

# SIMOTION

## Frequently asked Questions

Description and example for the data exchange  
via OPC XML interface

**SIEMENS**

Data exchange via OPC XML

We reserve the right to make technical changes to this product.

## Copyright

Reproduction, transmission or use of this document or its contents is not permitted without express written authority. Offenders will be liable for damages. All rights, including rights generated by patent grant or registration or a utility model or design, are reserved.

Data exchange via OPC XML

## General Notes

### Note

The Application Examples are not binding and do not claim to be complete regarding the circuits shown, equipping and any eventuality. The Application Examples 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. 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.

### 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 Application Examples 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.

**Copyright© 2007 Siemens A&D.** It is not permissible to transfer or copy these standard applications or excerpts of them without first having prior authorization from Siemens A&D in writing.

For questions regarding this application please contact us at the following e-mail address:

[applications.erlf.aud@siemens.com](mailto:applications.erlf.aud@siemens.com)

Data exchange via OPC XML

## Qualified personnel

In the sense of this documentation qualified personnel are those who are knowledgeable and qualified to mount/install, commission, operate and service/maintain the products which are to be used. He or she must have the appropriate qualifications to carry-out these activities

e.g.:

- Trained and authorized to energize and de-energize, ground and tag circuits and equipment according to applicable safety standards.
- Trained or instructed according to the latest safety standards in the care and use of the appropriate safety equipment.
- Trained in rendering first aid.

There is no explicit warning information in this documentation. However, reference is made to warning information and instructions in the Operating Instructions for the particular product.

## Reference regarding export codes

AL: N

ECCN: N

---

Data exchange via OPC XML

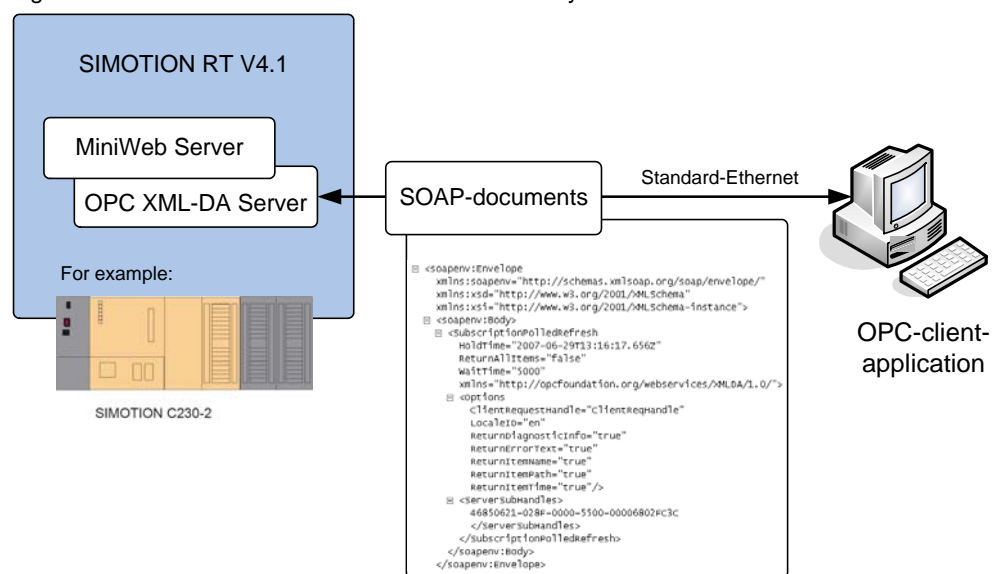
## Table of Contents

<b>1</b>	<b>How can you use the OPC XML-DA interface? .....</b>	<b>6</b>
<b>2</b>	<b>Voraussetzungen and targets of these FAQs .....</b>	<b>7</b>
2.1	What is OPC about? Why OPC XML-DA? .....	7
2.2	Besondere Eigenschaften der OPC XML-DA communication .....	8
<b>3</b>	<b>OPC-Client Software in Java .....</b>	<b>10</b>
3.1	Grafische Benutzeroberfläche des Java-Programms .....	10
3.2	Erstellung der Klassen zur OPC communication .....	12
3.3	Klassenbeschreibung der OPC Browser application .....	14
3.3.1	Klasse MainWindow (MainApplet) .....	15
3.3.2	Klasse PWDWindow .....	20
3.3.3	Klasse CyclicRead .....	20
3.3.4	Klasse Subscription .....	20
3.4	Ausführen des Beispiel-Programms .....	23
3.5	Import des Beispiel-Projektes in Eclipse .....	24
<b>4</b>	<b>Aufzeichnen von SOAP-Telegrammen zu Diagnosezwecken .....</b>	<b>29</b>
4.1	Einstellungen des Trace Tools MSoapt3 .....	29
4.2	Beispiele der Telegrammaufzeichnungen .....	30
<b>5</b>	<b>Appendix .....</b>	<b>33</b>
5.1	Quelltexte and Softwaredokumentation .....	33
5.2	Weiterführende Informationen .....	33
5.3	Literatur- and Quellenverzeichnis .....	33

## 1 How can you use the OPC XML-DA interface?

The present document describes the use of the OPC XML-DA (Open Process Control XML-DataAccess) server functionality of the SIMOTION control. The interfaces to the OPC communication between an OPC-Client and the control are presented with a sample program being realized in Java. Figure 1-1 shows the schematic structure of the OPC XML-DA client-server system.

Figure 1-1: Schematic structure of the client-server system



From the firmware version 4.1 on, the OPC XML-DA server is integrated in the SIMOTION runtime. By means of standard Ethernet or PROFINET connection, it is possible to realize a data exchange between this OPC server and a client application. This makes it possible to access to certain program variables as well as to system functions of the control without additional engineering tools such as SIMOTION Scout. The available data are made available by the so-called „variable provider“ (see chapter 5 - „variable provider“ of the SIMOTION documentation: SIMOTION IT - Ethernet based HMI- and diagnostics function „product information SIMOTION DIAG.pdf“).

The following sections of this document describe the basics of the applied technologies. With the sample program „OPC-XML-Browser“, the variables - provided by the SIMOTION control per OPC - can be searched. With the aid of the source codes, the OPC functions for access and communication are explained in detail.

## 2 Prerequisites and targets of these FAQs

This document explains how to deal with OPC XML-DA by giving an example with OPC client software in Java. The target is to offer a simple pattern for a Java application, with which the customer can develop his own application according to his special requirements. For this, the source text of this Java application is made available for free.

For an optimal comprehension of this document as well as for using the example source texts, programming skills in Java are necessary. Furthermore, it is absolutely essential to familiarize yourself with the OPC XML-DA 1.0 specification.

To change the existing projects, we recommend using the Java development environment Eclipse, which is free of charge. You can find a detailed description for its use in chapter 3.4 in this document.

### 2.1 What is OPC about? Why OPC XML-DA?

OPC describes a standardized and special possibility of communication between different terminals that is independent of the manufacturer. The maintenance and the further development of the standard are subject to the OPC foundation (<http://opcfoundation.org>).

The communication by means of OPC-DA (OPC Data Access, not to be confused with OPC XML-DA), is based on Microsoft COM/DCOM technology (Component Object Model, Distributed COM, see <http://www.microsoft.com/com/> for further information) and assumes a Microsoft-based operating system for the application of this technology.

However, with OPC XML-DA you come off from the COM technology and, with the XML document format, you will use an open standard that can be used by many operating systems. The real communication between the terminals is realized via the SOAP (Simple Object Access Protocol, a permanently defined XML data structure). SOAP again is based upon the http protocol. This already shows that, from the hardware side, a data exchange via OPC XML-DA is based upon an Ethernet connection.

As a principle, the communication with OPC-DA and OPC XML as well is a client-server system. An OPC server makes certain data available, and an according OPC client is in the position to call data from the server or to change them on the server. To do so, a server has to offer certain determined methods and data interfaces according to OPC specification, which can be accessed to by the client. For example, each server has to offer a read function, which permits the client to read out an existing variable or data set (precisely: OPC item(s)) of the server. For example, a further function 'write' that realizes a writing, thus a change of a variable made available by the server. In Table 2-1 you can find a comparison in note form between OPC XML-DA and OPC-DA.

## Data exchange via OPC XML

Table 2-1: Comparison of OPC XML-DA and OPC-DA

SIMOTION IT OPC XML-DA	SIMATIC NET OPC-DA
Keine Projektierung (OPC-Export) mit SCOUT nötig. Programmvariable über Schalter aktivierbar.	OPC-Export mit SIMOTION SCOUT erforderlich, zu wiederholen bei jeder Projektänderung.
Symbole werden erst im SIMOTION - Gerät aufgelöst, Kommunikation per Textformat (XML).	Symbole werden beim OPC-Export aufgelöst und im OPC-Server auf Windowssystem binär hinterlegt, Kommunikation binär-> höher Datendurchsatz.
Derzeit nur SIMOTION mit OPC XML-DA. Zugriff auf S7-Geräte derzeit noch nicht möglich.	Gleichzeitiger Zugriff auf SIMOTION und S7-Geräte möglich.
Client läuft auf beliebigen Betriebssystemen.	Basiert auf Windows COM/-DCOM-Technik, Client und Server laufen nur auf Windowsbetriebssystemen.
Kommunikation über Standardprotokolle (TCP/IP, XML, SOAP), keine herstellerspezifischen (SIEMENS) Tools, Treiber auf Clientsystem notwendig.	Verwendung S7-Protokoll zur Kommunikation, auf Client-Seite entsprechende herstellerspezifische Treiber notwendig.
Kommunikation nur über Ethernet möglich.	Kommunikation über PROFIBUS/MPI und Ethernet möglich.
Direkte Adressierung über Firewalls möglich.	DCOM - Kommunikation wird in der Regel an Firewalls nicht freigegeben.

## 2.2 Particular properties of the OPC XML-DA communication

As explained in Table 2-1, the utilization of the OPC XML-DA standard has especially the advantage to be independent of the operating system. Only this permits the realization of the OPC server functionality on a Simotion control, as SIMOTION RT is not a Windows-based system and thus, OPC-DA is not available, here. With the utilization of the open SOAP protocol standards, it is also possible to access to these server functionalities, independent of the operating system. This way, any client system being able to process SOAP can be applied as OPC client.

Due to the communication structure being based on http, a mutual, thus server and client side initialization of the connection is not possible. A communication between OPC client and OPC server has always be initiated by the client side so that the call back mechanism of the server with OPC XML-DA cannot be realized easily. With OPC-DA, the server itself is able to establish the connection to the client in order to send a message, for example, that certain data have changed. To come to know about a changed value in the OPC server within a certain time interval, an OPC XML-DA client ought to request this value continuously at least one time within the desired time interval (so-called polling) If the value to be requested does not change within a series of cycles, it will still be requested each time and transmitted to the client. In case of very short time intervals, this leads to a high network communication and, depending on the system, to a high system load.



## Data exchange via OPC XML

To reduce this system load, OPC XML-DA offers the function 'subscription'. Please refer to chapter 3.3.4. in this document to find detailed information about the mode of functioning of the subscription.

Data exchange via OPC XML

### 3 OPC client software in Java

The following section describes in detail the Java sample program with regard to the functionalities of the OPC and its application. On the basis of this description it is possible to expand the source text. Furthermore, there is a description of how to create a project within the development environment Eclipse.

#### 3.1 Graphical user interface of the Java program

Figure 3-1: User interface of the Java program

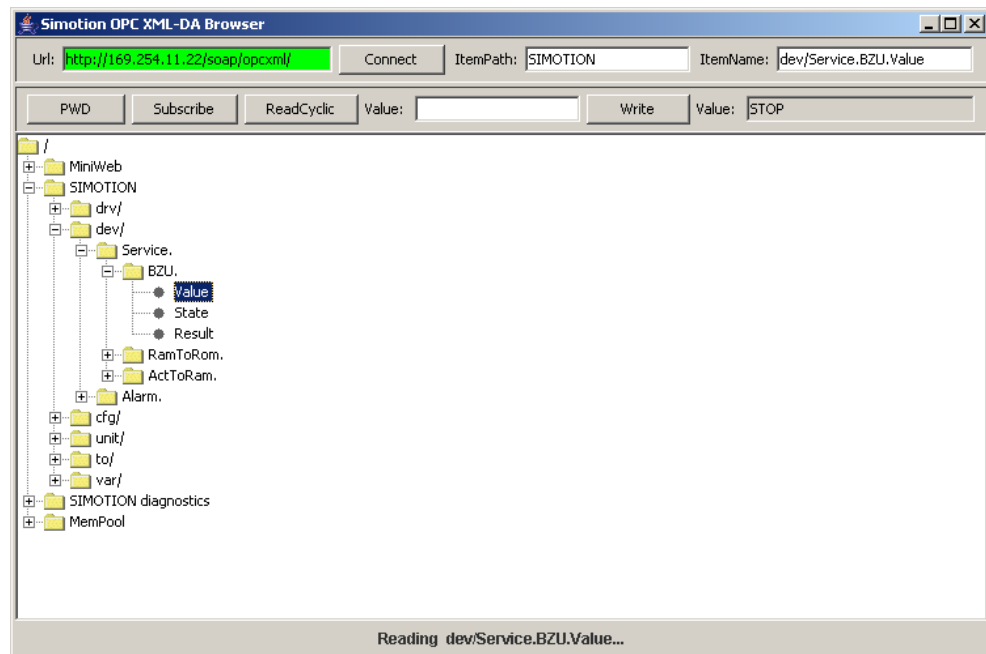


Figure 3-1 shows the graphical user interface (GUI). The following OPC functionalities are supported:

- **Read:** By clicking on a new page in the browse tree (causes an implicit change of ItemPath and ItemName), the current value of the selected variable is read from the OPC server. The value of the variable is indicated in the output field (here "STOP" as an example). In addition to this ItemPath and ItemName can be entered to these fields manually. Pressing "Enter" will result in reading the specified item from the OPC server. In case the item does not exist, a corresponding message is displayed in the status area.
- **Write:** Writing of the Simotion variables being specified in the input fields ItemPath and ItemName. The writing process is triggered by changing

## Data exchange via OPC XML

the value in the input field ‚Var’ and by a final click on the „Write“ button or completion of the input by „Return“.

- **Browse:** Browsing the tag management in the Simotion control; the functionality is similar to the Windows Explorer. With every opening of the node, the belonging information is detected by the Simotion control. As long as the Browse response is not available, yet, or there are no variables (corresponds to the pages of the tree), a dummy variable is visible as a page.
- **Connect:** By changing the URL (input by clicking on the button „Connect“ or complete with „Return“) a connection to a new Simotion device can be established. Doing so, the existing variable tree is renewed.  
An established connection is visualized by a green colored background and a missing connection is indicated by an orange background.
- **In the PWD window,** login name and password can be changed, default settings are the standard settings of the Simotion control (user name: simotion; password: simotion).
- **Subscribe:** By clicking onto „Subscribe“, the OPC function Subscription can be initialized. After the input of the required values (or acceptance of the preset standard values) in the following input dialog (Figure 3-2), the variable being indicated in the field ItemPath and ItemName is registered for a Subscription. For a detailed description of this functionality being available in OPC XML-DA, please refer to chapter 3.3.4.

Figure 3-2: configuration mask ‚subscription’

Item Name:	dev/Service.BZU.Value
Item Value:	STOP
Holdtime (ms):	2000
Waittime (ms):	3000
Status:	Value change at: 11:00:27 a.m.

Buttons: Cancel subscription, Exit

- **ReadCyclic:** By pushing ReadCyclic you can start a cyclic reading („Polling“) with the adjusted interval of Simotion variables being specified in the input fields ItemPath and. You can stop the cyclic reading by pushing the button ‚stop’ which is visible then. The number of the realized reading calls is indicated in the output field ‚Read Count:’

### Note

Depending on the version and configuration of the Java Virtual Machine, the design of the graphical user interface being indicated in Figure 3-1 may be different. However, this does not influence the functionality of the program.

## Data exchange via OPC XML

### 3.2 Creation of the classes for the OPC communication

The classes permitting the communication with the OPC server are contained in the following two packages

„org.opcfoundation.webservices.XMLDA.\_1\_0“ as well as  
 „org.opcfoundation.webservices.XMLDA.\_1\_0.holders“

They are available as source text files as well as compiled Java classes (\*.class) and have been generated by means of the tool „WSDL2Java“ (see Apache Axis). This tool permits to generate Java source code from a WSDL<sup>1</sup> - file, which realizes the conversion from SOAP telegrams into Java. The file OpcXmlDa\_R1\_0.wsdl, indicated in parts in Figure 3-3, is the basis for the generated Java classes and is made available as a standard for the OPC XML-DA-communication by the OPC foundation (<http://opcfoundation.org/webservices/XMLDA/1.0/>). It is a description of the, generated in XML format, which are required for the communication of client application and OPC server. In Figure 3-3, the element „Write“ is indicated among others.

Figure 3-3: Extract from the WSDL file OpcXmlDA\_R1\_0.wsdl

```

221     <s:element name="Write">
222       <s:complexType>
223         <s:sequence>
224           <s:element minOccurs="0" maxOccurs="1" name="Options" type="s0:RequestOptions" />
225           <s:element minOccurs="0" maxOccurs="1" name="ItemList" type="s0:WriteRequestItemList" />
226         </s:sequence>
227         <s:attribute name="ReturnValuesOnReply" type="s:boolean" use="required" />
228       </s:complexType>
229     </s:element>
230     <s:complexType name="WriteRequestItemList">
231       <s:sequence>
232         <s:element minOccurs="0" maxOccurs="unbounded" name="Items" type="s0:ItemValue" />
233       </s:sequence>
234       <s:attribute name="ItemPath" type="s:string" />
235     </s:complexType>
236     <s:element name="WriteResponse">
237       <s:complexType>
238         <s:sequence>
239           <s:element minOccurs="0" maxOccurs="1" name="WriteResult" type="s0:ReplyBase" />
240           <s:element minOccurs="0" maxOccurs="1" name="RItemList" type="s0:ReplyItemList" />
241           <s:element minOccurs="0" maxOccurs="unbounded" name="Errors" type="s0:OPCError" />
242         </s:sequence>
243       </s:complexType>
244     </s:element>
  
```

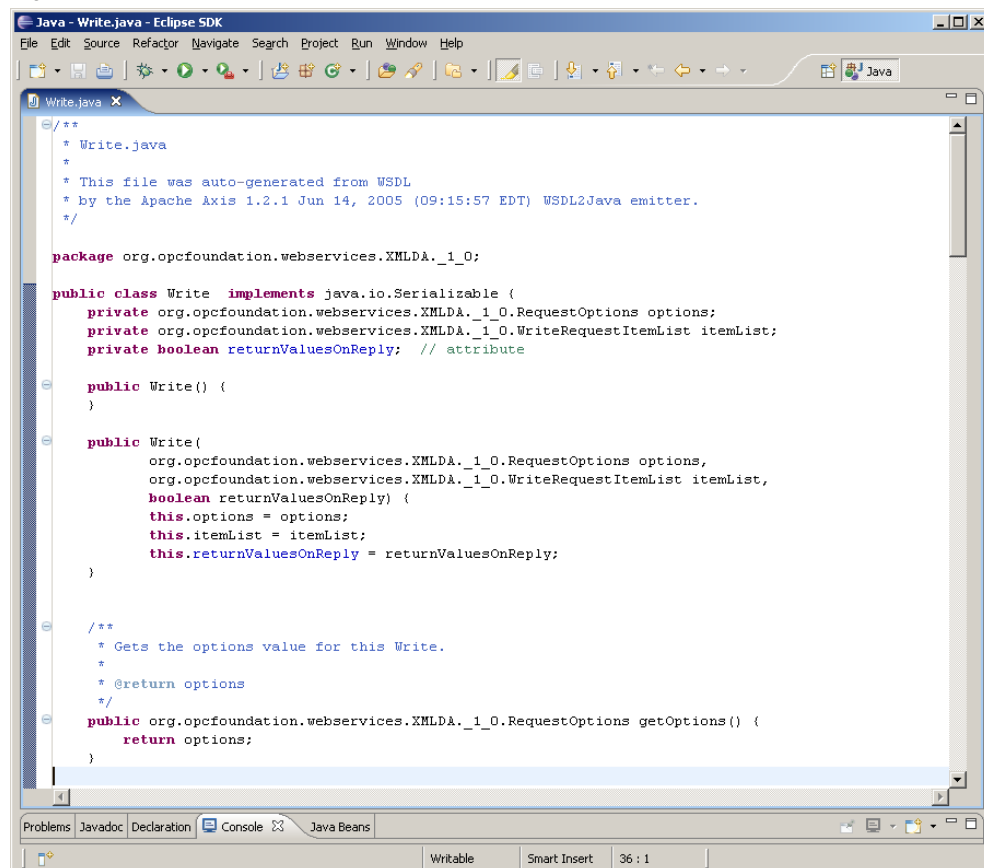
From the element described here, the tool „WSDL2Java“ creates, for example, the class Write, with the three variables „Options“ of type

<sup>1</sup> WSDL – Web Service Description Language, for further information please refer to <http://www.w3.org/TR/wsdl>

## Data exchange via OPC XML

RequestOptions, „ItemList“ of type WriteRequestItemList and „ReturnValuesOnReply“ of type Boolean. Figure 3-4 indicates an extract from the class generated from that.

Figure 3-4: Extract of the class Write



```
Java - Write.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help

Write.java x
/**
 * Write.java
 *
 * This file was auto-generated from WSDL
 * by the Apache Axis 1.2.1 Jun 14, 2005 (09:15:57 EDT) WSDL2Java emitter.
 */

package org.opcfoundation.webservices.XMLDA._1_0;

public class Write implements java.io.Serializable {
    private org.opcfoundation.webservices.XMLDA._1_0.RequestOptions options;
    private org.opcfoundation.webservices.XMLDA._1_0.WriteRequestItemList itemList;
    private boolean returnValuesOnReply; // attribute

    public Write() {
    }

    public Write(
        org.opcfoundation.webservices.XMLDA._1_0.RequestOptions options,
        org.opcfoundation.webservices.XMLDA._1_0.WriteRequestItemList itemList,
        boolean returnValuesOnReply) {
        this.options = options;
        this.itemList = itemList;
        this.returnValuesOnReply = returnValuesOnReply;
    }

    /**
     * Gets the options value for this Write.
     *
     * @return options
     */
    public org.opcfoundation.webservices.XMLDA._1_0.RequestOptions getOptions() {
        return options;
    }
}
```

In the constructor (line 21-29), the variables are initialized. In the further process of this class, there are methods with which new values can be set (e.g. setOptions(options)), or with which the actually stored can be read out (e.g. getOptions()).

However, when dealing with these classes generated by WSDL2Java, it is recommended to use the classes supplied with application (Java package „org.opcfoundation.webservices.XMLDA.\_1\_0“), as within different versions of Apache Axis, there often version conflicts.

Data exchange via OPC XML

**Note**

During the creation of the communication class by WSDL2Java, the creation of the two classes `service.java` and `serviceStub.java` were incorrect. The result was that for the utilization of the read function it became necessary to make corrections within the above mentioned classes. Furthermore, this led to a slightly changed call of the read functions within the client application. For further information, please refer to chapter 3.3.1. In the software version that is made available here, however, this error has been eliminated, and the read function can be applied without faults.

### 3.3 Class description of the OPC browser application

In the following, the single classes of the Java program and its function mode are explained. The real application, i.e. the OPC browser, consists of four basic classes. Table 3-1 contains an overview as well as a short description of each class. In the following section of this document you can find a detailed description of the methods contained in the single classes.

Table 3-1: Class overview of the applications

Java example application	
<b>MainWindow</b>	Class for the initialization of all initial values of the program. Contains, among others, the GUI, some OPC communication functions as well as the <code>main()</code> function
<b>PWDWindow</b>	Class for the creation of the window to enter the password
<b>CyclicRead</b>	Class for the creation and update of the window to enter the parameters of the program function Cyclic Read
<b>Subscription</b>	Class for the creation and update of the window to enter the parameters of the program function Subscription

In the following the OPC XML-DA access functions `getStatus`, `browse`, `read`, `write` and `subscription` are described. `getStatus`, `browse`, `read` and `write` are realized within the class `MainWindow`, the function `subscription` accordingly in the class of the same name `Subscription`. The class `CyclicRead` is an extension of the `read` function and permits a cyclic reading of a certain variable. This corresponds to the polling of a variable being already mentioned in chapter 3.1.

---

## Data exchange via OPC XML

For each of these functions there is a Java class within the package „org.opcfoundation.webservices.XMLDA.\_1\_0“ (e.g. `Read.class`, `Write.class` etc.), which was automatically generated from the WSDL file. There are further classes existing in addition, for the function `write` e.g. `WriteResponse.class` or `WriteRequestItemList.class`. Depending on the OPC function, these classes have certain tasks, e.g. the class `WriteResponse.class` is the data type for the data of a `write` call being returned by the server.

Basically, the access to the OPC XML-DA server of the Simotion control is realized as follows (this is an example with the function `write`):

1. First, a global instance of the class `ServiceStub` named `mySimotionWebService` is generated. The class `ServiceStub` results from the WSDL file and was generated automatically. It is the basic class for the web services being usable by means of Apache Axis. You can find further information on this in the documentation of the Apache Axis software.
2. Creation of an instance and the following initialization of the class `Write`. When calling the `write` function it gets the parameters to be transmitted to the OPC server. For the parameterization of this instance of `Write` there are instances of further classes required as according data types, for example `WriteRequestItemList`, which contains the list of the items to be written to the OPC server.
3. Creation of an instance of the belonging response class (here `WriteResponse`, this class exists for all OPC functions, e.g. `ReadResponse`, etc.). After calling the OPC function `write`, this instance will contain the data returned by the OPC server, in this example a status message if `write` was successful
4. After the OPC function has been executed, the data transmitted by the OPC server are evaluated, thus the data contained in the instance of the class `WriteResponse`. These data are now available for further processing within the Java program.

### 3.3.1 Class `MainWindow` (`MainApplet`)

The class `MainWindow` contains the fundamental functions for the creation of a graphical user interface (GUI) of the program, which is not elaborated any further in this place.

Furthermore, it contains the implementations of the OPC functions `getStatus`, `browse`, `read` and `write`, which will be explained in detailed in the following. The method `getStatus` is a very good entry as the belonging OPC call is based upon a rather simple data structure. However, the method `mySimotionWebService_SetUp` should be

---

## Data exchange via OPC XML

regarded first: This method does not realize an OPC function, but initializes an instance of the class `ServiceStub`, on which the OPC communication is based upon.

```
private void mySimotionWebService_SetUp()
```

As described in chapter 3.3 - 1., the variable `mySimotionWebService` is generated as global variable within the Java program by the call

```
org.opcfoundation.webservices.XMLDA._1_0.ServiceStub  
mySimotionWebService;
```

This way it is available within all other methods. The initialization of `mySimotionWebService` is realized in the method regarded here:

```
mySimotionWebService =  
    new org.opcfoundation.webservices.XMLDA._1_0.ServiceStub  
    (new java.net.URL(jTextFieldURL.getText()), null);  
  
mySimotionWebService.setUsername(sUName); //credentials  
mySimotionWebService.setPassword(sPWD);
```

An instance of the class `ServiceStub` is generated.

The URL entered in the URL text field of the Java program is transmitted to the constructor of the class, which corresponds to the URL under which Simotion control (and thus the OPC server) is accessed to. The second transmitted variable (`zero`) is not regarded any further. Finally, user name and password are set.

```
private void getStatus()
```

```
org.opcfoundation.webservices.XMLDA._1_0.GetStatus parameters =  
    new org.opcfoundation.webservices.XMLDA._1_0.GetStatus();
```

The variable `parameters` is an instance of the class `GetStatus`. In the following, this variable is initialized, in this case only with the assignment

```
parameters.setLocaleID("en");
```

Until here, this corresponds to the summary in chapter 3.3 under 2. With this rather simple call `getStatus`, the initialization of the `parameters` consists only of this single assignment. Further parameterizations as described in chapter 3.3- 2. are not required here. It is sufficient to initialize the variable `parameter` directly.

Now the response class is instantiated to `GetStatus` (see chapter 3.3- 3.):

```
org.opcfoundation.webservices.XMLDA._1_0.GetStatusResponse  
getStatusResp = null;
```



## Data exchange via OPC XML

The variable `getStatusResp` of type `GetStatusResponse` will receive the data returned by the OPC server after calling the `getStatus` method. This assignment is realized with the real call of the OPC function (see chapter 3.3- 4.):

```
getStatusResp = mySimotionWebService.getStatus(parameters);
```

Here, the actual call of the function `getStatus` is sent to the OPC server. The data transmitted by the server, are stored in the variable `getStatusResp` of data type `GetStatusResponse`. An evaluation of these data sent by the server, thus a conversion into a Java program variable, is realized in the following two instructions

```
ServerStatus serverStatus = getStatusResp.getStatus();  
String simotionServerStatus = serverStatus.getStatusInfo();
```

The data type `ServerStatus` is also a class being generated from the WSDL file. The variable `simotionServerStatus` of type `String` can be used within the Java program and output, for example, as a status message.

**private void browse()**

The communication to the OPC server with the function `browse` is very similar to the above described function `getStatus`. First, a variable `parameters` is generated again, in this case as an instance of the class `Browse` instead of instance of class `getStatus`. Then, in addition to the call

```
parameters.setLocaleID("en");
```

which is also executed above, there are still two member variables `itemPath` and `itemName` being initialized by `parameters`:

```
parameters.setItemPath(itemPath);  
parameters.setItemName(itemName);
```

A particularity in this place is that after the first establishment of the connection of the client to Simotion control, the method `browse` is called without any currently existing variables `itemPath` and `itemName`. In this case, empty strings are transmitted to the OPC server with the request to return the root node of its internal variable structure.

During the following process of this method, the data received from the OPC server are processed further, analogous to `getStatus`.

In this place, the received OPC items of the internal server variables are definitely represented as a tree structure within the GUI.

**private void write()**

A `write` call is based upon a more complex data structure:

## Data exchange via OPC XML

```
(1) org.opcfoundation.webservices.XMLDA._1_0.Write
    parameters = new (...)Write();

(2) org.opcfoundation.webservices.XMLDA._1_0.RequestOptions
    options = new (...)RequestOptions();

(3) org.opcfoundation.webservices.XMLDA._1_0.WriteRequestItemList
    itemList = new (...)WriteRequestItemList();

(4) org.opcfoundation.webservices.XMLDA._1_0.ItemValue[] items =
    new ItemValue[1];

(5) org.opcfoundation.webservices.XMLDA._1_0.ItemValue
    item = new ItemValue();
```

Like `getStatus` and `browse`, an instance of the class `Write` is generated automatically (here again a variable named `parameters` (1)). This variable `parameters` is, as already seen before, transmitted to the OPC server on calling the `write` function. However, in this case a different initialization of the function is necessary. For this, the variable `options` is declared (2) and then assigned again to the variable `parameters` after being initialized itself (`options.setLocaleID("en")`):

```
parameters.setOptions(options);
```

The variable `itemList` follows as an instance of the class `WriteRequestItemList` (3) as well as the variables `items` (4) and `item` (5) as an instance of `ItemValue`. This structure seems to be redundant at first sight, but it is necessary to maintain the data structure being defined by the OPC XML-DA specification. Thus, `itemList` is a list of items, and this list of items needs again a list of corresponding values (i.e. `ItemValues` (4)). As, in this case, only one item is transmitted to the server, `items` is initialized as content as array with only one element (also (4)). Finally, the variable `item` (5) represents the actually item to be transmitted. The whole structure becomes more clearly if you look at the following assignments:

```
(1) item.setItemPath(itemPath);
(2) item.setItemName(itemName);
(3) item.setValue(str); //String str is the entered value of the
                        //according items to be written into the GUI
(4) items[0] = item;
(5) itemList.setItems(items);
(6) parameters.setItemList(itemList);
```

First, the variable `item` is assigned with the current values, i.e. `itemPath` (1), `itemName` (2) and `str` (3) as the value to be written to the OPC server. Then, this `item` is written to position 0, thus the first (and in this case the only) position of the Array `items` (4). By means of the method

## Data exchange via OPC XML

`setItem`s, the Array `items` is now assigned to the variable `itemList` (5), which again is assigned to the variable `parameters` by means of `setItemList` (6). By this, the variable `parameters` is initialized for the write function of the OPC server. Analogous to the previous functions, only the `WriteResponse` type is declared:

```
org.opcfoundation.webservices.XMLDA._1_0.WriteResponse  
writeResponse = null;
```

The actual `write` access to the OPC server is realized analogous to the already described functions.

```
writeResponse = mySimotionWebService.write(parameters);
```

In this example, the only evaluation of `writeResponse` is to check if it is zero or not. If not, the current value (which has been written right now) of the OPC items is requested when calling the `read` function (see next section). A further possibility to return the value of the currently written variable by the server in the `writeResponse` would be, for example, by means of the following instruction

```
parameters.setReturnValuesOnReply(true);
```

Any other possible options and settings of the parameters to be transmitted to the OPC server can be found in the OPC XML-DA specification 1.0.

**private String read()**

The data structure of the parameters of method `read`, which reads an OPC item of the server, is very similar to that one of the `write` call. The difference to `write` is that the `read` function or the class `Read` was generated from the WSDL file with errors (see chapter 3.2). This means that when doing the call after the error correction not only one variable (`parameters`) is transmitted as described above, but the parameter `options` and `itemList`, which, however, have been initialized analogous to the above mentioned. This means that the two assignments

```
parameters.setOptions(options);  
parameters.setItemList(itemList);
```

are dropped, which (in the above cases) initialize the variable `parameters` to be transmitted to the OPC server. Instead of that, in this case, the two parameters `options` and `itemList` are directly transmitted:

```
readResponse = mySimotionWebService.read(options,itemList);
```

Of course, these two parameters are initialized before with name, path etc., analogous to the `write` function.

A further difference to the previous methods is that the method `read` is of type `String`, i.e. transmits a `String` as return value. This return value is that value of the item that was just read by the OPC server, which is

---

## Data exchange via OPC XML

determined from the variable `readResponse` during the process of the method and then assigned to the return value of the `read` method.

### 3.3.2 Class PWDWindow

The task of class `PWDWindow` is only to create and indicate a new window with a user name password request. The entered data are written in the variables `String sUName` and `String sPWD` and then transmitted to the string of the same name in `MainWindow`, which was declared as global variable there.

### 3.3.3 Class CyclicRead

The class `CyclicRead` realizes a cyclic reading of a variable in the OPC server. For this, in the main program of the class `MainWindow` an instance of the class `CyclicRead` is generated. To do so, the currently selected item in the variable tree is transmitted to `CyclicRead`. `CyclicRead` now generates a new window, where the further parameters of the cyclic reading can be entered.

The basic functionality, i.e. the declaration and initializations of the required variables and parameters correspond to those of the standard-`Read` function as described in chapter 2.2.1. The cyclic reading of the variables, i.e. the cyclically repeated call of the `read` method is realized by the standard Java class `Timer` and the belonging function of a `TimerTask`. For further information on this subject, please refer to the standard Java API<sup>2</sup> (for Java 1.4.2 e.g. under <http://java.sun.com/j2se/1.4.2/docs/api/>).

### 3.3.4 Class Subscription

Basically, you could interpret the function `Subscription` as „intelligent polling“. The client registers one or more variables for the `Subscription` in the server. Then, by means of the function `SubscriptionPolledRefresh` this intelligent polling is started. From the technical side, the client establishes a connection to the server, which the server maintains for a certain period of time by delaying the response. If the values in the server that are registered for the subscription change within this period of time, the server sends a response directly after this change to the client, including the new values. If none of the values change within this period of time, an empty response is returned to the client, which can then be evaluated accordingly by the client. This makes it possible to achieve a reduction of the data transfer as well as of the system load caused by high frequent polling (see cyclic reading). With the call of the method `SubscriptionCancel` a running `Subscription` is terminated. Within a `Subscription`, i.e. the registration by means of the method

---

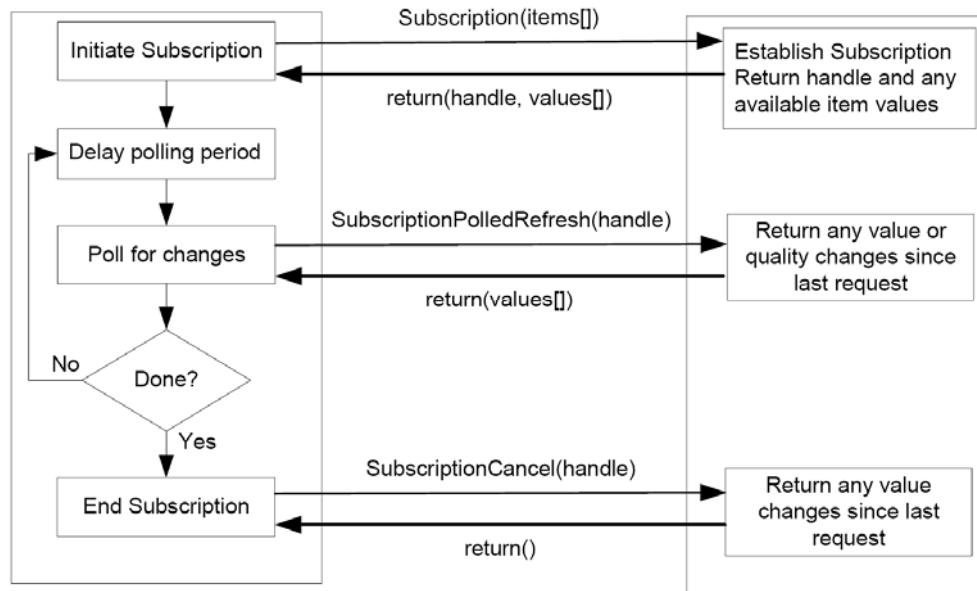
<sup>2</sup> application programming interface

## Data exchange via OPC XML

subscribe, one or more variables can be monitored. The identification of the variables registered for the subscription is realized via `serverHandles`. These `serverHandles` are returned by the OPC server when calling the method `subscribe`. The client uses these handles to identify the variable(s) to be responded when calling the methods `SubscriptionPolledRefresh` as well as `SubscriptionCancel`.

Figure 3-5 shows the entire process of the subscription function, which will be explained in detail after the figure. As it is already visible by now, there are some additional indications necessary for the subscription compared to the OPC functions we have seen, yet. In the following you can find a description of the entire function subscription.

Figure 3-5: Flow chart of a Subscription



1. First, just like it is the case for any other OPC function – a global instance of the class `ServiceStub` named `mySimotionWebService` is generated.
2. Then follows the declaration and initialization one instance each of the classes `Subscribe`, `SubscriptionPolledRefresh` and `SubscriptionCancel`. Just like it is the case for any other OPC functions, these instances are responsible for the transmission of the parameters to the OPC function.
3. The instances from 2. are initialized with the required values. To do so, further variables have to be generated, e.g. an instance of the class `SubscribeRequestItem`, which, in this case, only contains an item to be registered for the subscription.

---

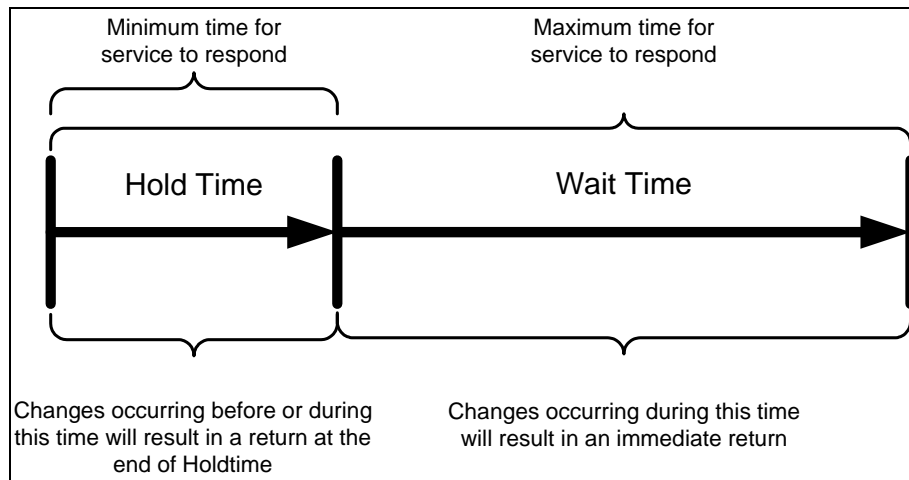
## Data exchange via OPC XML

4. The method `subscribe` is called, the previously assigned item is registered for the subscription in the OPC server.
5. By means of the method `SubscriptionPolledRefresh` the value of a variable is checked. This method has to be called cyclically, the responses of the server can be delayed by the two parameters `holdtime` and `waitTime` according to the required relevance to the time. This is the actual advantage of a subscription over a simple polling. `waitTime` is the period of time that the server shall wait for the change of a variable. The `holdtime`, however, describes a period of time that the server has to wait in any case before returning a variable value, no matter if there is a change of value or not. Thus, the real monitoring of a variable starts after the `holdtime`, after that, i.e. within the `waitTime`, the server transmits any change of variable directly to the client.
6. Figure 3-6 makes this functionality clear. By adapting these two parameters and depending on the requirements of the application with regard to the time, a compromise between period of time for notifying after change of value and network transfer can be found. It has to be considered here, that the OPC server as `holdtime` expects an absolute time in form of a `java.util.Calendar`. The inputs of the GUI in milliseconds are converted in the Java program into a variable of type `Calendar`. The basic time value is the moment of the server's response to the registration, i.e. the parameter `ReplyTime` which is returned with the method `subscribe`. However, the OPC server expects milliseconds as time indication for `waitTime` so that this can be taken directly from the GUI and transferred to the according parameter.

**Note**

The time indicated in the subscription dialog is based upon the SIMOTION-internal time settings. Please consider the setting of the time zone with regard to the GMT. During the summer time, the standard setting of GMT is to set from +60min to GMT +120min.

Figure 3-6: Comparison of holdTime and waitTime



7. If the monitoring of the variable shall be terminated, this is done by calling the method `SubscriptionCancel`. By transmission of the `serverHandle` or the array of `serverHandles` with several variables to the OPC server, this one terminates the according subscription, i.e. the monitoring of the variables concerning changes of value.

To avoid affecting the functionality of the remaining program during the cyclic call of `SubscriptionPolledRefresh`, the class `SubscriptionPolledRefreshThread` is used as an extension of the Java class `Thread`. The implementation of the function `SubscriptionPolledRefresh` within this class makes it possible that this one is executed in a separate thread and the 'while' loop containing the cyclic call of `SubscriptionPolledRefresh`, and thus does not affect the execution of the other program.

We do without a detailed description of the whole source text of the class `Subscription` as the basic mechanisms for the initialization of the required parameters etc. are identical to those OPC functions that have already been described. For the `Subscription` only some additional variables for parameterization are used (e.g. `holdTime` or `waitTime`). For more detailed information on the mode of functioning of a subscription, please refer to the OPC XML-DA 1.0 specification.

### 3.4 Execution of the example program

For the execution of the Java example program, the following prerequisites have to be met:

- Installation of Java Runtime Environment JRE V1.4.2 (or higher) - (<http://java.sun.com/j2se/1.4.2/download.html>)

---

## Data exchange via OPC XML

- Extracting of the zip archive OPC-Xml-Browser.zip. There is a directory with the same name OPC-Xml-Browser. The script run.cmd has to be located in the exactly subordinated path.

If these prerequisites are met, you can start the example application by executing the script „run.cmd“.

### 3.5 Import of the example project in Eclipse

To change or to extend the example program, we recommend using the Java programming environment Eclipse. This one is available for free and can be downloaded under <http://www.eclipse.org/downloads/>. The example project has been created and tested with version SDK 3.2.2 of the Eclipse programming environment. The project export and import is described on the basis of this version, too. We cannot exclude eventual changes of the dialogs or a restricted/faulty functionality of the project import as well as the runnability of the example project in general for other versions of the Eclipse SDK.

#### Note

Under <http://www.eclipse.org/documentation/> there is an English documentation of the Eclipse SDK which can help you to start working with Eclipse.

The following steps are necessary to open the project in Eclipse:

Via the menu „File“ → „Import“ (Figure 3-4) the following dialog opens (Figure 3-5). Here, please select the option „Existing Projects into Workspace“ and confirm with „Next“.



## Data exchange via OPC XML

Figure 3-4: Import of an Eclipse project 1

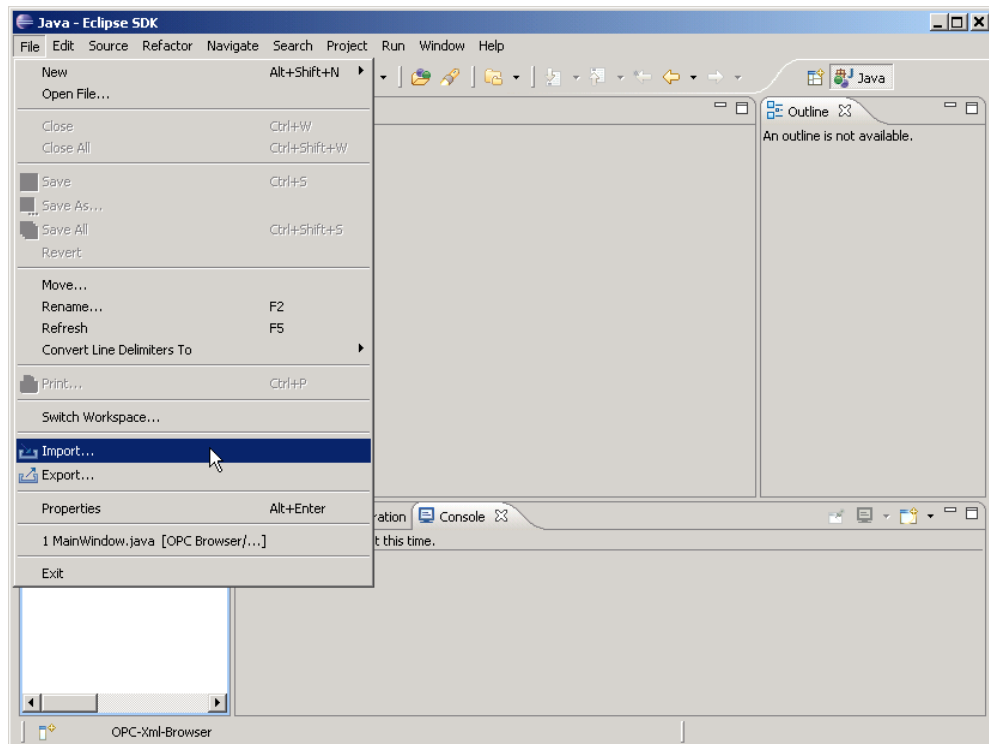
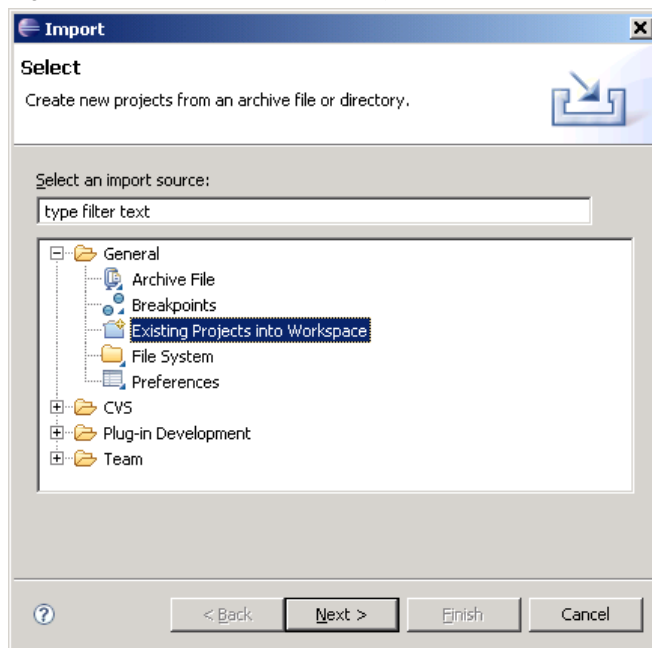


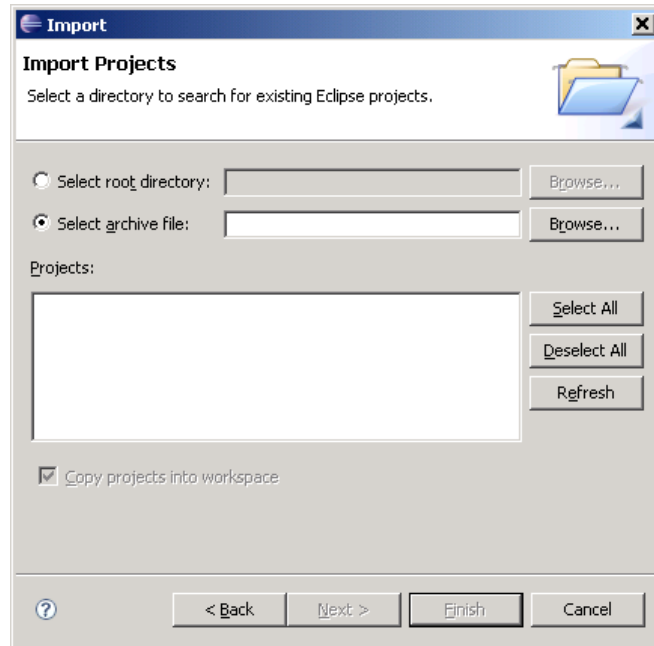
Figure 3-5: Import of an Eclipse project 2



## Data exchange via OPC XML

In the dialog „*Import*“ (Figure 3-6), please click on „*Select archive file*“ and select the zip-File „*OPC-Xml-Browser.zip*“ being supplied together with this FAQ.

Figure 3-6: Import of an Eclipse project 3

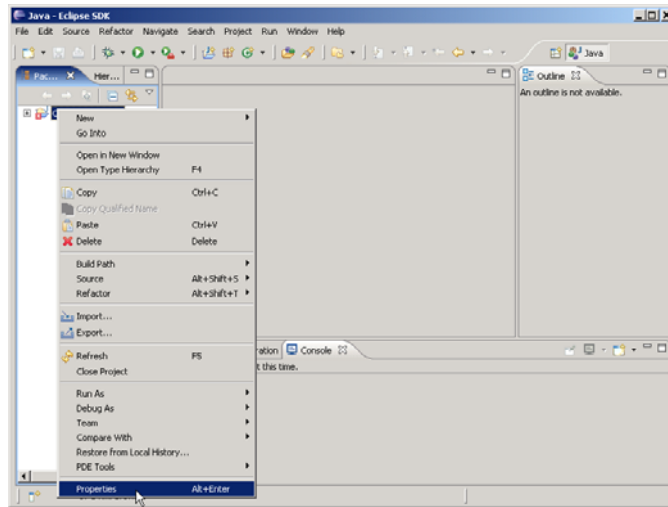


Now, the existing project is imported into the Eclipse Workspace and then compiled automatically. However, it will not be fault-free after this compilation as the necessary external libraries have not been integrated, yet, although they already exist in the project-zip-file, but they are not integrated automatically into the imported project. You can do this afterwards as follows:

Open the context menu by a right click on the imported project and select „*properties*“ (Figure 3-7).

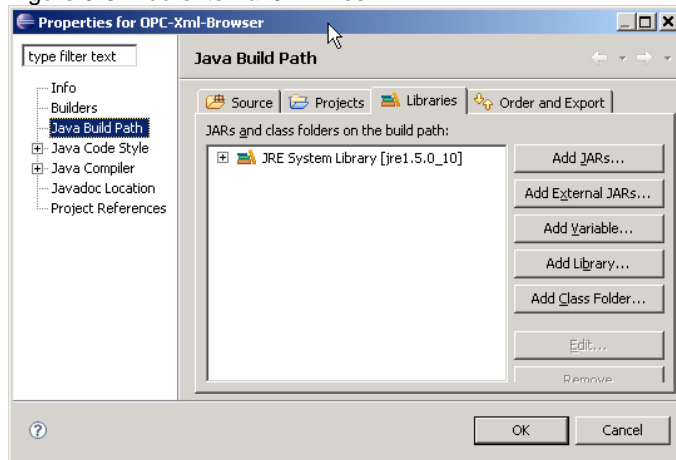
Data exchange via OPC XML

Figure 3-7: Add external JAR files 1



The 'properties' dialog (Figure 3-8) appears. Now, please activate „Libraries“ under „Java Build Path“. By means of the button „Add external JARs“ you can add the missing external libraries.

Figure 3-8: Add external JAR files 2

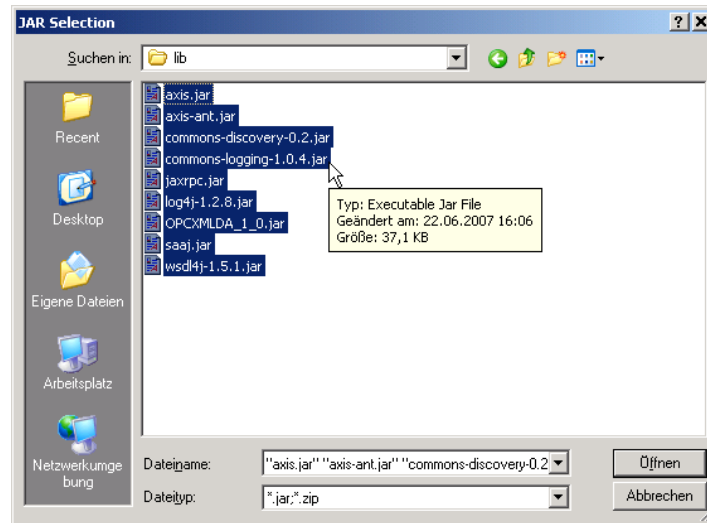


These libraries are stored in the new generated project. During the import of the projects, these libraries are located in the new created project folder „OPC-Xml-Browser“ and there in the sub-folder “lib”. You can find the project folder in “Workspace” of Eclipse which has been created during the initial start of the Eclipse software.

After pushing the button „Add external JARs“ a file selection dialog appears (Figure 3-9). All libraries (\*.jar files) contained in the „lib“ directory have to be selected and added to the project.

## Data exchange via OPC XML

Figure 3-9: Add external JAR files 3



After confirmation of these settings, Eclipse compiles the project again; now the source code is compiled without faults.

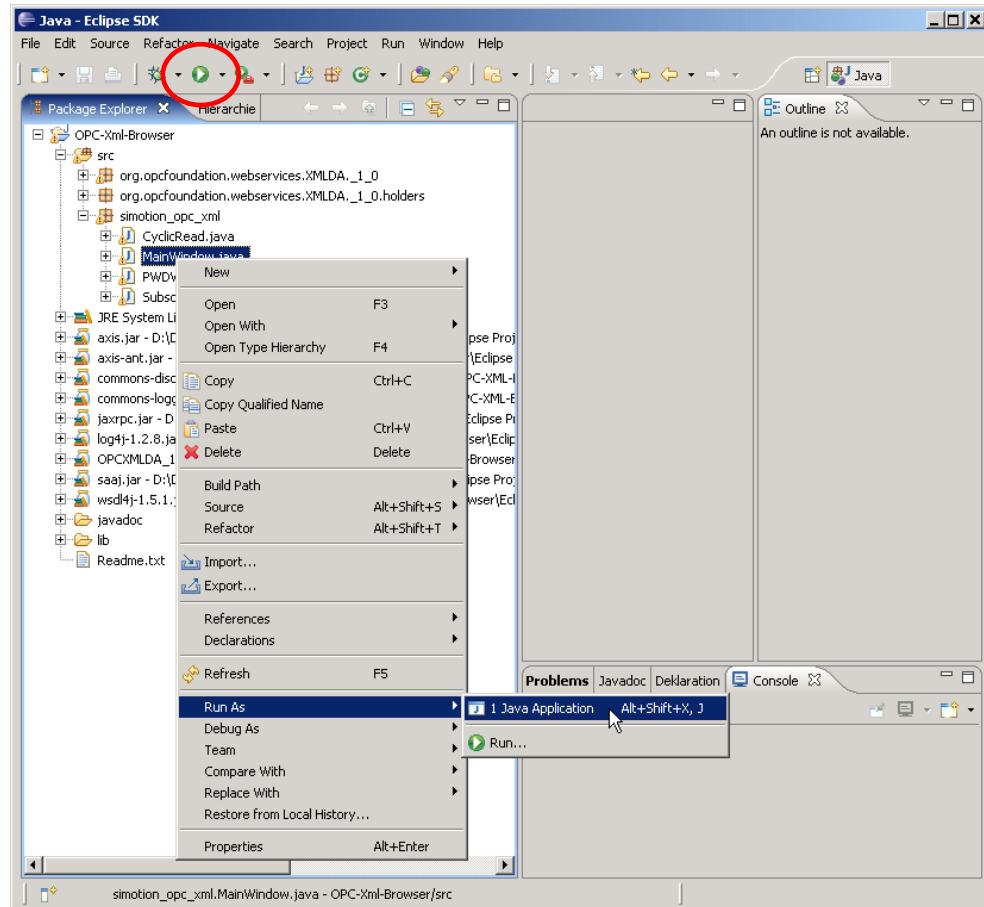
By opening the directory tree „OPC-Xml-Browser → src → simotion\_opc\_xml“ created in the „Package Explorer“, the Java class „MainWindows.java“ can be selected. As described in chapter 3.3, this is the main class of the Java program. A click on this class with the right mouse key opens the belonging context menu and with „Run as → Java Application“ the program is executed (Figure 3-10).

### Note

After the initial execution of the program, you can start it by a click on the red “start” button as shown in Figure 3-10.

## Data exchange via OPC XML

Figure 3-10: Execute program



After having terminated the import of the project, you can change the OPC-browser application.

### Note

After storing a changed source text file, Eclipse compiles automatically the changed file(s). To make changes, you only need to store the sources as Eclipse creates automatically the belonging \*.class files.

## 4 Recording of SOAP telegrams for diagnostics purposes

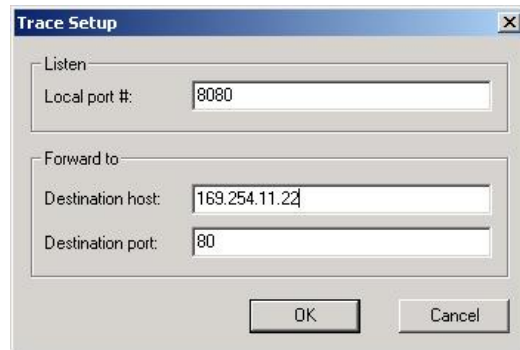
### 4.1 Settings of the Trace Tools MSSoapT3

The sent and received SOAP telegrams can be indicated for diagnostics purposes in XML format by means of the Trace-Tools MsSoapT3.exe.

### Data exchange via OPC XML

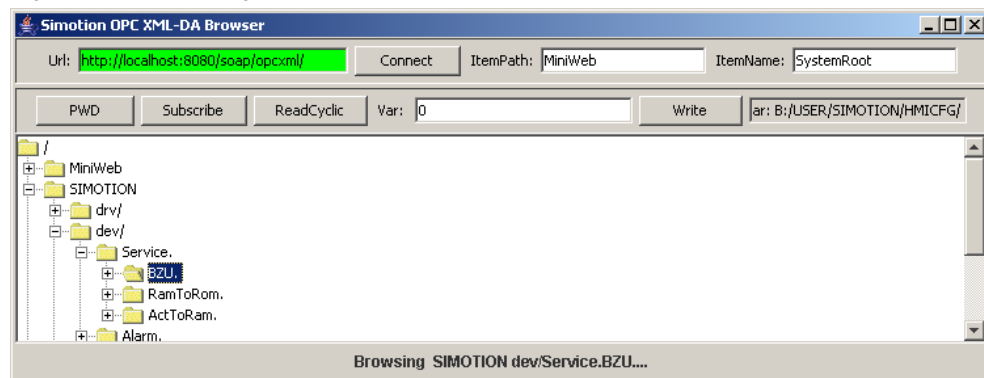
To do so, the following settings have to be selected in MsSoapT3 (Figure 4-1):

Figure 4-1: Setting Trace-Tool



The IP address of the Simotion control has to be entered as „Destination host“. Then the OPC XML browser does not address directly to the control, but uses the following URL instead: <http://localhost:8080/soap/opcxml/> (Figure 4-2). This way, the Trace Tool acts as a gateway and passes all local requests to Port 8080 on the OPC.

Figure 4-2: Addressing in the OPC browser



## 4.2 Examples of the telegram recordings

In Figure 4-3 and Figure 4-4 the recordings of the SOAP telegrams are represented as an example.

Figure 4-3 contains the representation of a browse request to the OPC server and its response.

Data exchange via OPC XML

Figure 4-3: Browse request (above) and response (below)

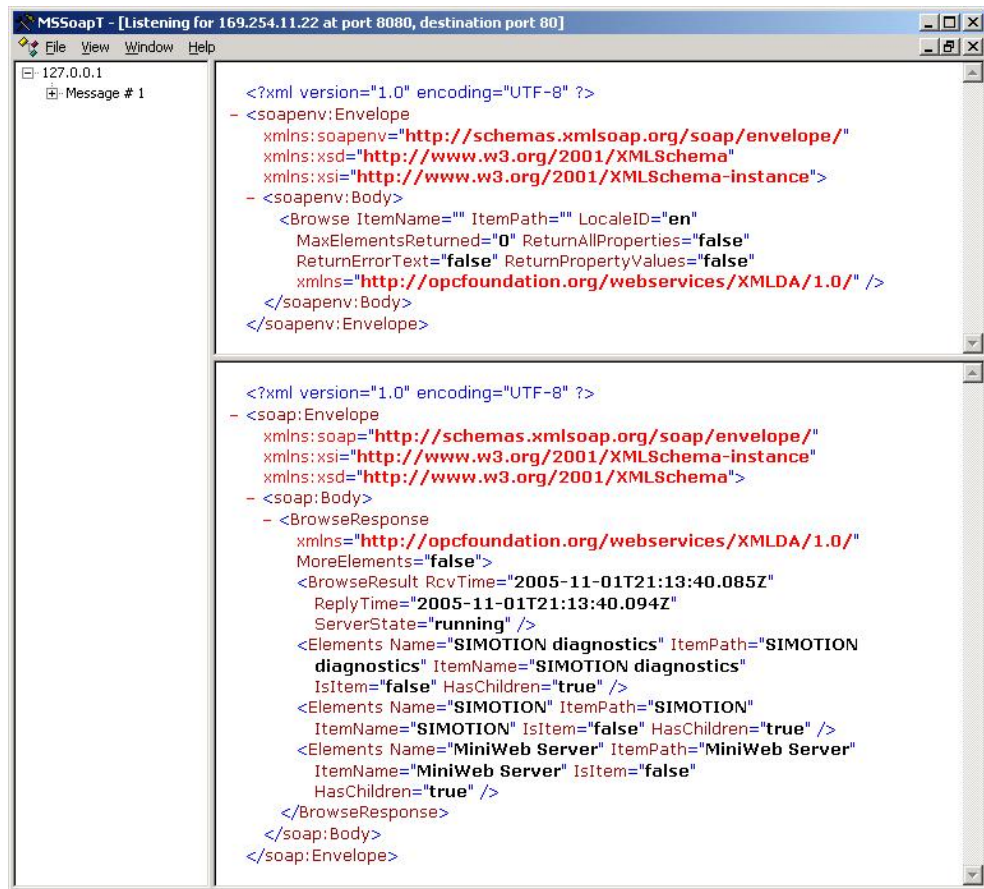
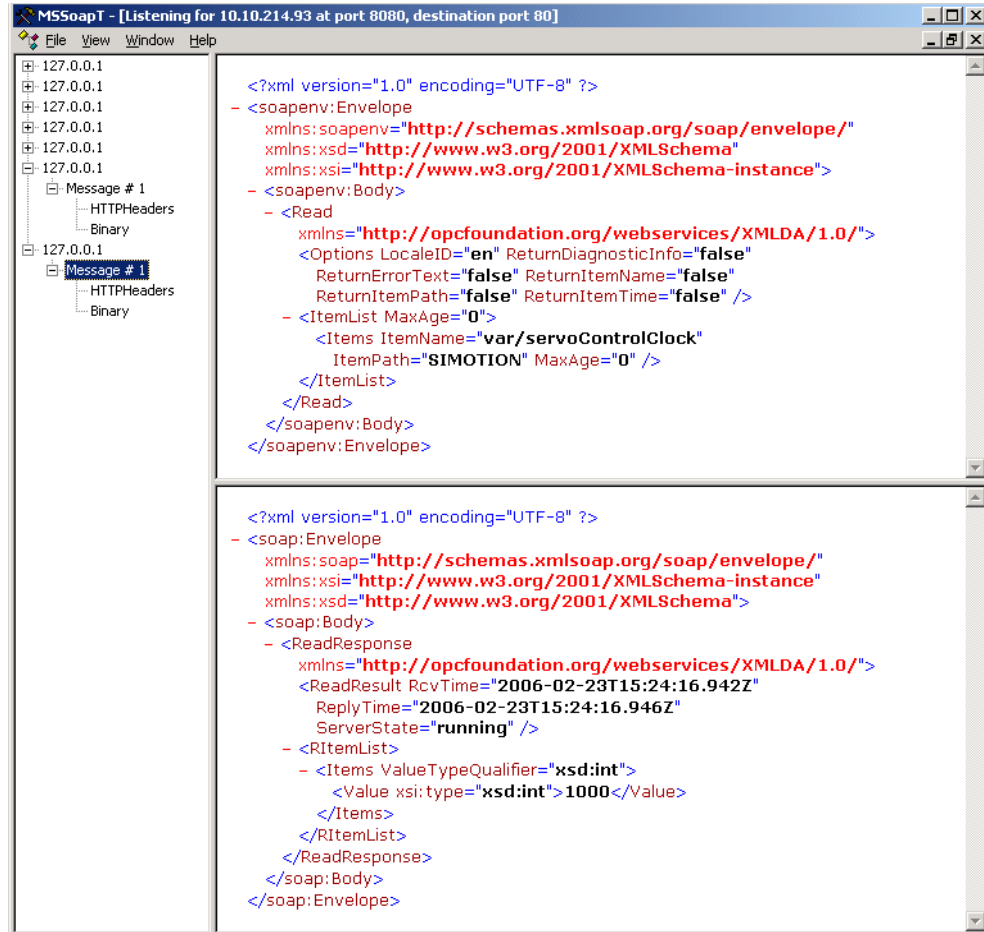


Figure 4-4 represents a read request to the server.

Data exchange via OPC XML

Figure 4-4: Read request of the variable var/servoControlClock including result 1000:





## 5 Appendix

### 5.1 Source texts and software documentation

You can find the Java source texts as well as a standard software documentation (JavaDoc, see <http://java.sun.com/j2se/javadoc/>) on the „Tools and Applications“ CD, which contains further additional applications / FAQ's on the subject SIMOTION.

### 5.2 Further information

For any further information on the subject OPC in general, we can recommend the website of the OPC foundation (<http://www.opcfoundation.org/>). This organization develops and maintains the OPC standard specifications.

For the subject of Eclipse, we recommend the internet page <http://www.eclipse.org/>. Here, you can find detailed information on the project Eclipse itself as well as on further projects concerning the Eclipse development environment.

### 5.3 List of literature and sources

Table 2-1: SIMOTION IT - Ethernet based HMI- and diagnostics functions, Siemens AG

Figure 3-5: „OPC XMLDA 1.01 Specification.pdf“, OPC Foundation