernet network card. | The CP343-1 signals reception of the data to the Ethernet network card. |
| 5 | The SIMATIC operating system writes the values into the respective data block. | |
| 6 | The OPC server signals the OPC client the acknowledgement through the CP343-1, by calling the “OnWriteComplete“ method via the respective interface. | The OPC client can now assume, that the data are located in the respective data block or data area. |
| 7 | Since the OPC client now knows that the data has been written to the S7/CPU, it updates the view of conveyor and switch. | |

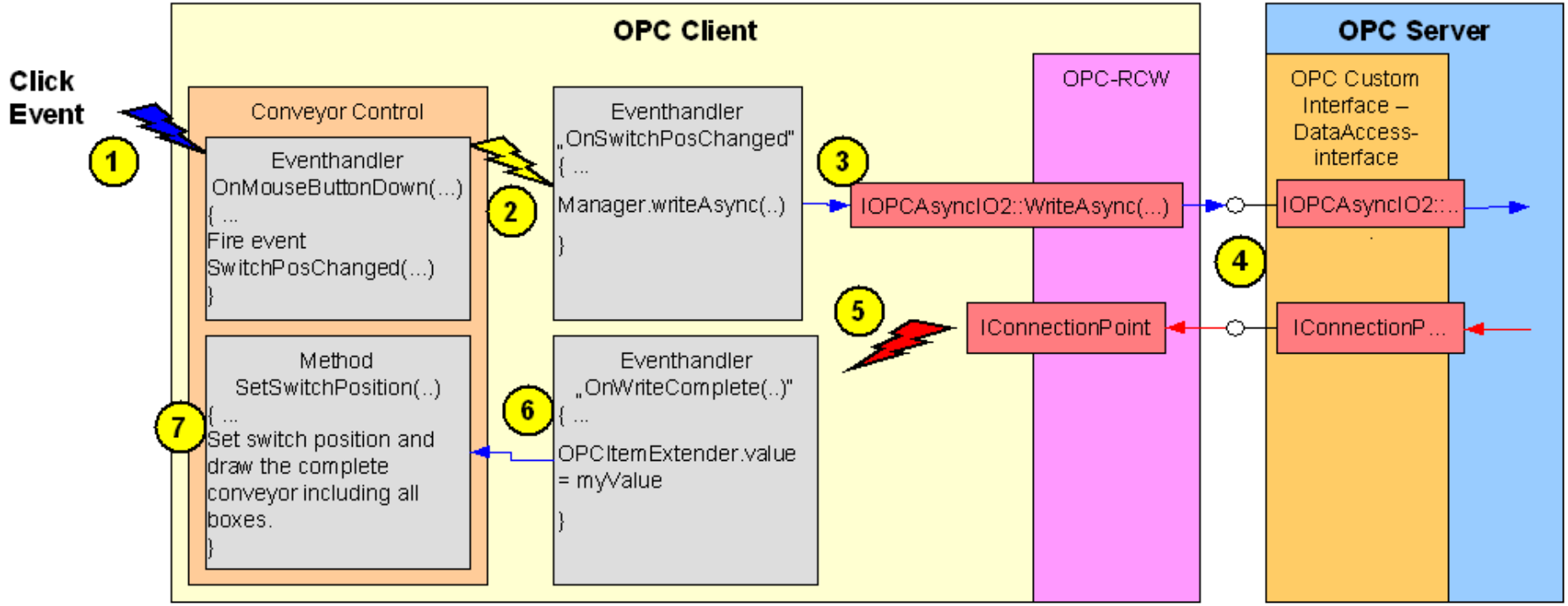
Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

Copyright © Siemens AG 2005 II rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

### Data flow model within the user interface

Here you can see the data flow within the OPC client:

Figure 5-3



### Description

Table 5-2

| No. | Instructions | Note |
|---|---|---|
| 1 | Clicking the switch calls the event handler “OnMouseButtonDown(...)“ of the “Conveyor“ control. | |
| 2 | Calling the user defined “SwitchPosChanged“ event informs the user interface, containing the conveyor control, of the switch position having to change.<br><br>**Note:**<br>Effectively this information is communicated to all .NET components logged in at the “SwitchPosChanged“ event handler. | The position of the switch is passed on via a parameter of the event. |
| 3 | In the “OnSwitchPosChanged“ event handler the OPCManager of the respective group is commissioned with executing an asynchronous write command.<br>In order to do this, the server handle must be fetched from the OPCItemExtenderList. This is performed via the ITEM_ID of the switch. (see chapter 4.5.2) | Using the WriteAsync method keeps the user interface operable. It can react to user input and other events if this is desired. |
| 4 | The OPC server writes the value of the variable into the controller.<br>Furthermore, it informs the OPC client of data having arrived in the CP of the controller. | |
| 5 | The “OnWriteComplete(...)“ event handler now determines the switch position written last.<br><br>**Note:**<br>This is realized in the OPC client via a “shadow variable“ – **no** ReadSync() or ReadAsync() is performed. | Registering at the IConnectionPointContainer enables the OPC client to receive events of this OPC server. |
| 6 | The switch position is transferred to the conveyor control. | |
| 7 | The conveyor control only draws its user interface. | |

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

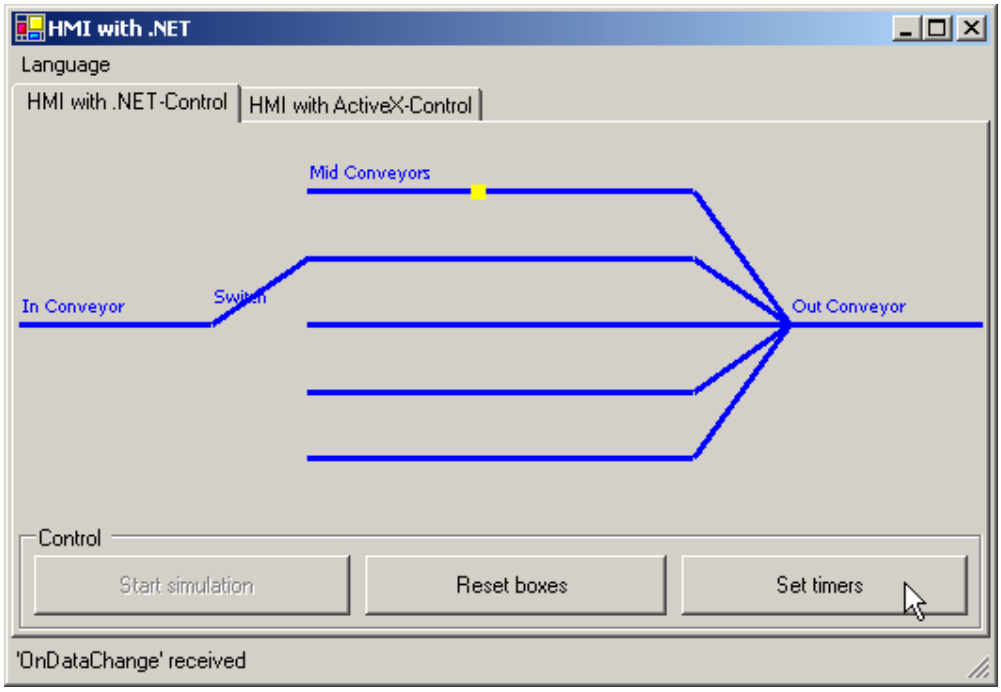
## 5.2 Monitoring a box on the conveyor belt (.NET-Control)

Now the data flow is described which occurs at a change of box position.

### User interface of the .NET-Controls "Conveyor"

The following figure shows the user interface of the .NET-Controls:

Figure 5-4



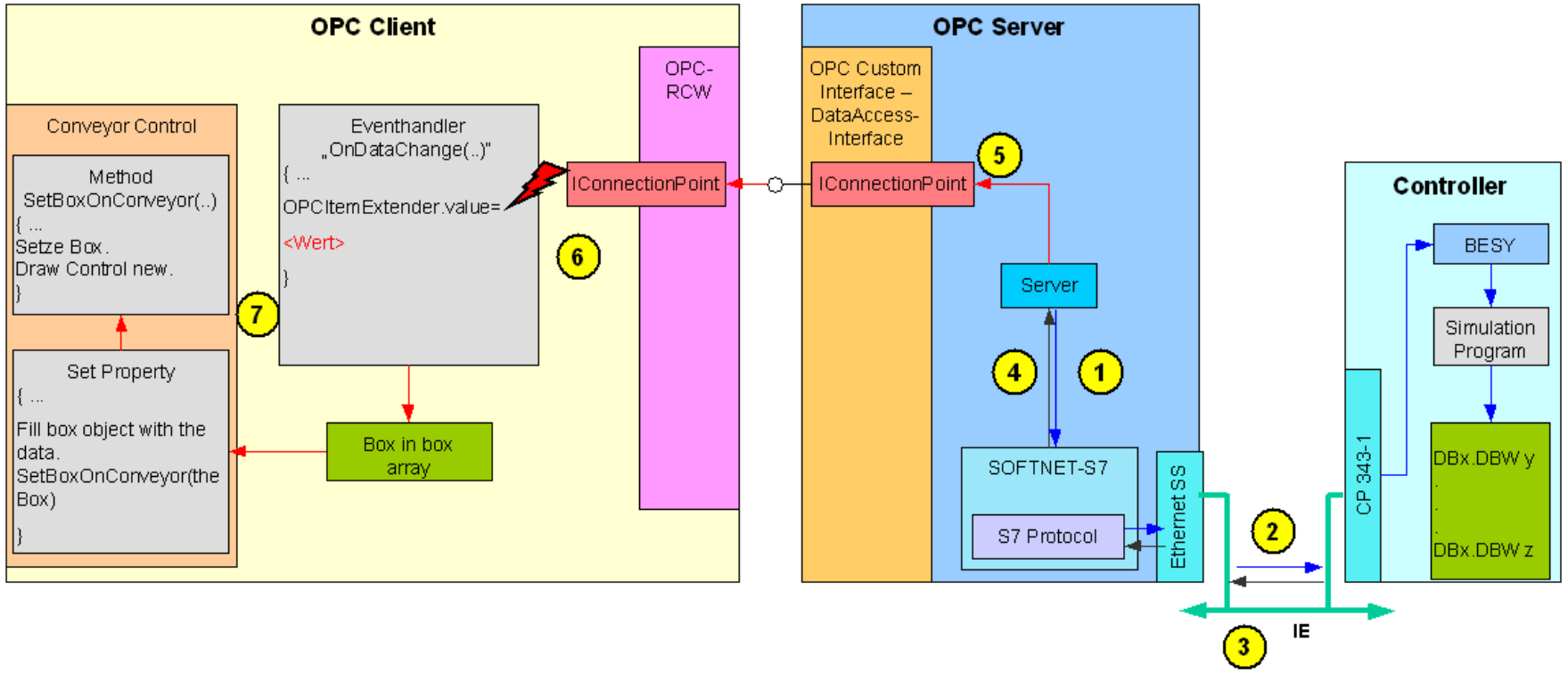

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

Copyright © Siemens AG **Fehler! Unbekannter Name für Dokument-Eigenschaft.** All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc



## Data flow

Figure 5-5



### Description

Table 5-3

| No. | Instructions | Note |
|---|---|---|
| 1 | The server polls (see Glossary) the current variables. | |
| 2 | The controller processes the job of the OPC server. | |
| 3 | Data are transferred to the OPC server. | |
| 4 | The server processes the data and checks whether they have changed. | |
| 5 | If the data has changed, the server uses the IConnectionPointContainers to call the “OnDataChange“ method of all OPC clients logged in at the “Sink-Interface“ (see Glossary). | The COM mechanisms on which this is based are available in the numerous secondary literature. |
| 6 | The OPC-RCW forwards the event of the server to the actual .NET-OPC client.<br>The “OnDataChange“ now writes the data into the respective box object in the box array. | Due to the fact that the “Box“ class implements the “IOPCItemControl“ interface it is sufficient here to set the property of the OPCItemExtender object. |
| 7 | The “SetBoxOnConveyor(..)“ method of the conveyor control is called. The parameter is a reference to the box object in the box array.<br>This method causes the conveyor control to determine the new position of the box and to redraw it. | |

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

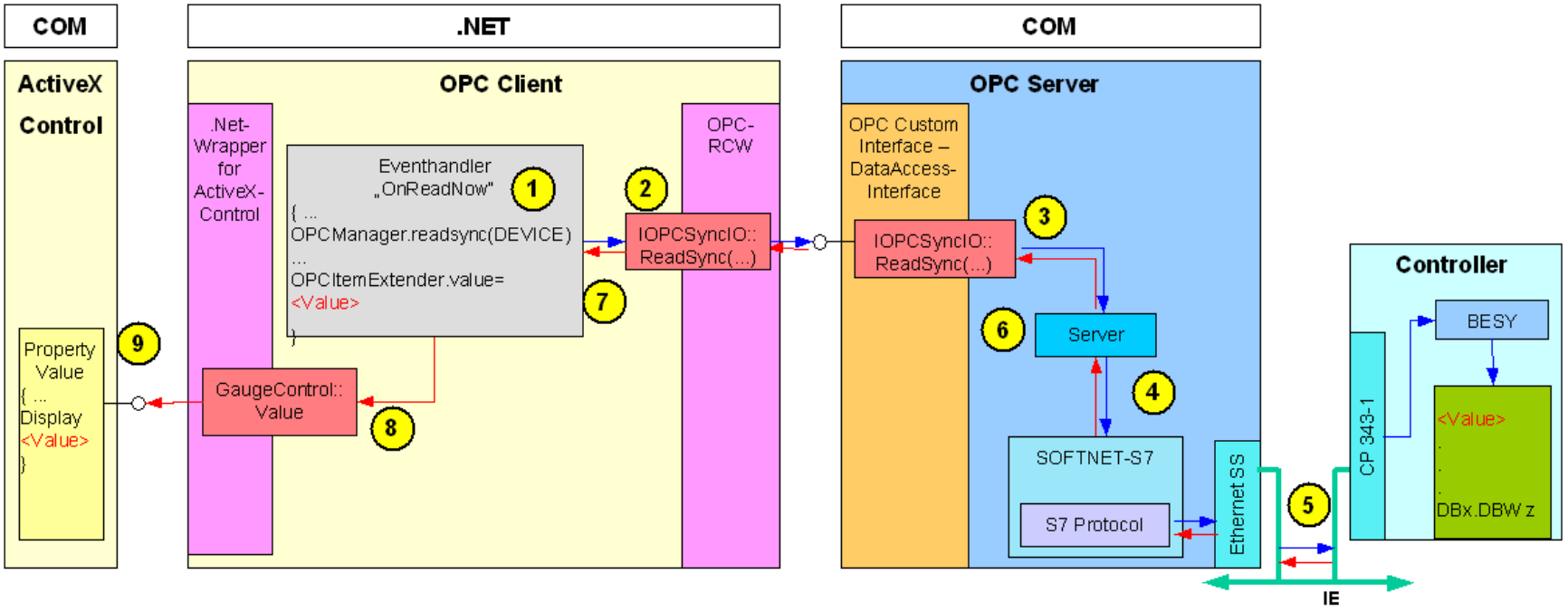


## 5.3 Synchronous read process with the ActiveX-Control

The following figure contains a description of the data flow if the current belt speed is read from the DEVICE.

### Data flow

Figure 5-6



### Description

Table 5-4

| No. | Instructions | Note |
|---|---|---|
| 1 | The “ReadNow“ button was pressed; this calls a synchronous read job via the OPC manager. | The write job is to be executed to DEVICE. |
| 2 | The OPCManager calls the “ReadSync“ method via the IOPCSyncIO interface provided by the OPC-RCW. | |
| 3 | The server accepts the job and performs a read job for the respective process variables. | The involved process variables are Timer1 to Timer3. |
| 4 | The driver and protocol blocks pack the job into an S7 protocol. SOFTNET-S7 is used for sending the job to the controller via the network card. | If reading from CACHE, there is no communication with the controller. The values existing in the CACHE would be returned. |
| 5 | The controller sends the desired variables to the OPC server. | |
| 6 | The server returns the read values. | |
| 7 | The OPC client assigns the value to the “correspondingObject“ of the respective OPCItemExtender object. | The “correspondingObject“ being of the IOPCItemControl type, the respective method (or property) of the underlying control is called (or set). |
| 8 | The data are transferred to the ActiveX-Control via the .NET-wrapper of the ActiveX-Control. | |
| 9 | The Active-Control accepts the value and causes the value to be displayed. | |

**Note** The ActiveX components are automatically integrated by the system.

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

# 6 Explanations on the Simulation Program

**Here you will find information on …**

how exactly the simulation has been realized in the controller.

**Discussed topics**


| Chap. | Title | Page |
|---|---|---|
| 6.1 | Basic structures of the simulation | 67 |
| 6.2 | Process of simulation | 69 |


## 6.1 Basic structures of the simulation

The program at the controller simulates the movement of boxes on the defined conveyors, by incrementing the position of the boxes on a conveyor section after a certain time has elapsed.

The following structure is used:

**Note** Connecting the structures with the visualization elements is explained in chapter 4.1.1.

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

**Structure "structBox2" (UDT1)**

Here, as apparent in the figure below, the last and the current conveyor on which the box is or was located are described. Furthermore, a horizontal position is assigned to the box (see chapter 4.1.2).

Figure 6-1

| | | | | |
|---|---|---|---|---|
| 0.0 | | STRUCT | | |
| +0.0 | nLastConveyor | INT | -1 | describes the conveyor the box lays on (last) |
| +2.0 | nActConveyor | INT | -1 | describes the conveyor the box lays on (actual) |
| +4.0 | nHorPos | INT | 0 | describes the horicontal position |
| =6.0 | | END_STRUCT | | |

### Structure “structSettings“ (UDT2)

Especially the variables are available for influencing the simulation.

Table 6-1

| Variable | Description |
|---|---|
| MidConveyors | Sets the number of mid conveyors (see chapter 4.1.1 and 4.1.2) |
| NumberOfBoxes | Sets the number of simulated boxes |
| TimeForConv1 - 3 | Initial times in 10ms after which the positions of the boxes are incremented on the respective conveyor section |

### Structure “structSettings“ (UDT2) – Data view

Figure 6-2

| | | | | |
|---|---|---|---|---|
| 0.0 | | STRUCT | | |
| +0.0 | ResetSimulation | BOOL | FALSE | Resets the simulation |
| +2.0 | MidConveyors | INT | 2 | number of conveyors in the mid of the conveyor-graph |
| +4.0 | SwitchPosition | INT | 1 | starting position of the switch |
| +6.0 | NumberOfBoxes | INT | 1 | number of boxes to simulate |
| +8.0 | TimeForConv1 | INT | 100 | simulation time for conveyor 0 && 1 in 10ms |
| +10.0 | TimeForConv2 | INT | 150 | simulation time for mid conveyors in 10ms |
| +12.0 | TimeForConv3 | INT | 200 | simulation time for end conveyor in 10ms |
| =14.0 | | END_STRUCT | | |

### Structure “structTimers“ (UDT3)

This structure is only used for visualizing or controlling the times of the conveyor sections. Serves exclusively for improved separation of simulation settings, however, is not urgently necessary.

Figure 6-3

| | | | |
|---|---|---|---|
| 0.0 | | STRUCT | |
| +0.0 | Timer1 | INT | 0 |
| +2.0 | Timer2 | INT | 0 |
| +4.0 | Timer3 | INT | 0 |
| =6.0 | | END_STRUCT | |

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

## 6.2 Process of simulation

### Description of the process

The processes within the controller are quite simple.

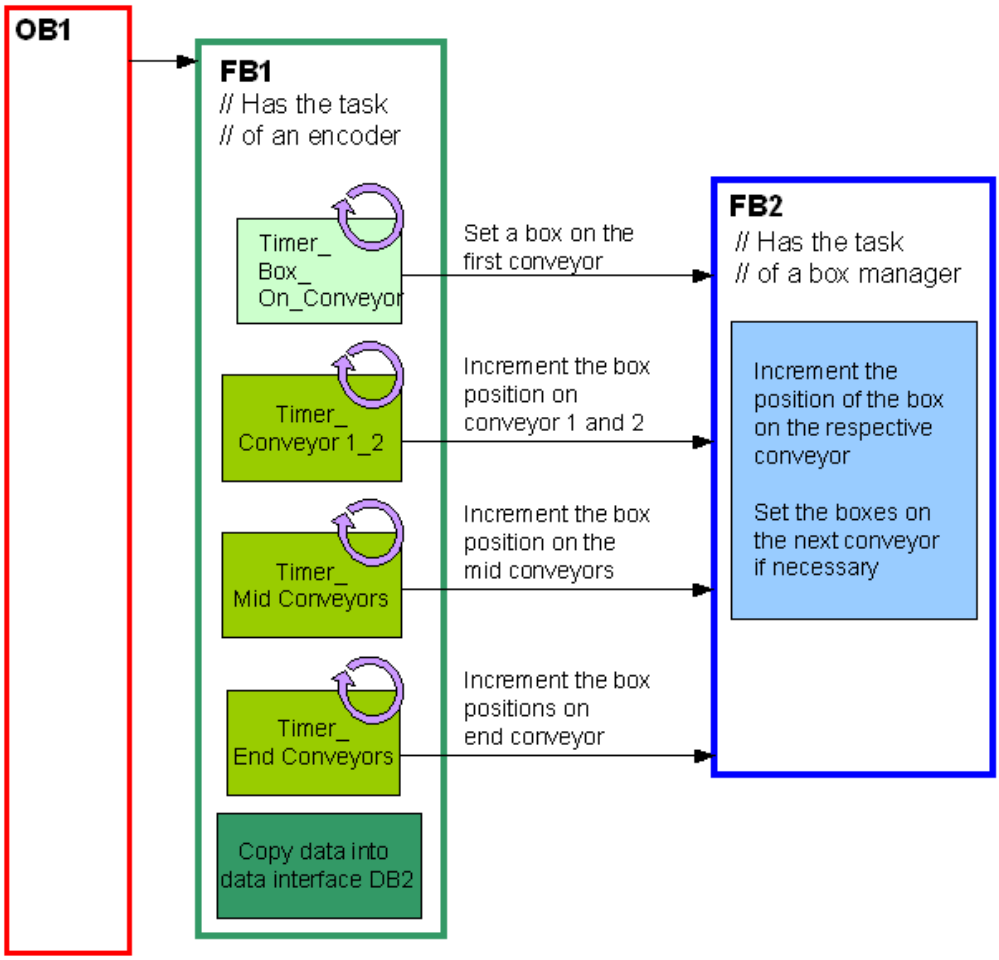
OB1 cyclically calls the FB1 block “Simulation“, whose only task is monitoring or controlling the process of the respective Timer, as well as storing data in DB2 “DataBoxes“. Thereby, the DB2 “DataBoxes“ serves as data interface with the OPC server.

In order to provide movement of the individual boxes, the FB1 “Simulation“ calls the FB2 “BoxSimulator“. Depending on the elapsed time of the Timer the position of the respective boxes is incremented.

### Call path of the blocks

The following diagram illustrates the call path:

Figure 6-4




Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

### The block FB2 “BoxSimulator”

The block FB2 “BoxSimulator” manipulates the boxes, whereby the FB1 ”Simulator“ reports to this block, via the parameters ”nInConv1“ and ”nInConv2“, for which conveyor section the respective Timer has elapsed. The following section shows the relevant SCL code of FB2.

### The block FB2 “BoxSimulator” (code extract)

```
VAR
    Boxes : ARRAY[0..4] OF udtl;
END_VAR


BEGIN

    IF bInReset = true THEN
        FOR nIdx:=0 TO MAX DO
            Boxes[nIdx].nLastConveyor := -1;
            Boxes[nIdx].nActConveyor := -1;
            Boxes[nIdx].nHorPos := 0;
        END_FOR;
    ELSE
        // If a box is to set on the first conv...
        nIdx := nInBoxToSet - 1;
        IF nIdx > -1 THEN
            Boxes[nIdx].nLastConveyor := -1;
            Boxes[nIdx].nActConveyor:=0;
            Boxes[nIdx].nHorPos := 0;
        ELSE
        // examine all boxes and their positions
            nTmp := nInBoxes-1;
           FOR nIdx:=0 TO nTmp DO
                IF (Boxes[nIdx].nActConveyor >= nInConv1) AND (Boxes[nIdx].nActConveyor < nInConv2) THE
                    // increment position
                    Boxes[nIdx].nHorPos := Boxes[nIdx].nHorPos + 1;
                    CASE Boxes[nIdx].nHorPos OF
                        END_MEAS_CONV1:
                            Boxes[nIdx].nLastConveyor := 0;
                            Boxes[nIdx].nActConveyor := 1;
                        END_MEAS_CONV2:
                            Boxes[nIdx].nLastConveyor := 1;
                            Boxes[nIdx].nActConveyor := 1 + nInSwitchPos;
                        END_MEAS_CONV3:
                            Boxes[nIdx].nLastConveyor := Boxes[nIdx].nActConveyor;
                            Boxes[nIdx].nActConveyor := Boxes[nIdx].nActConveyor + nInMidConveyors;
                        END_MEAS_CONV4:
                            Boxes[nIdx].nLastConveyor := Boxes[nIdx].nActConveyor;
                            Boxes[nIdx].nActConveyor := nInMidConveyors*2 + 2;
                        END_MEAS_CONV5:
                            // Set the box back on the first conveyor
                            Boxes[nIdx].nLastConveyor := -1;
                            Boxes[nIdx].nActConveyor := 0;
                            Boxes[nIdx].nHorPos := 1;
                    END_CASE;
                END_IF;
            END_FOR;
        END_IF;
    END_IF;
```

Here the boxes are moved back and forth between the conveyors

Figure 6-5

**Note** For further information regarding the realization of the simulation please refer to the source code.

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

# 7 Code Fragments of the OPC Clients

**Here you will find information on …**

the concrete action to be taken in order to provide operator control and monitoring with OPC variables.

Especially creating groups and items as well as activating and deactivating of groups is discussed.

**Note**

- All code fragments have been divided into “regions“.
- For further information, please refer to the commented source code.

**Discussed topics**


| Chap. | Title | Page |
|---|---|---|
| 7.1 | Introduction and basics | 71 |
| 7.2 | Connecting to the OPC server | 74 |
| 7.3 | Reading and Writing OPC Items | 80 |
| 7.4 | Release of references on the OPC server | 85 |
| 7.5 | Internationalization | 87 |


Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

## 7.1 Introduction and basics

### Source code files of the OPC Client

The zip.-file also contains the created source code files:

Table 7-1

| Component | File | Note |
|---|---|---|
| Complex client – Main user interface | Complex-Client-Form.cs | Contains the main user interface |
| | Complex-Client-Form.de.resx | German resources for the main user interface |
| | Complex-Client-Form.en.resx | English resources for the main user interface |
| | Complex-Client-Form.resx | Language independent resources for the main user interface |
| | Complex-Client_res_de.resX | File for German texts not directly located in the main user interface |
| | Complex-Client_res_en.resX | File for English texts not directly located in the main user interface |
| | OPCActiveXConnector.cs | Class for connecting the ActiveX-Control used in the user interface with the OPC items |
| | OPCItemExtender.cs | Class representing certain properties of an OPC item |
| | OPCItemExtenderList.cs | Class containing a list of OPCItemExtender objects |

| Component | File | Note |
|---|---|---|
| | OPCServerManagement.cs | Class encapsulating the calls to the OPC server required in this application |
| | OPCGroupManagement.cs | Class encapsulating the calls via the OPC-DA interface |
| | SpeedsDialog.cs | Dialog for dynamic setting of the timers simulating the conveyor speed in the controller |
| | SpeedsDialog.de.resx | German resources for the dialog |
| | SpeedsDialog.en.resx | English resources for the dialog |
| | SpeedsDialog.resx | Language independent resources for the dialog |
| Complex-client - .NET-Control | ConveyorGraphControl.cs | File contains the implementation of the .NET-Control |
| | ConveyorGraphControl.de.resx | German resources for the .NET-Control |
| | ConveyorGraphControl.en.resx | English resources for the .NET-Control |
| | ConveyorGraphControl.resx | Language-independent resources for the .NET-Control |
| Complex-client - Interface | IOPCItemControl.cs | Interface providing standardized interface for connecting of OPCItemExtender objects and user interface elements. |

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

**Structures in the OPC client**

The OPC client uses structures instead of classes.

As far as sensible within the framework of this application, object-oriented methods are used.

The class diagrams as well as the tasks of the most important classes have already been discussed in chapter 4.2.

### Basic variables for connecting to the OPC server

In this application, the OPC client requires the following references (as fields) to the OPC-COM interfaces for connecting to the OPC server:

Table 7-2

| Function | Description |
|---|---|
| OPCServerManagement.cs | |
| Connection to the OPC server: | private IOPCServer m_OPCServer = null#*#;<br>private IOPCCommon m_OPCCommon = zero; |
| OPCGroupManagement.cs | |
| Connection to process variables | // Connecting to process variables<br>private IOPCGroupStateMgt m_OPCGroupStateMgt = zero;<br>private IOPCItemMgt m_OPCItem = zero;<br>// Asynchronous read/write<br>private IOPCAsyncIO2 m_OPCAsyncIO2 = zero;<br>// Synchronous read/write<br>private IOPCSyncIO m_OPCSyncIO = zero;<br>// Necessary for receiving events of the group<br>private IConnectionPoint m_OPCCP = zero;<br>private IConnectionPointContainer m_OPCCPContainter = zero; |
| Memory bits for the connection to ConnectionPointContainer (so-called cookies) | private int m_Cookie_CP = 0; |
| Handles | • to group objects:<br>private int m_grpSrvHndl = 0;<br>private int m_grpClntHndl = 0;<br>• to item objects:<br>In this application , the item-client handles correspond with the array-index of the respective OPCItemExtender object in the OPCItemsExtenderList. The server handles however, are also being memorized in the OPCGroupManagement class:<br>private int[] m_itmServerHandles; |

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

## 7.2 Connecting to the OPC server

### Connection to the OPC server:

The connection to the OPC server is made via the “connectOPCServer()“ method in the OPCServerManagement.cs file:

Figure 7-1

```
//Get the Type from ProgID.Text
typeofOPCserver = Type.GetTypeFromProgID(PROG_ID);

string strDummy = m_ResMan.GetString("EXT_OPC_CONNECT_SERVER");
if (typeofOPCserver == null) throw new Exception(strDummy);

// Must be freed with Marshal.ReleaseComObject(myOPCServer)
m_OPCServer = (IOPCServer) Activator.CreateInstance(typeofOPCserver);
if (m_OPCServer == null)
{
    strDummy = m_ResMan.GetString("EXT_OPC_CREATE_INST_FAILED");
    throw new Exception(strDummy);
}

// Don't free it - we don't AddRef it!!!
m_OPCCommon = (IOPCCommon) m_OPCServer;
if (m_OPCCommon == null)
{
    strDummy = m_ResMan.GetString("EXT_OPC_COMMON_FAILED");
    throw new Exception(strDummy);
}

m_OPCCommon.GetLocaleID(out m_LocaleID);
```

Determining the ProgID

Creating the COM instance on the OPC server

Required for Locale_ID

### Creating OPC groups

OPC group objects are added via the OPC server object. OPC groups are created using the “addGroup“ method of the OPCServerManagement class. This method, in return, uses the “addOPCGroup(...)“ of the OPCGroupManagement class and also returns the OPCGroupManagement object.

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

## Creating OPC groups – Code extract

The following code extract only shows the call for the “AddGroup“ method of the IOPCGroupStateMgt interface:

Figure 7-2

```
m_OPCServer.AddGroup(   m_grpName,
                        Convert.ToInt32(m_isActive),
                        m_reqUpdateRate,
                        m_grpClntHndl,
                        pTimeBias,
                        pDeadband,
                        m_LocaleID,
                        out m_grpSrvHndl,
                        out m_revUpdateRate,
                        ref iid,
                        out objGroup );
```

If true, the CACHE for this group is activated. This is a prerequisite for the "Monitoring functionality“.

The method returns a group object.

```
if (objGroup == null)
{
    strMsg = m_ResMan.GetString("EXT_OPC_IF_GROUP2") + m_grpName
    throw new Exception(strMsg);
}

// Get our reference from the created group
m_OPCGroupStateMgt = (IOPCGroupStateMgt) objGroup;
if (m_OPCGroupStateMgt == null)
{
    strMsg = m_ResMan.GetString("EXT_OPC_IF_GROUP4");
    throw new Exception(strMsg);
}
```

Save group object

**Note**

The required update rate (m_reqUpdateRate) need not correspond to the actual update rate!

Please refer the parameter “m_revUpdateRate“.

## Creating OPC items

Creating of items has been encapsulated within the “addOPCItems“ of the OPCGroupManagement class (in OPCGroupManagement.cs).

For better overview, the following points are observed:

- Filling the relevant parameters and the OPCItemExtender objects
- Treating COM return pointers in .NET
- Creating OPC items
- Error handling

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

### Filling the relevant parameters and the OPCItemExtender objects

In this application, the parameters for creating OPC items in certain methods are set. An example is filling the parameters for the OPC items of the OPC group “Timer” (method “buildItemsForTimers”, class) “OPCClientForm“, file: “Complex-Client-Form.cs“).

Figure 7-3

```
// ************************************
// Timers
// ************************************
// build items for the timers
int[] itmsClntHndl = new int[NUMTIMERS];
int[] itmsSrvHndl = new int[NUMTIMERS];
bool[] itmsActive = new bool[NUMTIMERS];
IOPCItemControl[] correspondingObject = new IOPCItemControl[NUMTIMERS];
string[] itmsID = new string[NUMTIMERS];
OPCItemExtender itm;

int i;

for(i=0; i < NUMTIMERS; i++)
{
    itmsClntHndl[i] = i;
    itmsActive[i] = true;

    // Items are given symbolically
    itmsID[i] = ITEM_TIMER_PREFIX + (i+1);
}
correspondingObject[0] = new OPCActiveXConnector(ctlWinCCGauge1,Timer1Label);
correspondingObject[1] = new OPCActiveXConnector(ctlWinCCGauge2,Timer2Label);
correspondingObject[2] = new OPCActiveXConnector(ctlWinCCGauge3,Timer3Label);

try
{
    // Now build the opc items for the conveyor
    initOPCGroupsAndItems(
        ref m_mgtTimers, GRP_NAME_TIMERS,
        reqUpdateRateTimers,null,
        itmsActive,itmsID,itmsClntHndl,out itmsSrv
}
catch(Exception)
{
    // Just forward it
    throw;
}
// save the opc-item information for
for(i=0; i< NUMTIMERS; i++)
{
    itm = new OPCItemExtender(itmsActive[i],itmsID[i], itmsSrvHndl[i], itmsClntHndl[i],
        ref correspondingObject[i]);
    m_itmExtTimers.Add(itm);
}
```

Filling the parameters:
- clientHandles
- aktive Items
- item-IDs

The OPC items correspond to certain objects of the user interface. The OPCActiveXConnector class must implement the OPCItemControl class

This method creates an OPCGroupManagement-object with the OPCServerManagement object and assigns it to "m_mgtTimers“. This generates the OPC items for the timers.

Here the OPCItemExtender objects were generated and saved in the respective ItemExtenderList "m_itmExtTimers“.

The respective code must now be executed for each group (see methods “initOPCVariables“ as well as “ApplyButton_Click“ of the “OPCClientForm“ class).

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

### Treating COM return pointers in .NET

Due to all OPC objects being COM objects, and return values in COM always having to be returned as COM pointers, preliminary work must be performed in order to be able to process these COM pointers in .NET.

To do this, .NET provides the class library “System.Marshal“ and “System.Interop“.

In this application, the C# type “IntPtr“ must be used for returned COM pointer:

Figure 7-4

```
IntPtr ppResults = IntPtr.Zero;
IntPtr ppErrors = IntPtr.Zero;
```

**Note**

The memory area for this pointer reserved by the OPC client with “Marshal.FreeCoTaskMem(<varName>)“ must be released again, as this reserved memory area consists of unmanaged code and is thus not subordinate to the GarbageCollector.

Further information is available in the secondary literature \2\.

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

### Creating OPC items

This code now shows how OPC items can be generated in .NET:

Figure 7-5

```
// Get Item interface
m_OPCItem = (IOPCItemMgt) m_OPCGroupStateMgt;
if (m_OPCItem == null)
{
    strDummy = m_ResMan.GetString("EXT_OPC_IF_ITEM");
    throw new Exception(strDummy);
}

// Now add the items
for(int i=0; i < numberOfItems; i++)
{
    // Accesspath not needed
    Itemdefs[i].szAccessPath = "";
    // AddItem Active, so OnDataChange will come in an active group for this item
    Itemdefs[i].bActive = Convert.ToInt32(itemsActive[i]);
    // We want to get the items as string
    Itemdefs[i].vtRequestedDataType = Convert.ToInt16(VarEnum.VT_BSTR);
    // "BinaryLargeOBject" not needed by SimaticNet OPC Server
    Itemdefs[i].dwBlobSize = 0;
    // no blob
    Itemdefs[i].pBlob = IntPtr.Zero;

    Itemdefs[i].hClient = itmClntHndls[i];
    Itemdefs[i].szItemID = itemIDs[i];
}

// Adding items to the Group
m_OPCItem.AddItems( numberOfItems,
                    Itemdefs,
                    out ppResults,
                    out ppErrors);
```

Get Items-Interface from Groups interface.

Assemble Item structure.
**Note:**
The item definitions are transferred by the calling method.

Add items

### Error handling

First the return values are to be tested:

Figure 7-6

```
if (ppResults == IntPtr.Zero)
{
    strDummy = m_ResMan.GetString("EXT_OPC_RESULT_FAILED");
    throw new Exception(strDummy);
}
if (ppErrors == IntPtr.Zero)
{
    strDummy = m_ResMan.GetString("EXT_OPC_ERROR_FAILED");
    throw new Exception(strDummy);
}
```

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

### Converting COM return values

When generating OPC items, different errors can now occur. The return values (as already mentioned) are COM pointers, therefore, they must be converted into .NET-compatible data types. The mentioned methods of "System.Marshal". The following sequence shows the individual steps:

Table 7-3

| Code section / Explanation |
|---|
| The errors delivered by the OPC server can be easily copied as it is a simple data type.<br>`//Evaluate return ErrorCodes to exclude possible Errors`<br>`int[] errors = new int[numberOfItems];`<br>`Marshal.Copy(ppErrors, errors, 0, numberOfItems);` |
| The "results", however, must be manually copied due to the complex OPCITEMRESULT structure (→ Custom Interface).<br>`OPCITEMRESULT[] result = new OPCITEMRESULT[numberOfItems];`<br>`IntPtr pos = ppResults;` |
| Copying occurs with a loop via the result array:<br>`for(int dwCount = 0; dwCount < numberOfItems; dwCount++)`<br>`{`<br>`  try`<br>`  {`<br>`  if (errors[dwCount] != 0)`<br>`  {`<br>`  strDummy = m_ResMan.GetString("EXT_OPC_ERROR_ITEM_ADD");`<br>`  throw new Exception(strDummy);`<br>`  }`<br>`  // Item was added succesfully`<br>`  result[dwCount] = (OPCITEMRESULT) Marshal.PtrToStructure(pos,`<br>`  typeof(OPCITEMRESULT));`<br>`  itemServerHandles[dwCount] = result[dwCount].hServer;`<br>`  }`<br>`  catch (Exception)`<br>`  {`<br>`  // Item was not added`<br>`  statusBar.Text = m_ResMan.GetString("EXT_EXC_MSG_ALL_CAPTION");`<br>`  throw;`<br>`  }`<br>`  pos = (IntPtr)(pos.ToInt32() + Marshal.SizeOf(typeof(OPCITEMRESULT)));`<br>`}`<br>Even a single error throws an excepton.<br>Copy result<br>The result array is run through with the pointer arythmetics of the old Windows world |
| The memory slot for the values delivered by the OPC server was reserved on the memory area. The must be released again by the OPC client.<br>`// Free allocated COM-ressouces`<br>`Marshal.FreeCoTaskMem(ppResults);`<br>`Marshal.FreeCoTaskMem(ppErrors);` |

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

## 7.3 Reading and Writing OPC Items

### Logging in the OPC clients for asynchronous methods

In order to be able to receive OPC server messages the OPC client must log in at the interface of the OPC server or the OPC group. This has been realized in the “setAsyncInterface“ method of the OPCGroupManagement class (in OPCGroupManagement.cs):

Figure 7-7

```
// Query interface for Async calls on group object
// Take care:   IOPCAsyncIO is only for DA 1.x
//              IOPCAsyncIO2 is for DA 2.x
//              IOPCAsyncIO3 is for DA 3.x
m_OPCAsyncIO2 = (IOPCAsyncIO2) m_OPCGroupStateMgt;
if(m_OPCAsyncIO2 == null)
    throw new Exception(m_ResMan.GetString("EXT_OPC_ASYNCIO2_FAILED"));

// Query interface for callbacks on group object
m_OPCCPContainer = (IConnectionPointContainer) m_OPCGroupStateMgt;
if(m_OPCCPContainer == null)
    throw new Exception(m_ResMan.GetString("EXT_OPC_CP_FAILED"));

// Establish Callback for all async operations
Guid iid = typeof(IOPCDataCallback).GUID;
m_OPCCPContainer.FindConnectionPoint( ref iid, out m_OPCCP);

if(m_OPCCP == null)
    throw new Exception(m_ResMan.GetString("EXT_OPC_CP_FAILED"));
// Creates a connection between the OPC servers's connection point and
// this client's sink (the callback object).
m_OPCCP.Advise(m_callbackReceiver, out m_cookie_CP);

if (m_cookie_CP == 0)
{
    string strDummy = m_ResMan.GetString("EXT_OPC_ADIVSE_FAILED");
    throw new Exception(strDummy);
}
```

Referencing the AsyncIO2-Interface of the

ConnectionPoint zum OPC-Callback-Interface holen.

Introducing the ConnectionPoint.

### Activating/deactivating an OPC group

In order to reduce the communication load between OPC server and controller, only the displayed or controlled variables should be updated automatically by the OPC server.

In order to realize this, the “IOPCGroupStateMgt“ interface provides the option of changing the status of the respective group.

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

### Code fragment – Activating an OPC group

The following code fragment shows how this can be implemented with .NET and what needs to be taken account of (method ”changeGroupState“, file: OPCGroupManagement.cs):

Figure 7-8

```
IntPtr pTimeBias = IntPtr.Zero;
IntPtr pDeadband = IntPtr.Zero;

// Access unmanaged memory
GCHandle hRequestedUpdateRate = GCHandle.Alloc(reqUpdateRate, GCHandleType.Pinned);
GCHandle hLoc = GCHandle.Alloc(Loc, GCHandleType.Pinned);
GCHandle hActive = GCHandle.Alloc(Active, GCHandleType.Pinned);
GCHandle hServerGroup = GCHandle.Alloc(m_grpSrvHndl, GCHandleType.Pinned);

try
{
    // Set state
    if (activateGroup)
        hActive.Target = 1;
    else
        hActive.Target = 0;

    // Because we have to allocate unmanaged memory, we have to pinn them. Otherwise .Net's
    // garbage collector might free the memory and the opc server will do something randomly.
    m_OPCGroupStateMgt.SetState(hRequestedUpdateRate.AddrOfPinnedObject(),
                                out revUpdateRate,
                                hActive.AddrOfPinnedObject(),
                                // if the parameter is used, it must be pinned!!!
                                pTimeBias,
                                // if the parameter is used, it must be pinned!!!
                                pDeadband,
                                hLoc.AddrOfPinnedObject(),
                                hServerGroup.AddrOfPinnedObject());
}
catch (Exception)
{
    throw;
}
finally
{
    // now free the unmanaged memory
    if(hActive.IsAllocated) {hActive.Free();}
    if(hLoc.IsAllocated) {hLoc.Free();}
    if(hRequestedUpdateRate.IsAllocated) {hRequestedUpdateRate.Free();}
    if(hServerGroup.IsAllocated) {hServerGroup.Free();}
}
```

The OPC Server will access the .NET-storage areas. In order for the GarbageCollector not to release these memory areas, they must be "pinned" (see comment at "SetState“).

**Note** The necessity for “pinning down” memory was briefly explained in chapter 3.5.3.

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

### Reading and Writing OPC Items

OPC items are read and written with interfaces which provide the OPCGroupStateMgt interface. Therefore this functionality has been encapsuled in the OPCGroupManagement class (file: OPCGroupManagement.cs).

### Example: Writing synchronously

The “writeSync“ method writes the “itemValues“ transferred to it into the OPC items identified with the ”itemServerHandles”.

Figure 7-9

```
// sync write will block the user interface! so the ui is not responsive.
// therefore it should only be used if the user is not allowed to control the ui.
m_OPCSyncIO.Write(  numItems,
                    itemServerHandles,
                    pItemValues,
                    out pErrors);

errors = new int[numItems];
//check if an error occured
Marshal.Copy(pErrors, errors, 0, numItems);

for(int xCount = 0; xCount < numItems; xCount++)
{
    // Data has been received
    if(errors[xCount] != 0)
    {
        // Errors occured - raise Exception
        m_OPCServer.GetErrorString(errors[xCount], m_LocaleID, out ErrorStr);
        System.Diagnostics.Debug.WriteLine(ErrorStr);
    }
}

if(ErrorStr != "")
    throw new Exception(ErrorStr);

// Free allocated COM-ressouces
Marshal.FreeCoTaskMem(pErrors);
```

Calling the write method of the OPC-SyncIO Interface.

Error handling

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

### Example: Asynchronous writing

The asynchronous writing occurs via the OPC-IAsyncIO2 interface. This can be realized as follows:

### The call

First the values are written asynchronoulsy into the controller.

Like for all read and write operations, the “ItemServerHandles“ must also be given. In the following case, they are transferred via the “itemServerHandles“ parameter. The called function is hence responsible for the correct parameter supply.

Figure 7-10

```
// We never use dwCancelID anymore
int dwCancelID = 0;

transactionID++;

m_OPCAsyncIO2.Write(    numItems,
                        itemServerHandles,
                        itemValues,
                        transactionID,
                        out dwCancelID,
                        out pErrors);

//check if an error occured
Marshal.Copy(pErrors, errors, 0, numItems);

for(int xCount = 0; xCount < numItems; xCount++)
{
    // Data has been received
    if(errors[xCount] != 0)
    {
        // Errors occured - raise Exception
        m_OPCServer.GetErrorString(errors[xCount], m_LocaleID, out ErrorStr);
        System.Diagnostics.Debug.WriteLine(ErrorStr);
        throw new Exception(ErrorStr);
    }
}
// Free allocated COM-ressouces
Marshal.FreeCoTaskMem(pErrors);
```

Calling the write method of the OPC-AsyncIO2-Interface.

Error handling

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

### Displaying the values

Due to the fact that it is a synchronous call, only the “OnWriteComplete“ call-back method can be used to check whether the write job has been successful.

Due to the fact that all groups have a call-back method, the groups and client handles must now be differentiated in order to be able to assign the objects in the OPC client (see chapter4.5.2 / 4.5.3). The following code fragments show the procedure:

Figure 7-11

```
if (hGroup == CLNT_HNDL_GRP_CONVSETTINGS)
{
    // First: check for errors
    // just throw an exception if an error occured
    if(hrMastererr != 0)
    {
        // throw the first error
        m_OPCServer.GetErrorString(pErrors[0], m_LocaleID, out errorStr);
        throw new Exception(errorStr);
    }

    OPCItemExtender itm;
    for(int xCount = 0; xCount < dwCount; xCount++)
    {
        itm = (OPCItemExtender)(m_itmExtConvSettings[pClienthandles[xCount]]);
        if(dwTransid == m_TransIDSwitchPos)
        {
            // the switch position in the plc is not delivered via a parameter;
            // so the opc-client has to manage it;
            // here this is done with m_SentSwitchPos;
            // the switch position should only be adjusted, if the writing was
            // successful.
            itm.actValue = m_SentSwitchPos.ToString();
        }
    }
}
```

First the error handling.

Write value into conveyor control.

**Note**

For single-threaded OPC clients the following applies:

In order to keep the OPC client operable during read / write operations the asynchronous method must be used!
Synchronous calls block the OPC client which often causes the user to not receive any feedback from the OPC client and hence might suggest to him that the OPC client has crashed.

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

## 7.4 Release of references on the OPC server

This section now shows the way in which the COM references are released.

### General information

In .NET it is common to use the IDisposable pattern. As shown in chapter 4.2, the respective class implements the IDisposable interface. This provides the Dispose(void) method.

Calling the Dispose(void) method executes the Dispose(true) method, which releases the references to be released manually.
If this method is not called (programming error), a finalizer (destructor), which calls the same dispose(false) method.

Result: the COM references are being released again in any case.

**Note** The call of the finalizer is not deterministic. This means, the time interval between deleting the last reference to an object and deleting the object from the memory is not defined.

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

### Necessity of this pattern

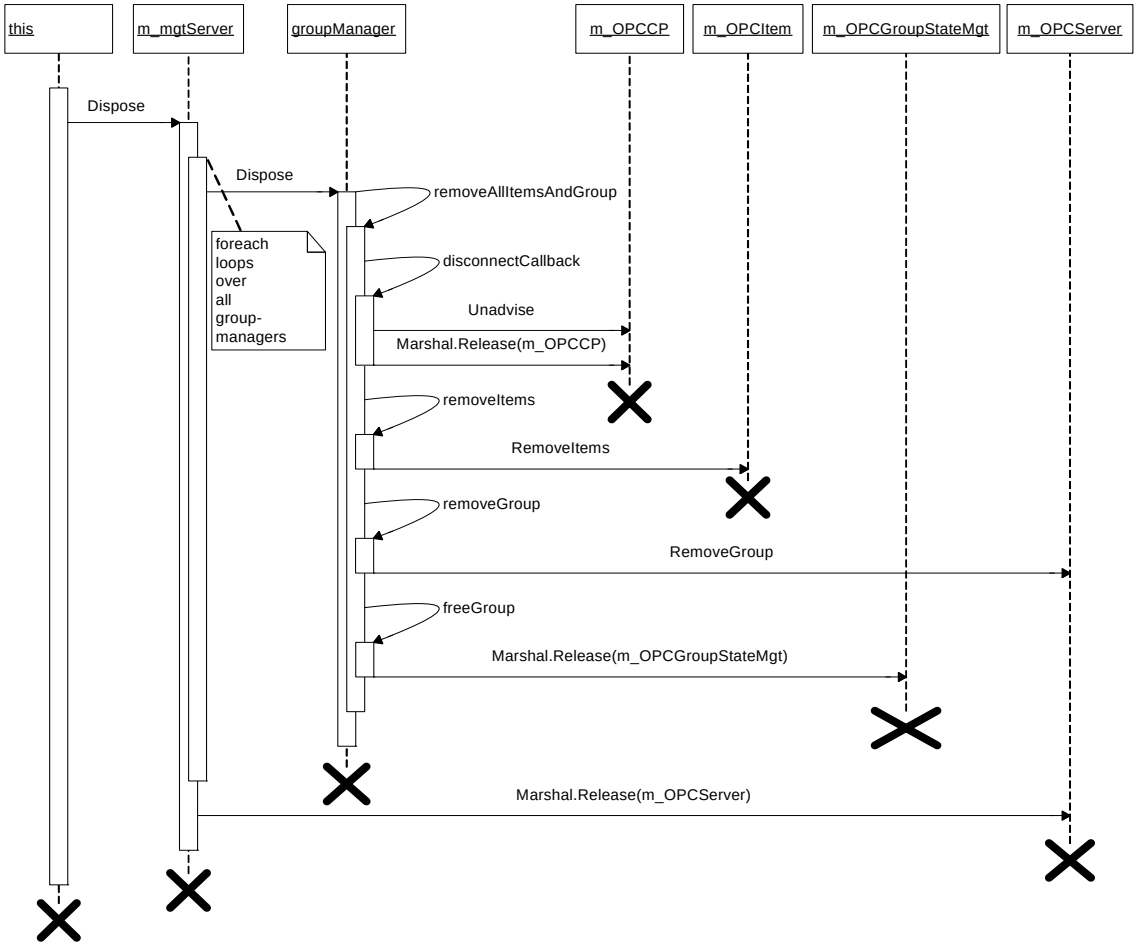
The necessity for this procedure becomes apparent when considering that the COM objects cannot be recorded by the GarbageCollector. They are available in the unmanaged code area.

An additional factor are the dependencies between the COM objects. The COM object of the Typs IOPCServer type, for example, must not be released as long as COM objects of the IOPCGroupStateMgt type are still referenced. Otherwise the IOPCGroupStateMgt objects could not be released anymore (the IOPCServer object required for this does then not exist anymore).

### Implementation in this application

The following sequence diagram shows the processes when closing the OPC client:

Figure 7-12




Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

## 7.5 Internationalization

### Introduction

Here you will learn about the steps performed in this application for generating a multi-language user interface.

The Code sections have been taken from the “OPCClientForm” class.

### Multi-language feature which cannot be swapped during runtime

For the application to support multi-language the following steps must be performed:

Table 7-4

| No. | Instructions | Note |
|---|---|---|
| 1 | During the layout design you select the internationalization. |  |
| 2 | Generate your user interface for the supported languages. | |
| 3 | Generate the additional resource files which contain the resources (strings, icons, cursors etc.) **not** contained in the respective user interfaces. | • Due to the resource files generated by VS.NET having to be re-generated at each user interface adjustment, and hence the resources introduced by the user having to be deleted, the extra resource files should be used.<br>• The procedure for generating and assigning to individual objects is described in **MSDN2003** |
| 4 | Please include the following lines into the constructor of your user interface:<br><br>Allocate the resource file you have generated to a resource manager object. | // The start up language is english<br>System.Threading.Thread.CurrentThread.CurrentUICulture = System.Globalization.CultureInfo.CreateSpecificCulture(“en”);<br><br>m_ResMan = new System.Resources.ResourceManager (« Complex_Client.Complex-Client_res_en », this.GetType().Assembly); |

**Note**

- Please save the language into a configuration file or into the registry in order to poll it when restarting.
- Generate the layout in the ”standard“ language.
- Only switch to the respective language when changing the language specific controls. This saves development time.

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

### Multi-language feature which can be swapped during runtime

In order to provide multi-language during runtime additional steps need to be performed:

Table 7-5

| No. | Instructions |
|---|---|
| 1 | Implementing a method for reinitializing the control elements of the user interface:<br>In order to switch all control elements of this application a recursive method has been implemented (it may however differ from another application): |

```
/// <summary>Reinitializes the resources, such as name and controlwidth</summary>
/// <param name="resMan">the ResourceManager instance</param>
/// <param name="curControls">the first control container of the form</param>
private void ReInitLanguage(System.Resources.ResourceManager resMan,
                            Control.ControlCollection curControls)
{
    foreach(Control cont in curControls)
    {
        // Do a recurse call in order to get all controls!
        // Beware that the ConveyorGraph control must be updated outside this function,
        // because it has its own resource file
        if (cont.GetType().ToString() !=
            "Siemens.CS2.Samples.OPC.Complex_Client.ConveyorGraph.ConveyorGraphControl"
            && cont.Controls.Count > 0)
            ReInitLanguage(resMan,cont.Controls);

        cont.Text = resMan.GetString(cont.Name + ".Text");
        cont.Size = ((System.Drawing.Size)(resMan.GetObject(cont.Name + ".Size")));
        cont.Location = ((System.Drawing.Point)(resMan.GetObject(cont.Name + ".Location")));
    }
}
```

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

2 Switching the references to the manually generated resources:
In order to do this, the “CultureID“ has been switched and another resource DLL loaded.

```
/// <summary>The menu handlers do control the language of the user interface</summary>
#region Menu handler

private void EngMenuItem_Click(object sender, System.EventArgs e)
{
    // Switch language to english
    GerMenuItem.Checked = false;
    EngMenuItem.Checked = true;

    // Set the thread UICulture
    System.Threading.Thread.CurrentThread.CurrentUICulture =
        System.Globalization.CultureInfo.CreateSpecificCulture("en");

    // Set the resources for the strings
    m_ResMan.ReleaseAllResources();
    m_ResMan =
        new System.Resources.ResourceManager("Complex_Client.Complex-Client_res_en",
                                              this.GetType().Assembly);

    // Set the resources of the thread UICulture
    System.Resources.ResourceManager res =
        new System.Resources.ResourceManager(typeof(OPCClientForm));

    this.Text = res.GetString("$this.Text");
    TopLangMainMenu.Text = res.GetString("TopLangMainMenu.Text");

    // Now set the new language for the conveyor graph
    ConveyorGraphControl.SetLang();

    // Reinit the speeds dialog, if it is open
    if (m_SpeedsDlg != null)
        m_SpeedsDlg.SetLanguage();

    // reinit the language for all other controls in this application
    ReInitLanguage(res,this.Controls);

}
```

Switch language of application

Load own resource ID

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

# 8 Brief Introduction into Using .NET

**Here you will find information on …**

how to include the SIMATIC NET OPC-DA interfaces, .NET and AcitveX controls into a .NET application.

**Note** This chapter does not provide an operation instruction for Studio .NET nor and introduction into .NET-Framework. Only the main tricks for including the above components are listed.

**Discussed topics**


| Chap. | Title | Page |
|---|---|---|
| 8.1 | Including the OPC-DA-RCW interfaces | 90 |
| 8.2 | Installing a.NET-UserControl | 93 |
| 8.3 | Installing an ActiveX -UserControls | 95 |


Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

## 8.1 Including the OPC-DA-RCW interfaces

**Prerequisites**

The following prerequisites must be fulfilled:

- SIMATIC NET V6.2 has been installed
- Visual Studio .NET has been started with a .NET windows application

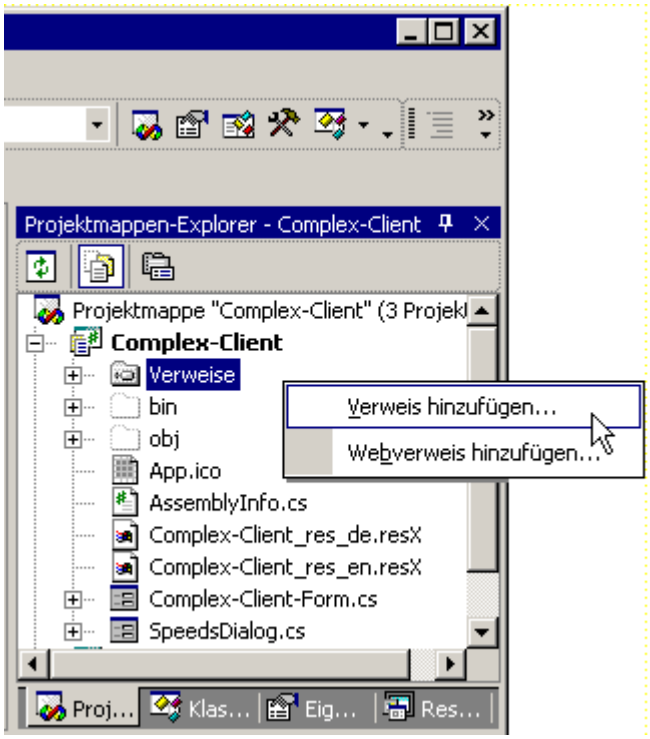
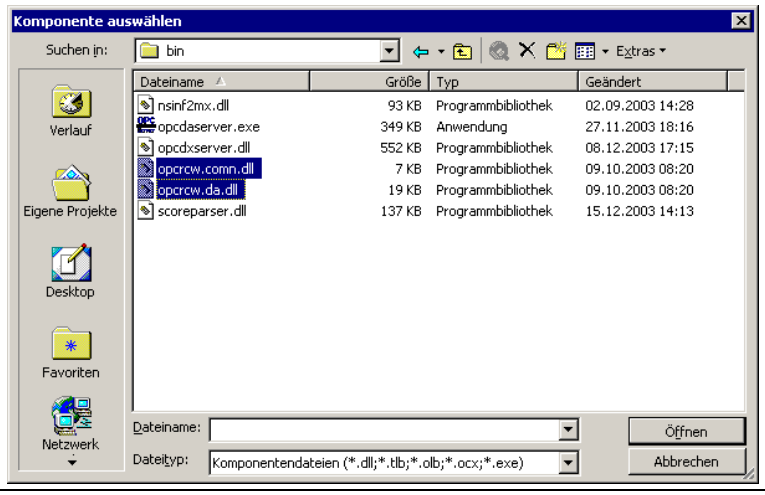
**Including the .NET-RCW (Runtime Callable Wrapper)**

In order to access the OPC-DA interface, you must generate a reference to the OPC-DA-RC wrapper within the project.

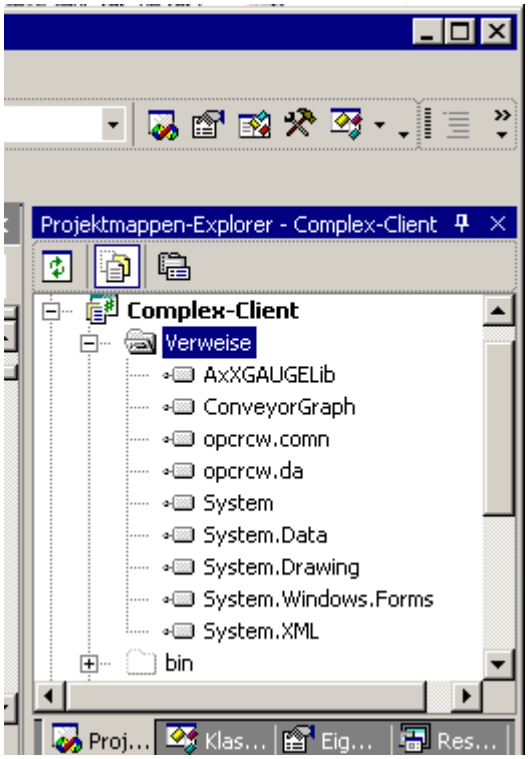
Chapter 7 describes how the respective OPC-DA interfaces must be used.

**Note** The necessity for an RCW and its explanation is given in chapter 3.5.3.

Table 8-1

| No. | Instructions | Note |
|---|---|---|
| 1 | Click `Add reference...` within the project. |  |
| 2 | Click `Search` in the following dialog and select the files `opcrcw.comn.dll` and `opcrcw.da.dll`, which should be located at [SIMATIC-NET-installation path]`\opc2\bin\`.<br><br>Confirm with `Open` and then with `OK`. |  |

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

| No. | Instructions | Note |
|---|---|---|
| 3 | The RCWs now appear under the directories. |  |



Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

## 8.2 Installing a.NET-UserControl

### Prerequisites

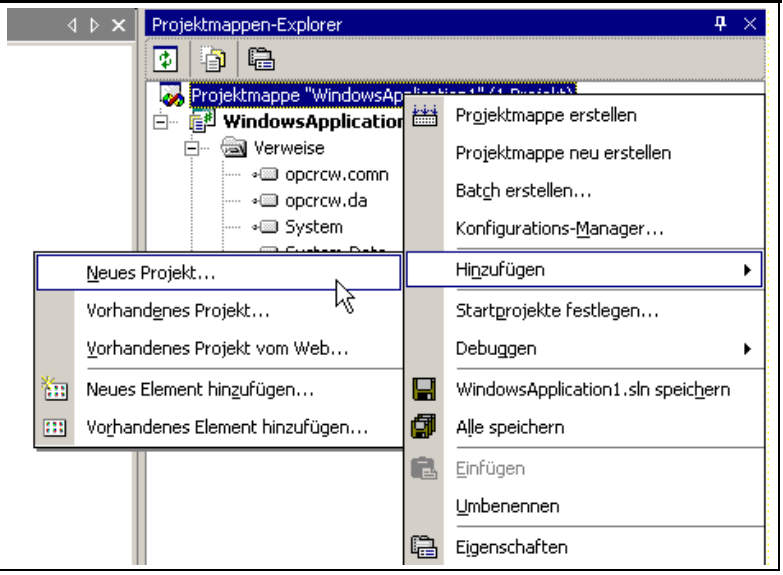
The following prerequisites must be fulfilled:

- Visual Studio .NET has been started with a .NET windows application
- The .NET-Usercontrol has been generated in the same project as the main application.

**Note** If the .NET-Usercontrol is not within the same project, you only need to insert a reference to the control (analog to chapter 8.1)

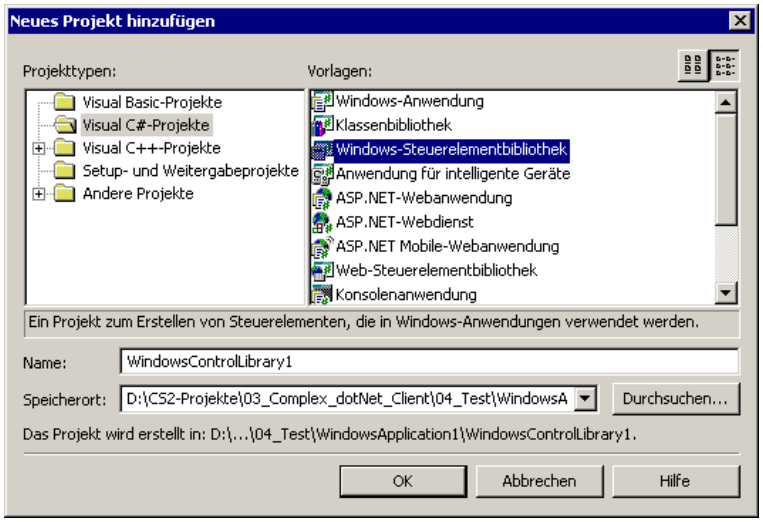
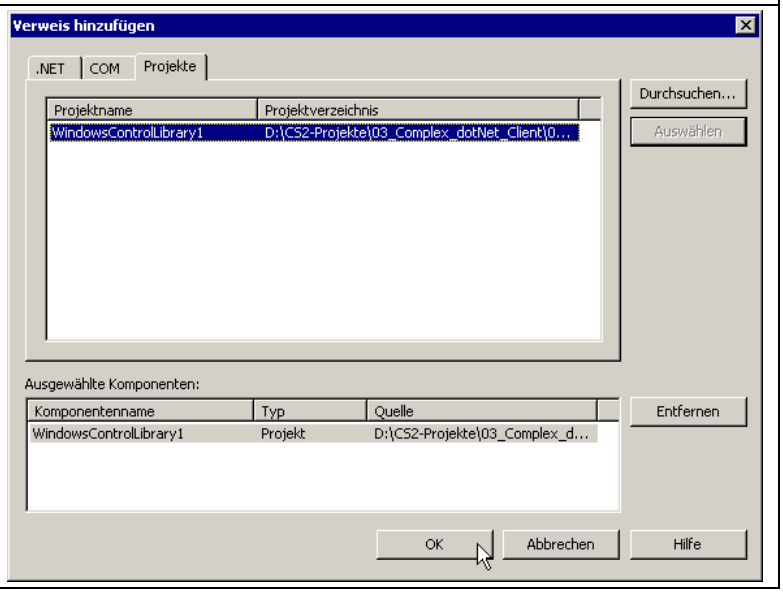
### Installing a.NET-UserControl

Table 8-2

| No. | Instructions | Note |
|---|---|---|
| 1 | Enter a new project into the existing project. |  |



Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

| No. | Instructions | Note |
|---|---|---|
| 2 | Please select a `Windows Control element library`.<br><br>Confirm with `OK`. |  |
| 3 | Implement the control. | |
| 4 | Enter a reference to the control into the main application. (chapter 8.1<br>Please select the `Projects` tab and confirm `OK`.<br><br>Now you can use this control via the ToolBox in the Visual Studio .NET. |  |

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

## 8.3 Installing an ActiveX -UserControls

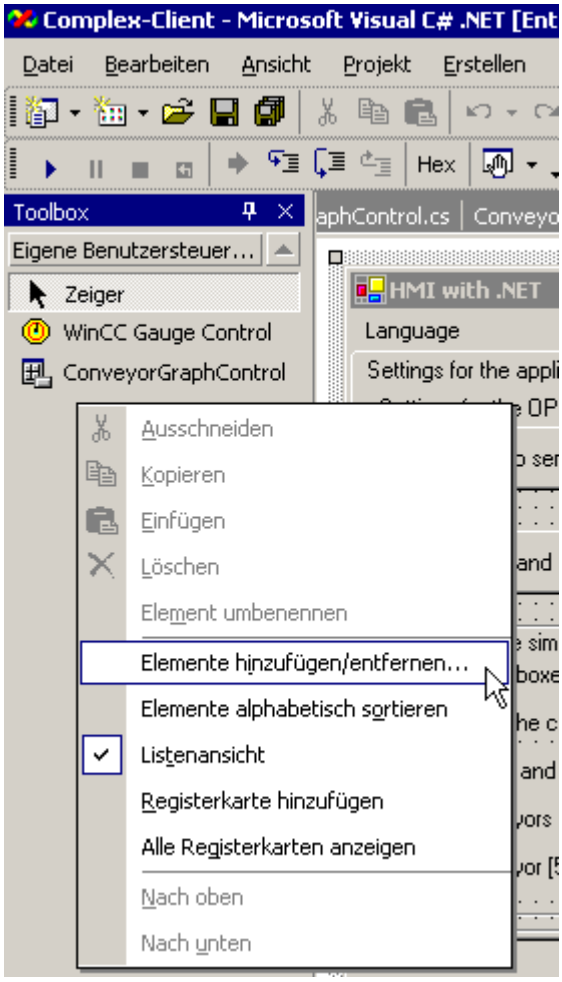
### Prerequisites

The following prerequisites must be fulfilled:

- Visual Studio .NET has been started with a .NET windows application

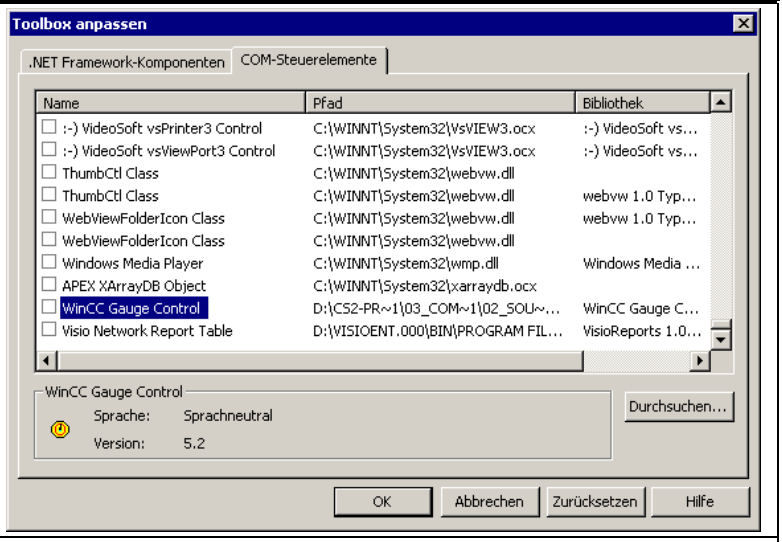
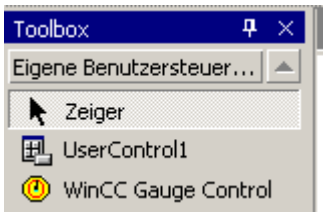
### Installing the ActiveX -UserControls

Table 8-3

| No. | Instructions | Note |
|---|---|---|
| 1 | Add a reference to the ActiveX-UserControl by adding it to the ToolBox of Visual Studio .NET. |  |



Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

| No. | Instructions | Note |
|---|---|---|
| 2 | Select the `COM Control elements` tab and select the respective component.<br><br>Confirm with `OK`. |  |
| 3 | The control now appears in the ToolBox. Now it can be integrated into the user interface of the main application via drag&drop. | <br><br>**Note:**<br>The Visual Studio .NET automatically generates an RCW. |

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

# Structure, Configuration and Operation of the Application

**Content**

This part leads you step by step through the structure, important configuration steps, commissioning and operation of the application.

# 9 Installation and Commissioning

**Here you will find information on …**

the hardware and software to be installed and which steps are necessary for commissioning the example.

**Discussed topics**


| Chap. | Title | Page |
|---|---|---|
| 9.1 | Installation of Hardware and Software | 97 |
| 9.2 | Installation of the application software | 98 |


Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

## 9.1 Installation of Hardware and Software

This chapter describes the hardware and software components to be installed. The description and manuals as well as delivery information contained in the delivery scope of the respective products, should be followed in any case.

**Installation of the hardware**

The hardware components are available in chapter 2.3. For the hardware setup, proceed according to the table below:



**Warning**

**Switch on the power supply after the last step only .**

Table 9-1

| No. | Focus | Instructions |
|---|---|---|
| 1 | Controller – S7-300 station | Install the station in compliance with the diagram shown in chapter 2.1). |
| 2 | Industrial Ethernet | Connect the controller with the PG as illustrated in chapter 2.1.<br>**Note** Instead of using a hub or a switch, a cross cable may also be used. |

**Note** Setup guidelines for SIMATIC S7 must generally be followed.

### Installation of the standard software

STEP 7 and SIMATIC NET must be installed on the PG/PC. On SIMATIC PGs STEP 7 is already installed.

At this point, we will not go further into the installation of STEP7 and SIMATIC NET. The installation occurs in familiar Windows environment and is self-explanatory as well as described in the respective manuals.

## 9.2 Installation of the application software

The application software is delivered with a setup program.



**Attention**

**If using an older operating system than Windows XP SP1, you must first install .NET-Framework 1.1.**

**Information on this is available on the Microsoft internet pages**
http://www.microsoft.com/

For installing the user interface proceed as follows.

Table 9-2

| No. | Instructions | Note |
|---|---|---|
| 1 | Unzip the 21043779_C#_OPC_Client_SETUP_v10.zip file. | |
| 2 | Double-click on the **Setup.EXE** file. | |
| 3 | The installation assistant now leads you through the individual installation steps. | Install the standard package. |

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

### Installed files

The installation routine now installs the following files into the given folder:

Table 9-3

| File | Belongs to … |
|---|---|
| AxInterop.XGAUGELib.dll | Imported ActiveX-Control |
| Interop.XGAUGELib.dll | Imported ActiveX-Control |
| CCGauge.ocx | Binary of the ActiveX Control |
| Stdole.ocx | Runtime library required from ActiveX Control |
| Complex-Client.exe | Main application |
| OpcCRcw.Comn.dll | OPC-RC Wrapper |
| OpcRcw.Da.dll | OPC-RC-Wrapper |
| ConveyorGraph.dll | Conveyor belt control |
| Complex-Client.resources.dll | Resource file for the main application; each in the sub-folder “en” and “de”. |
| ConveyorGraph.resources.dll | Resource file for the conveyor control; each in the sub-folder “en” and “de”. |
| opcConnector.dll | Contains the Interface IOPCItemControl |

### Created program group

The setup creates a program group which can be reached via the Windows start menu:

Programs → CS2 Samples → Complex Client .NET

### Uninstalling

Uninstalling via: Control Panel→ Software.

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

# 10 Configuration Process

**Here you will find information on …**

the steps necessary for configuring your PG/PC (PC-station) for using the application.

You will also learn about the steps necessary for configuring an OPC server in STEP 7 project (this has already happened in the delivered STEP 7 project).

**Discussed topics**


| Chap. | Title | Page |
|---|---|---|
| 10.1 | Configuring the PC station | 100 |
| 10.2 | Configuration of the OPC server | 104 |


Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

## 10.1 Configuring the PC station

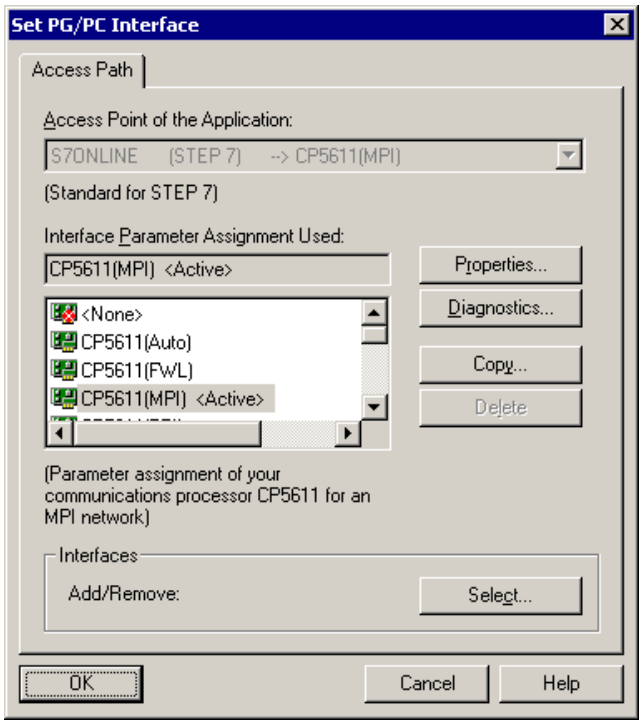
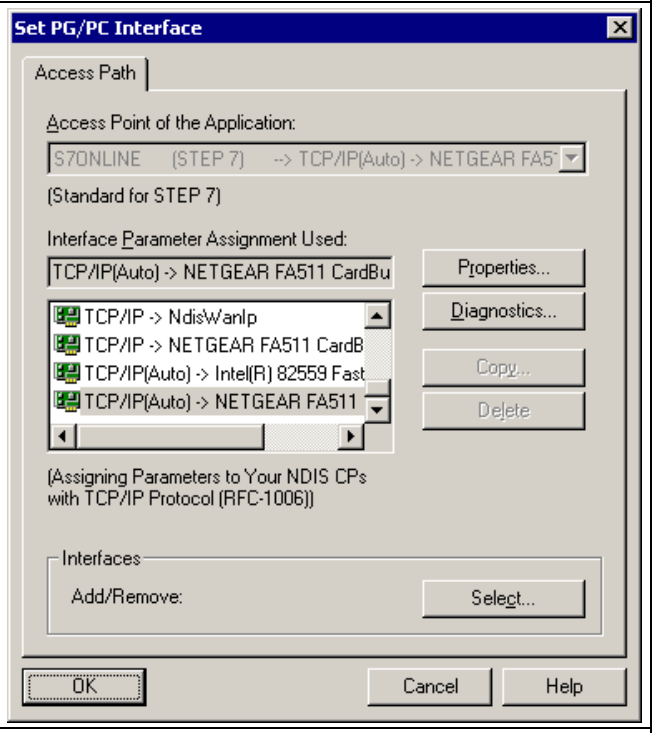
**Set PG/PC interface**

The Ethernet network card must be set as PG/PC interface.

**Note** Please ensure that the network card has the default IP address 140.80.0.2 (it can be set via the network settings and the TCP/IP properties).

Please proceed as follows:

Table 10-1

| No. | Instructions | Note |
|---|---|---|
| 1 | Start the SIMATIC manager | |
| 2 | Open the dialog for setting the PG/PC interface via `Extras > Set PG/PC interface`. |  |
| 3 | Set your network card as depicted.<br><br>**Note:**<br>In this case a NETGEAR network card has been used. |  |
| 4 | Confirm with `OK`. | |





Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

### Address assignment for the involved modules

If you wish to operate the project at an existing Industrial Ethernet, you must follow the address assignment below:

Table 10-2

| Focus | Module | IP address |
|---|---|---|
| Controller | CP 343-1 | 140.80.0.1 |
| PG | NDIS network card | 140.80.0.2 |

**Note**

- The correct subnet mask is 255.255.0.0.
- Alternatively it is also possible to change the IP addresses assigned in the STEP 7 project.

### Installation of the STEP 7 project

Proceed as follows:

Table 10-3

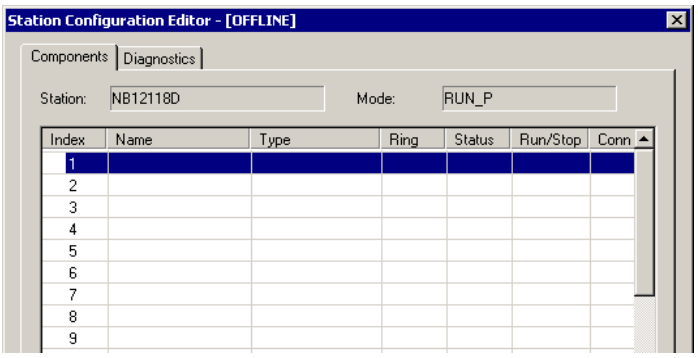
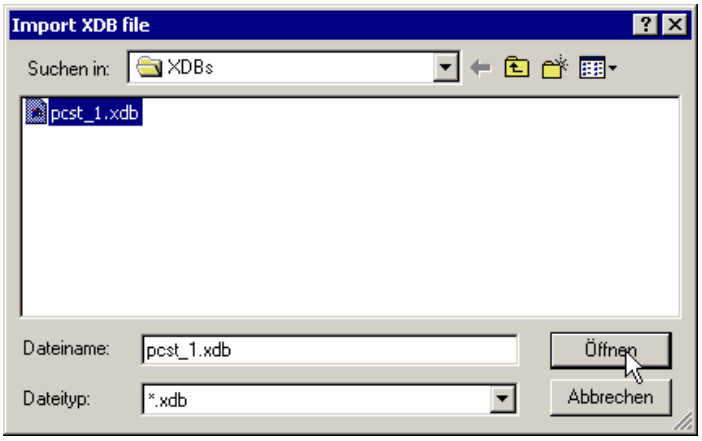
| No. | Instructions | Note |
|---|---|---|
| 1 | Open the SIMATIC Manager | |
| 2 | Extract the project via the `File > Extract` menu | Use the browser to search the 21043779_C#_OPC_Client_STEP7_v10.zip project and click OK to confirm..<br>**Note** If SCL has not been installed a note appears. However, commissioning is still possible. |
| 3 | Select a directory into which the project files are to be retrieved and open the project. | |
| 4 | In the project tree, click `SIMATIC 300` and load the program into `SIMATIC 300` with `PLC -> Download`. | **Note** Please ensure that the MMC of the CPU has been deleted prior to loading. |
| 5 | Please set the CPU 315-2 DP into ”RUN“ mode. | |

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

## Importing the XDP file into the component configurator

**Note** The SDB file already exists in the delivered STEP7 project

Table 10-4

| No. | Instructions | Note |
|---|---|---|
| 1 | Open the `Components configurator` by double-clicking on the  icon (in the toolbar next to the clock) |  |
| 2 | Click on `Import Station` | **Note** Confirm the query with `Yes` |
| 3 | Browse to the project folder of your STEP7 project. There you open the subfolder `XDBs` and select the XDB file.<br>Press `open`. |  |
| 4 | The import wizard now confirms that the import is possible.<br>Press `OK`. | |

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

## 10.2 Configuration of the OPC server

The configuration made within this application.

The main focus of this application is using OPC. Here a number of configurations have become necessary.

There are several options for configuring the OPC server.
Here the import of the XDB file delivered by STEP7 is illustrated.

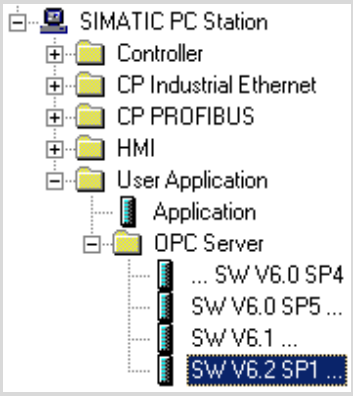
**Note** You must first set the PG/PC interface according to chapter 10.1.

**Note** The configurations need not be performed in the delivered STEP7 project.

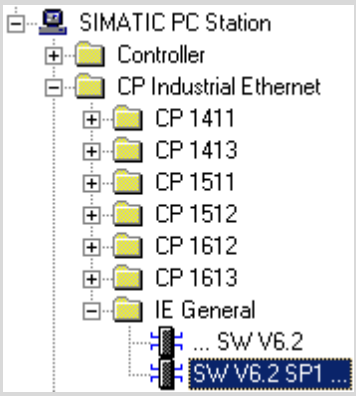
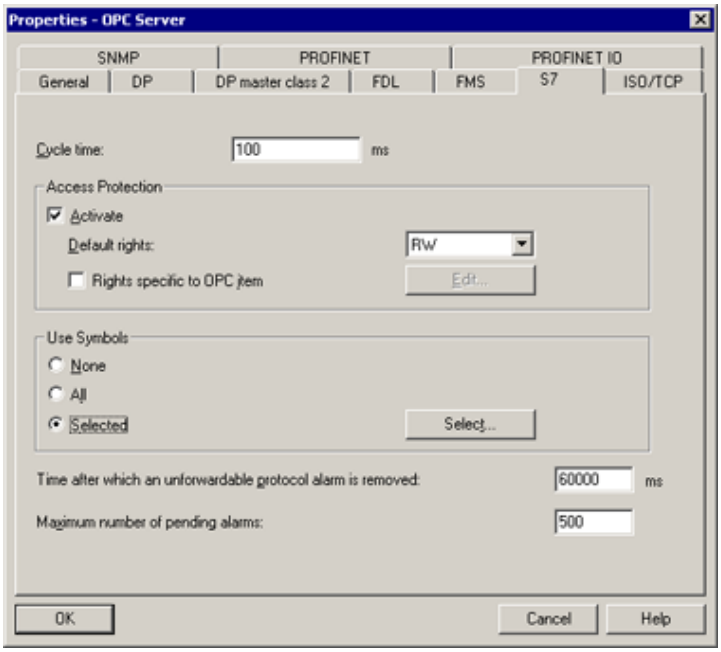
### Configuring the OPC server in HW-Config

Please perform the following steps in order to add an OPC server to an existing STEP7 project with already configured SIMATIC station.

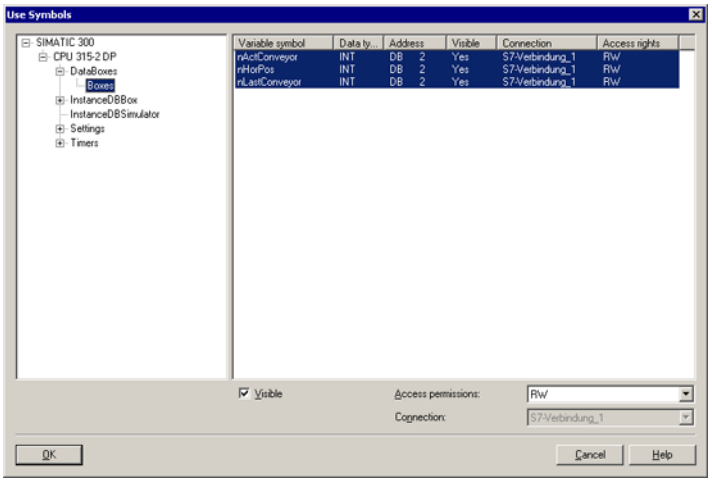
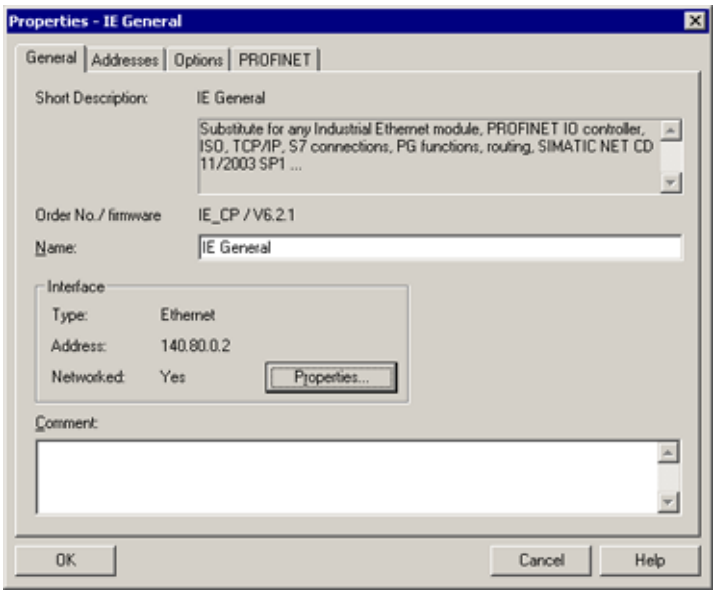
Table 10-5

| No. | Instructions | Note |
|---|---|---|
| 1 | Create a new station of the “SIMATIC PC Station“ type in STEP 7. | |
| 2 | Open the created station by double-clicking `Configuration` | The hardware configuration opens. |
| 3 | Add a SIMATIC NET OPC server to slot 1. | The SIMATIC NET OPC-Server V6.2.1 was used in this application.<br>**Note** It is located in the calalog directory:<br> |

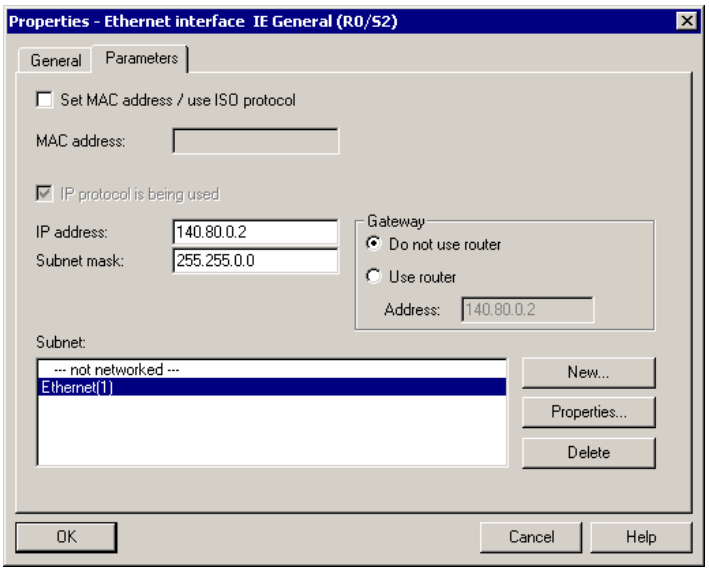
Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

| No. | Instructions | Note |
|---|---|---|
| | • Add an `IE General` → `SW V6.2 SP1`.<br>**Note:**<br>Please note the `visible` check box in the dialog. | In this application a standard Ethernet network card has been used<br>**Note** It is located in the calalog directory:<br> |
| 4 | Open the Properties – OPC server window with a double-click and select the S7 tab.<br>Here the settings for the communication with the S7 protocol are made. |  |
| 5 | Activate the access rights as well as using symbols (see step 5 of the figure) | |

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

| No. | Instructions | Note |
|---|---|---|
| 6 | If you wish to only symbolically access a selection of process variables, please select `Use symbols` → `Selected`<br>Use the `Select` button to select the symbolically addressable variables.<br>**Note:**<br>• Please note the `visible` check box in the dialog.<br>• For this application you select all symbols |  |
| 7 | Close the dialog and the object properties of the OPC server with `OK`. | |
| 8 | Open the object properties of the network card with double-click. |  |
| 9 | Click on `Properties` | |

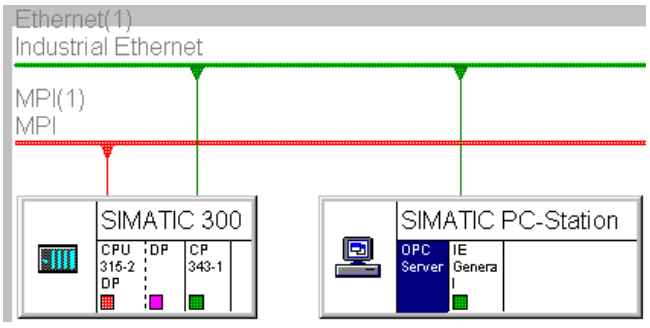
Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

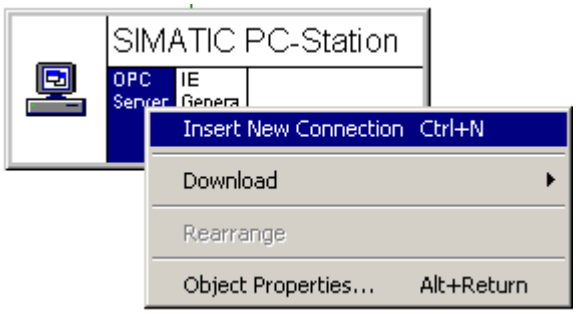
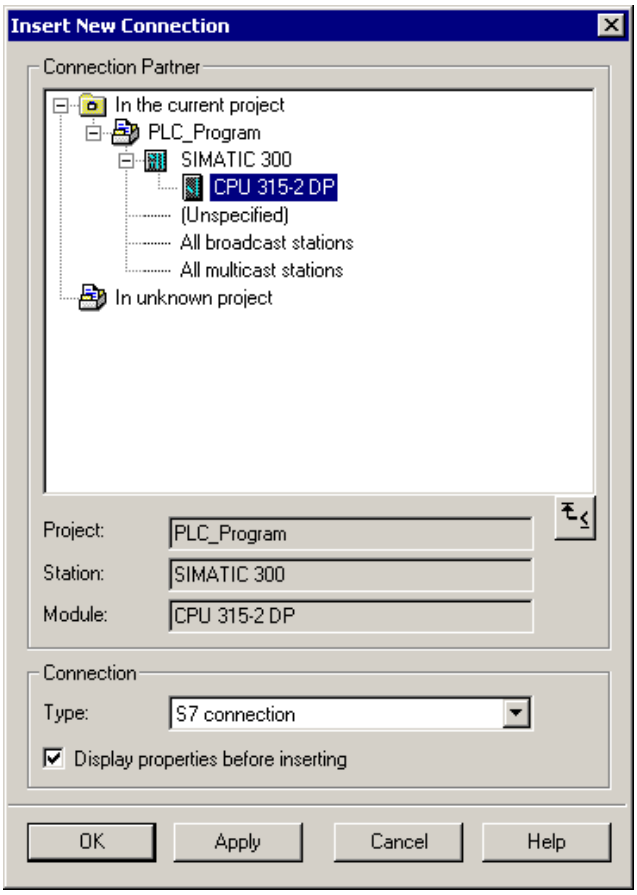
| No. | Instructions | Note |
|---|---|---|
| 10 | Adjust the parameters as depicted on the right.<br><br>**Note:**<br>If no IE-network has been configured, create another one with `New` and accept the standard parameters. |  |
| 11 | Save and compile the hardware configuration with .<br><br>Close the hardware configuration. | |

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

## Configuring the OPC server in NetPro

Table 10-6

| No. | Instructions | Note |
|---|---|---|
| 1 | Open NetPro from the SIMATIC Manager via the  button. | |
| 2 | Select the OPC server |  |

| No. | Instructions | Note |
|---|---|---|
| 3 | Create a new S7 connection |  |
| 4 | Select the CPU 315-2DP as a connection partner. Select an S7 connection as connection type. |  |
| 5 | Accept the standard parameters with `Apply` and `OK`. | **Note:**<br>Please note the name of the connection. It will be necessary in the application. |
| 6 | The following image appears in the connection table |  |
| 7 | Save and compile the configuration with . | |
| 8 | Close NetPro. | |

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

**Note** The created XDB file must now be loaded into the Station Manager via import.

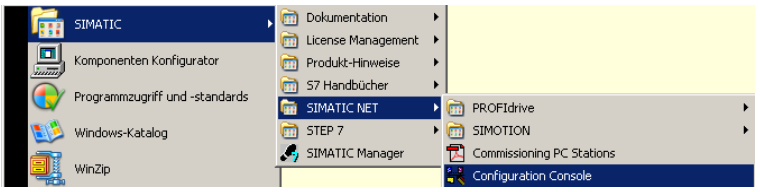
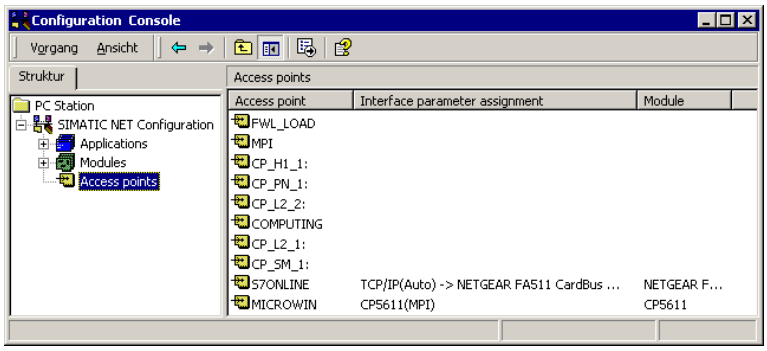
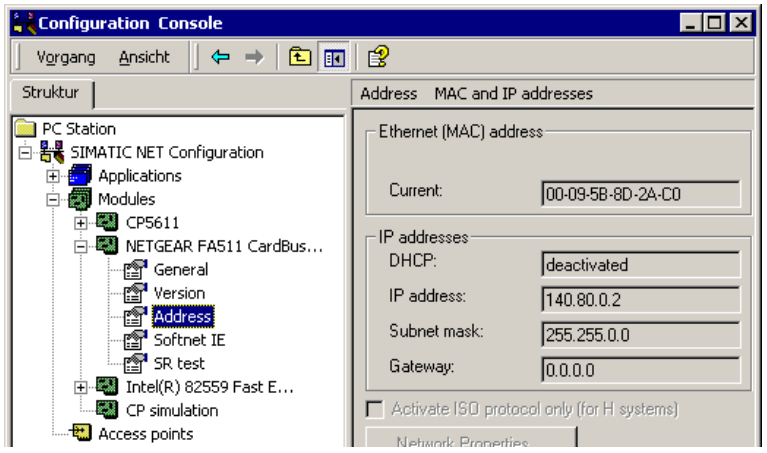
Proceed as described in chapter 10.1.

## Controlling the settings

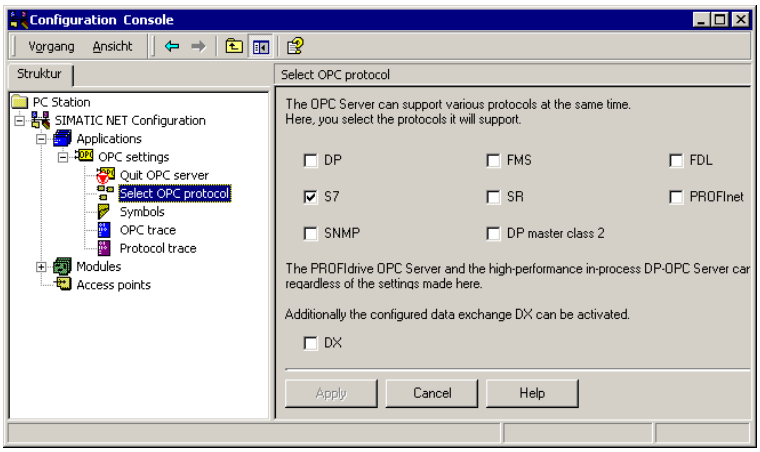
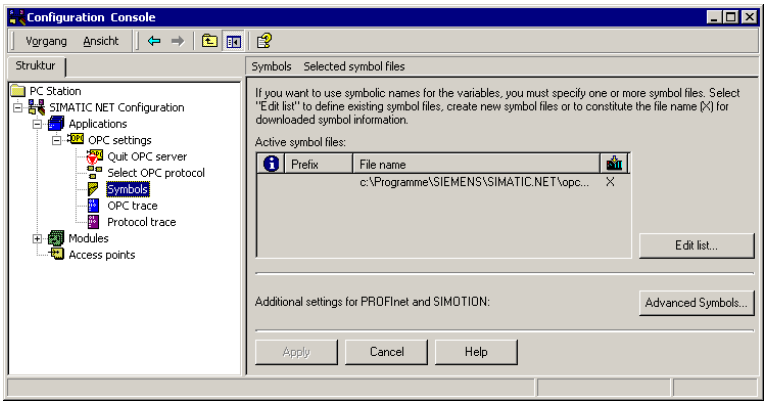
The settings can be controlled with the configuration console ”Set PC Station“.

Thereby the following settings should be considered:

Table 10-7

| No. | Instructions | Note |
|---|---|---|
| 1 | Open the configuration dialog |  |
| 2 | Control the set access point for `S7ONLINE`. |  |
| 3 | Control the set IP address |  |

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

| No. | Instructions | Note |
|---|---|---|
| 4 | Control the set protocols<br><br>**Note:**<br>For this application activating the S7 protocol is sufficient. |  |
| 5 | Check whether the symbols have been loaded. |  |
| 6 | If one of the settings should not correspond to the displayed images, please perform the previous configuration again.<br><br>Close the configuration dialog. | |

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

# 11 Operating the Application

**Here you will find information on …**

how to operated all function of this application.

**Possible settings for the user interface**

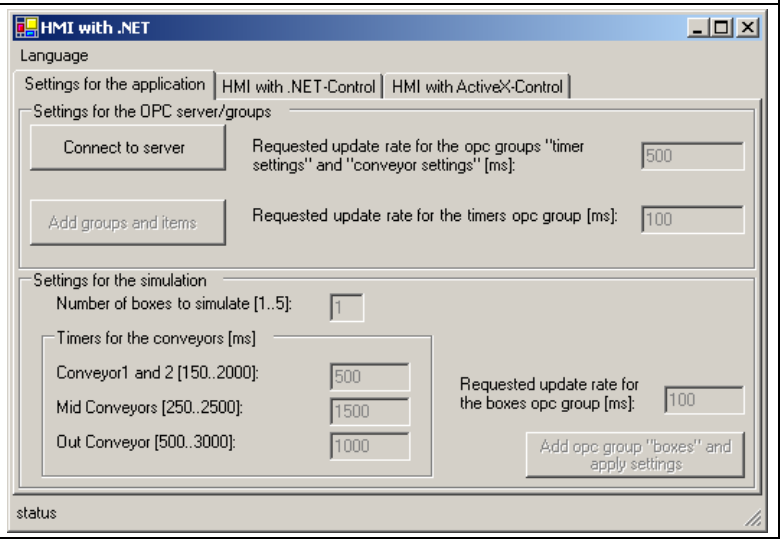
You can make the following settings:

- Update rate of the OPC server
- Number of boxes to be simulated
- Times according to which the positions of the boxes should be incremented by 1

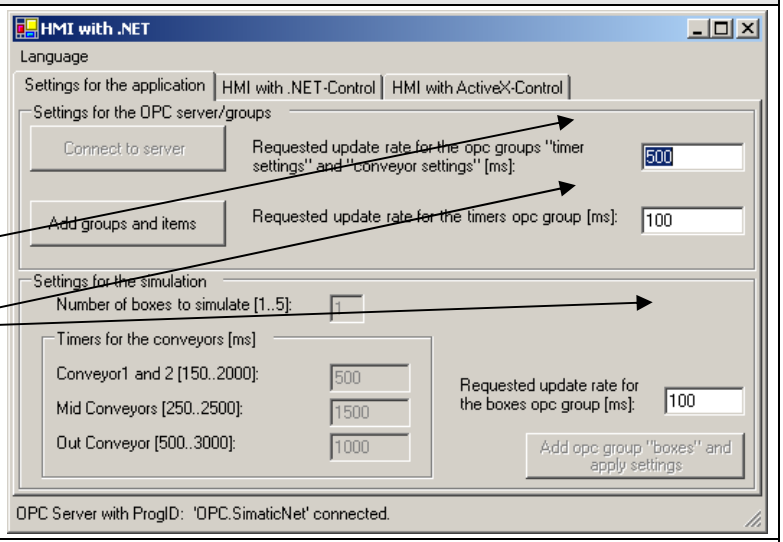
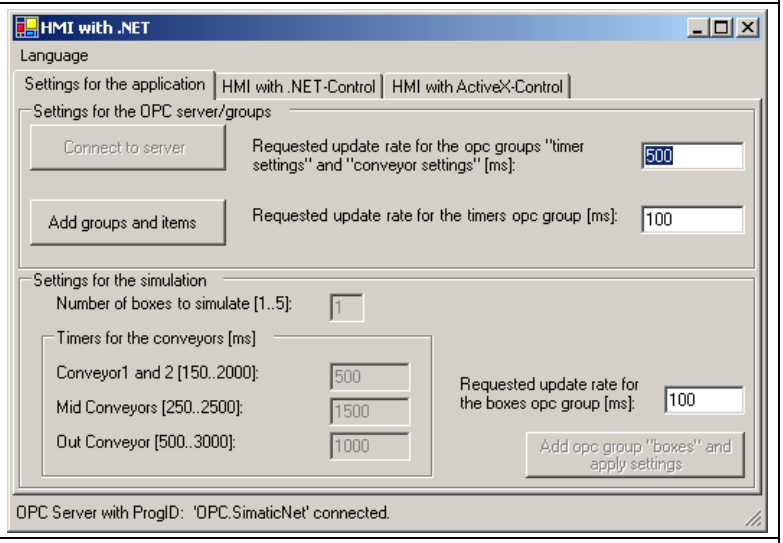
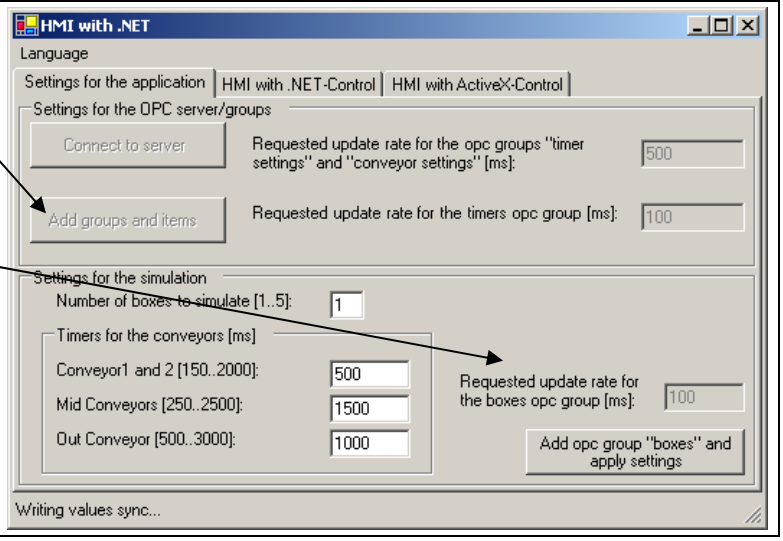
**Commissioning the simulation**

For commissioning and setting the simulation please proceed as follows:

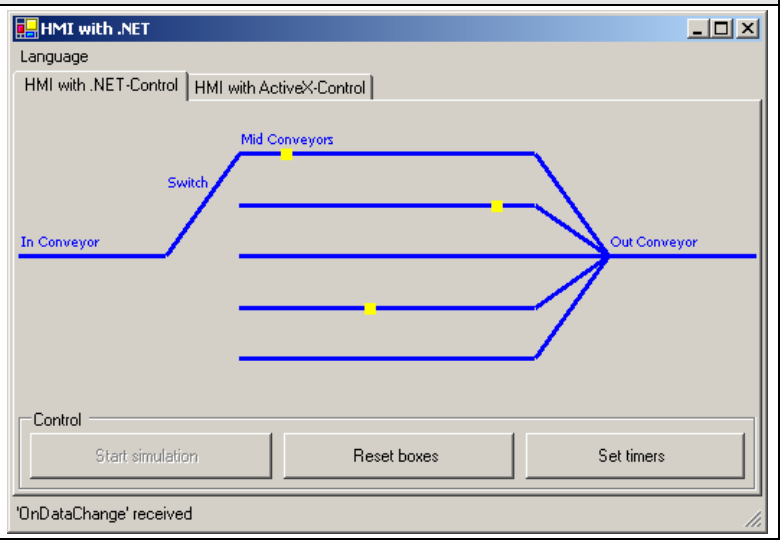
Table 11-1

| No. | Instructions | Note |
|---|---|---|
| 1 | Start the user interface via : `Start` → `Programme` → `CS2-Samples` → `Complex-Client-.NET` | |
| 2 | Connect to the OPC server:<br><br>Press the `Connect with server` button |  |

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

| No. | Instructions | Note |
|---|---|---|
| 3 | Set the update rate of the individual groups:<br><br>Set the times in ms.<br><br>• Group “Settings“<br>• Group “Times“ (conveyor speed)<br>• Group “Boxes“ |  |
| 4 | Create groups and items<br><br>Confirm with `Add groups and items.`<br><br>Now the process variables are introduced to the server by creating the OPC groups and OPC items for the settings. |  |
| 5 | Set simulation settings.<br>Accept the settings.<br>• Settings for the simulated belt velocities<br>• Add OPC group “Boxes“ and apply settings with synchronous write command |  |

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

| No. | Instructions | Note |
|---|---|---|
| 6 | Start simulation:<br><br>After pressing the `Apply` button please change to the `HMI with .NET-Control`.tab.<br><br>There you press `Start simulation`.<br>After approx. 4 seconds the first box appears on the left side of the conveyor. |  |

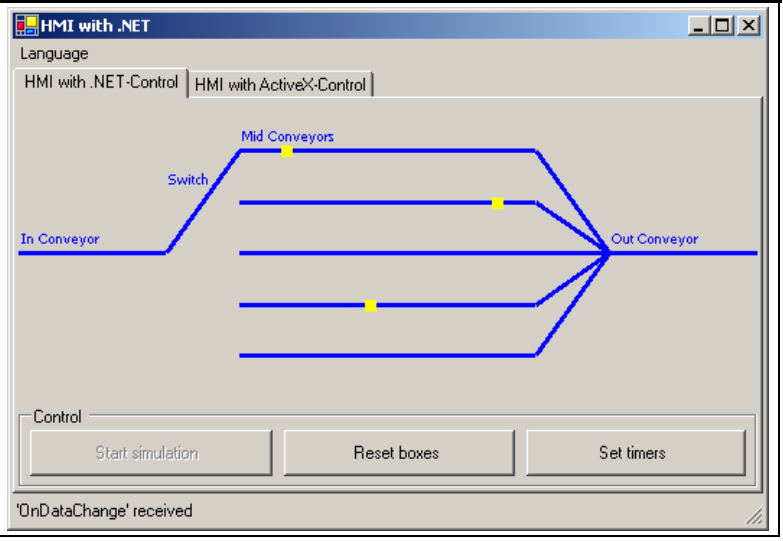
Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

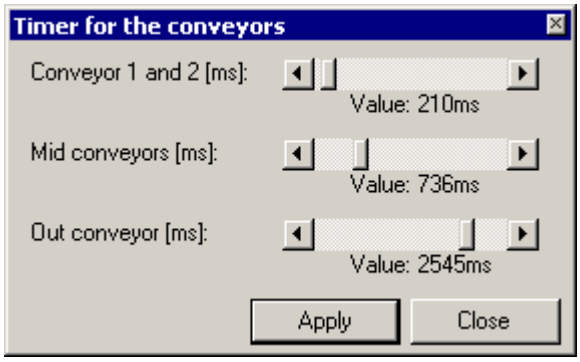
### Setting the conveyor velocities in runtime mode

The times simulating the conveyor speed can be changed during runtime

Please proceed as follows:

Table 11-2

| No. | Instructions | Note |
|---|---|---|
| 1 | Press the `Set Timers` button. |  |

| No. | Instructions | Note |
|---|---|---|
| 2 | Set the desired times and confirm with `Apply`.<br>**Note:**<br>The dialog can stay open. |  |

### Error case

In certain cases a side effect of the simulation may occur in the controller: It is possible that boxes overlap regarding their position.

This is due to the fact that the timer events of the controller can run simultaneously.

### Example

Every 10th time a continuously repeating timer with 100ms will run synchronous with a timer of 1000ms. This leads to the position of two boxes being incremented which at the interfaces of the respective conveyors may lead to overlapping of boxes.

### Work around

In `Reinitialize boxes` the boxes and timer can be readjusted.

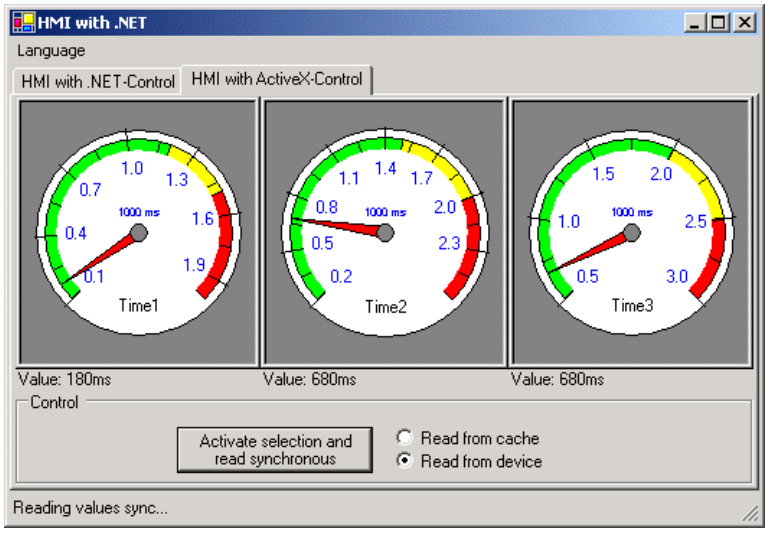
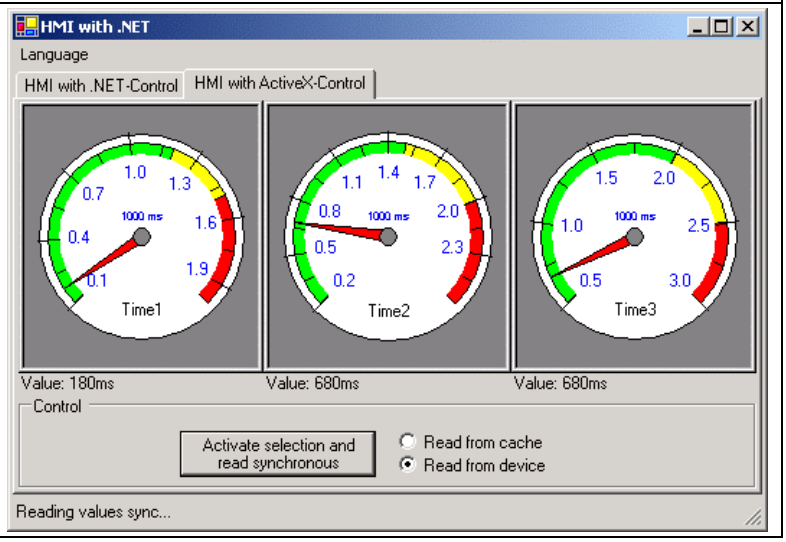
Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

### Display of timer status with the ActiveX-Controls

The current times of the simulation can be read with the ActiveX-Controls.

The following table describes the procedure:

Table 11-3

| No. | Instructions | Note |
|---|---|---|
| 1 | Change to the `HMI with ActiveX-Control` tab.<br>Note:<br>This deactivates the HMI functionality for the boxes (see chapter 7.3). This becomes apparent by the fact that the LEDs of the CPs are not blinking. |  |
| 2 | You can have the current status of times displayed with `Activate selection and read synchronous`. | This button executes a single synchronous read command. |
| 3 | With the radio buttons `Read from cache` and `Read from device` you can set the area for the read job. |  |

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

# Appendix and List of Further Literature

## 12 Glossary

**COM / DCOM**

COM: Component Object Model: Software model for communication between components based on a uniform interface; DCOM: Software model for communication beyond computer boundaries, based on COM.

**Event-Handler**

An event handler processes occurred events or Windows messages.

**Exception**

An exception refers to an exceptional situation.
This may be generated by the operating system (e.g. dividing by zero) or by the user program.

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

**Exception-Handler**

An exception handler prepares occurring exceptional situations. This is generally a save error behavior and/or a message to the user.

**HRESULT**

Return value data type of COM objects.

**IDL**

Interface Definition Language: A standardized language by Microsoft for definition of functioning and parameter interfaces.

**Polling**

English term which refers to (mostly cyclical) polling of certain values or statuses.

**Sink Interface**

A sink interface can be used for sending messages between components. The sink interface is based on COM mechanisms.

### Thread

"Threads" can be used for quasi parallel or simultaneous performing of several code fragments within an application or a process.

If an application uses several threads, the application has the property "multi-threaded".
If an application uses only one thread, it is referred to as "single-threaded" For this application all code fragments are always processed sequentially.

### Windows messages

In the usual operating systems by Microsoft Windows messages are exchanged for reporting events, e.g. the paint event.

### Wrapper

"Wrapper" generally refers to a class group which encapsulates other class groups for data conversion or simpler handling. It can be considered a "shell" encapsulating and covering the "wrapped" classes from the outside.

Copyright © Siemens AG 2005 All rights reserved
21043779_Csharp_OPC_Client_DOKU_v10_e.doc

# 13 Literature

### Bibliographic References

This list is by no means exhaustive and only gives a selection of appropriate sources.


| | Topic | Title |
|---|---|---|
| \1\ | OPC | • OPC DA 2.05 Specification bei http://www.opcfoundation.org/<br>• mn_opc_0.pdf und mn_opc_76.pdf von SIMATIC.NET V6.2 |
| \2\ | .NET | • Inside C#, Tom Archer<br>• .NET Crashkurs, Clemens Vasters, Oellers, Javidi, Jung, Freiberger, DePetrillo<br>• Microsoft .NET Framework Programmierung, Jeffrey Richter |