

SIEMENS

SIMOTION

SIMOTION IT SIMOTION IT Programming and Web Services

Programming Manual

Preface

Fundamental safety
instructions

1

Introduction

2

Software programming

3

Appendix

4


Valid as of Version 4.4


04/2014


Legal information

Warning notice system

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

 DANGER
indicates that death or severe personal injury will result if proper precautions are not taken.

 WARNING
indicates that death or severe personal injury may result if proper precautions are not taken.

 CAUTION
indicates that minor personal injury can result if proper precautions are not taken.

NOTICE
indicates that property damage can result if proper precautions are not taken.


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

Qualified Personnel

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

Proper use of Siemens products

Note the following:

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

Trademarks

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

Disclaimer of Liability

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

Preface

SIMOTION Documentation

An overview of the SIMOTION documentation can be found in the SIMOTION Documentation Overview document.

This documentation is included as electronic documentation in the scope of delivery of SIMOTION SCOUT. It comprises ten documentation packages.

The following documentation packages are available for SIMOTION V4.4:

- SIMOTION Engineering System Handling
- SIMOTION System and Function Descriptions
- SIMOTION Service and Diagnostics
- SIMOTION IT
- SIMOTION Programming
- SIMOTION Programming - References
- SIMOTION C
- SIMOTION P
- SIMOTION D
- SIMOTION Supplementary Documentation

Hotline and Internet addresses

Additional information

Click the following link to find information on the the following topics:

- Ordering documentation / overview of documentation
- Additional links to download documents
- Using documentation online (find and search manuals/information)

<http://www.siemens.com/motioncontrol/docu>

Please send any questions about the technical documentation (e.g. suggestions for improvement, corrections) to the following e-mail address:
docu.motioncontrol@siemens.com

My Documentation Manager

Click the following link for information on how to compile documentation individually on the basis of Siemens content and how to adapt it for the purpose of your own machine documentation:

<http://www.siemens.com/mdm>

Training

Click the following link for information on SITRAIN - Siemens training courses for automation products, systems and solutions:

<http://www.siemens.com/sitrain>

FAQs

Frequently Asked Questions can be found in SIMOTION Utilities & Applications, which are included in the scope of delivery of SIMOTION SCOUT, and in the Service&Support pages in **Product Support**:

<http://support.automation.siemens.com>

Technical support

Country-specific telephone numbers for technical support are provided on the Internet under **Contact**:

<http://www.siemens.com/automation/service&support>

Table of contents


	Preface	3
1	Fundamental safety instructions	9
1.1	General safety instructions.....	9
1.2	Industrial security.....	10
2	Introduction	11
2.1	Overview of SIMOTION IT.....	11
2.2	New features.....	12
3	Software programming	13
3.1	User-defined pages.....	13
3.1.1	User-defined Home page.....	13
3.1.2	Introduction.....	14
3.1.3	Loading of MWSL pages into the controller.....	14
3.1.4	Compiling MWSL files.....	15
3.1.5	Embedded, user-defined pages.....	19
3.1.6	Menu editor.....	20
3.1.7	JavaScript and web services.....	24
3.1.7.1	Variable access with JavaScript and web services.....	24
3.1.7.2	Communication with the OPC XML DA server (opcxml.js).....	24
3.1.7.3	Representation of OPC XML-DA data in the browser (appl.js).....	37
3.1.8	MiniWeb Server Language (MWSL).....	47
3.1.8.1	Mode of operation of the MWSL.....	47
3.1.8.2	Structure of a MWSL file.....	48
3.1.8.3	Error messages.....	48
3.1.8.4	Variable types.....	50
3.1.8.5	Script variables.....	50
3.1.8.6	Global variables.....	52
3.1.8.7	Special variables.....	53
3.1.8.8	Configuration constants.....	54
3.1.8.9	Variables and URL parameters.....	55
3.1.8.10	COOKIES.....	56
3.1.8.11	Variables and access to COOKIES.....	57
3.1.8.12	Variables and HTTP header information.....	57
3.1.8.13	Operators.....	58
3.1.8.14	Conditional operations.....	61
3.1.8.15	Loops.....	63
3.1.8.16	Functions.....	64
3.1.8.17	Comments.....	64
3.1.8.18	Overview of MWSL functions.....	65
3.1.8.19	Mode of operation of the template mechanism.....	66
3.1.8.20	Structure of the template file.....	67
3.1.8.21	Structure of a data source.....	68
3.1.8.22	Template transformation.....	69


3.1.8.23	MWSL in XML attributes.....	73
3.1.8.24	Examples.....	74
3.1.9	Server Side Includes (SSI).....	82
3.1.9.1	Integration of process values.....	82
3.2	OPC XML-DA web service.....	84
3.2.1	Web services introduction.....	84
3.2.2	Overview.....	84
3.2.3	Comparison of OPC XML DA / SIMATIC NET OPC DA.....	86
3.2.4	Schematic representation of creating the client application.....	87
3.2.5	Schematic representation at runtime of the client application.....	88
3.2.6	Installation.....	88
3.2.6.1	Hardware and software requirements for creating the client application.....	88
3.2.6.2	Configuring the SIMOTION device interface for using the client application.....	89
3.2.6.3	OPC XML DA access protection.....	89
3.2.7	OPC XML-DA variable access.....	90
3.2.8	Example of a client application.....	90
3.2.9	SIMOTION IT OPC XML-DA server interface.....	91
3.2.9.1	Overview.....	91
3.2.9.2	Methods which can be called synchronously.....	91
3.2.9.3	Access to variables.....	93
3.3	Trace Interface via SOAP (TVS) web service.....	94
3.3.1	Trace overview.....	94
3.3.2	Trace sequence.....	95
3.3.3	Procedure/terms.....	96
3.3.4	Error handling.....	97
3.3.5	Basics of subscriptions.....	97
3.3.6	Interface.....	99
3.3.6.1	Global definitions.....	99
3.3.6.2	Methods.....	101
3.3.6.3	Subscriptions.....	107
4	Appendix.....	109
4.1	MWSL functions.....	109
4.1.1	AddHTTPHeader.....	109
4.1.2	createGUID.....	109
4.1.3	DecodeString.....	110
4.1.4	die.....	110
4.1.5	EncodeString.....	111
4.1.6	ExistFile.....	111
4.1.7	ExistVariable.....	112
4.1.8	GetLanguage.....	112
4.1.9	GetVar.....	112
4.1.10	InsertFile.....	114
4.1.11	IsAuthAlgo.....	115
4.1.12	isFinite.....	115
4.1.13	isNaN.....	115
4.1.14	IsSSL.....	116
4.1.15	parseFloat.....	116
4.1.16	parseInt.....	116
4.1.17	ProcessXMLData.....	118
4.1.18	ReadFile.....	119

4.1.19	ReplaceString.....	120
4.1.20	SetVar.....	120
4.1.21	ShareRealm.....	120
4.1.22	write.....	122
4.1.23	WriteToTab.....	122
4.1.24	WriteVar.....	123
4.1.25	WriteXMLData.....	125
4.1.26	NodeIndex.....	126
4.1.27	NodeLevel.....	127
	Index.....	129

Fundamental safety instructions

1.1 General safety instructions

 WARNING
Risk of death if the safety instructions and remaining risks are not carefully observed
If the safety instructions and residual risks are not observed in the associated hardware documentation, accidents involving severe injuries or death can occur.
<ul style="list-style-type: none">• Observe the safety instructions given in the hardware documentation.• Consider the residual risks for the risk evaluation.

 WARNING
Danger to life or malfunctions of the machine as a result of incorrect or changed parameterization
As a result of incorrect or changed parameterization, machines can malfunction, which in turn can lead to injuries or death.
<ul style="list-style-type: none">• Protect the parameterization (parameter assignments) against unauthorized access.• Respond to possible malfunctions by applying suitable measures (e.g. EMERGENCY STOP or EMERGENCY OFF).

1.2 Industrial security

Note

Industrial security

Siemens provides products and solutions with industrial security functions that support the secure operation of plants, solutions, machines, equipment and/or networks. They are important components in a holistic industrial security concept. With this in mind, Siemens' products and solutions undergo continuous development. Siemens recommends strongly that you regularly check for product updates.

For the secure operation of Siemens products and solutions, it is necessary to take suitable preventive action (e.g. cell protection concept) and integrate each component into a holistic, state-of-the-art industrial security concept. Third-party products that may be in use should also be considered. For more information about industrial security, visit <http://www.siemens.com/industrialsecurity>.

To stay informed about product updates as they occur, sign up for a product-specific newsletter. For more information, visit <http://support.automation.siemens.com>



WARNING

Danger as a result of unsafe operating states resulting from software manipulation

Software manipulation (e.g. by viruses, Trojan horses, malware, worms) can cause unsafe operating states to develop in your installation which can lead to death, severe injuries and/or material damage.

- Keep the software up to date.
Information and newsletters can be found at:
<http://support.automation.siemens.com>
- Incorporate the automation and drive components into a state-of-the-art, integrated industrial security concept for the installation or machine.
For more detailed information, go to:
<http://www.siemens.com/industrialsecurity>
- Make sure that you include all installed products into the integrated industrial security concept.

Introduction

2.1 Overview of SIMOTION IT

Overview of SIMOTION IT manuals

The "SIMOTION IT Ethernet-based HMI and diagnostic functions" are described in three manuals (IT = Information Technology):

- **SIMOTION IT Diagnostics and Configuration**
This manual describes the direct diagnosis of SIMOTION devices. Access is by means of a standard browser (e.g. Firefox) via the IP address of the SIMOTION device. You can use the standard diagnostic pages or your own HTML pages for access.
See the manual SIMOTION IT Diagnostics and Configuration.
- **SIMOTION IT Programming and Web Services**
This manual describes the creation of user-defined web pages and access to the diagnostic functions via the two web services provided by SIMOTION IT.
A web service enables users to create their own client applications in any programming language. These applications then communicate with the SIMOTION device using web technologies. The SOAP (Simple Object Access Protocol) communication protocol is used for transmitting commands.
The manual includes information on programming such clients, as well as a description of the SIMOTION IT web services (OPC XML-DA, Trace via SOAP TVS) via which data and operating states of the controller can be accessed and the variable trace functions can be used.
- **SIMOTION IT Virtual Machine and Servlets**
This manual describes the Java-based function packages. The Jamaica Virtual Machine (JamaicaVM) is a runtime environment for Java applications on the SIMOTION device. It is an implementation of the "Java Virtual Machine Specification."
The Servlets section of the manual describes the use of servlets in a SIMOTION device.
See the manual SIMOTION IT Virtual Machine and Servlets.

See also

PDF in the Internet: SIMOTION IT Diagnostics and Configuration (<http://support.automation.siemens.com/WW/view/de/61148061/0/en>)

PDF in the Internet: SIMOTION IT Virtual Machine and Servlets (<http://support.automation.siemens.com/WW/view/de/61148107/0/en>)

2.2 New features

What new features does the current version offer?

Version 4.4

- New way of creating MWSL pages (Page 14). The MWSL files stored in HTML format on the control are compiled online. Offline compilation as with .mcs is no longer necessary.
- New MWSL functions (Page 65)
createGUID, die, DecodeString, EncodeString, ExistFile, GetLanguage, IsAuthAlgo, isFinite, isNaN, IsSSL, parseFloat, parseInt, ReadFile, ReplaceString, ShareRealm, WriteToTab
- New version of the MWSL server language
- New MWSL operators (Page 58)
- File extensions for web sites have been changed from .mcs/.mbs to .mws/.mws.cms.

Software programming

3.1 User-defined pages

3.1.1 User-defined Home page

You can create your own home page and display it instead of the home page for the standard diagnostic pages of the control. To do this, you need to change the default page of the web server in the WebCfg.xml file.

Procedure

1. Creating your own home page. Save this home page, for example, as `MyIndex.mwsl`.
2. Transmit the home page to the memory card of the SIMOTION device using the **Files** page.
3. Open the WebCfg.xml file in an available editor. The file can be found either on the supplied DVD in the 3_Configuration directory (in the default state) or on the SIMOTION device memory card (possibly in a modified state) in directory \USER\SIMOTION\HMICFG.
4. Replace the file name `index.mwsl` in the `<SERVEROPTIONS>` in element `<DEFAULTDOCUMENT VALUE="index.mwsl" />` with the name of your home page, including the path name "files" (all user-defined HTML pages are stored in the FILES directory).
Example: `<DEFAULTDOCUMENT VALUE="files/MyIndex.mwsl" />`
5. Save the changed WebCfg.xml on the memory card via the **Manage Config > WebCfg transmission** page.

3.1.2 Introduction

Individually designed web pages with access to device data

SIMOTION IT DIAG offers the option of storing machine-specific pages on the SIMOTION control. They are stored in a separate binary format (*.mws1.cms). The following resources can be used to integrate the process values of the SIMOTION control.

- **Server Side Includes (SSI):** Server-side, static, simplest, unformatted display (Page 82). In the HTML code of the page, process values can be integrated simply and without a representation option (e.g. number of decimal points) at any point.
- **MiniWeb Server Language (MWSL):** Server-side, static, formatted display using scripts in the SIMOTION control (Page 47). The generation of HTML code and the formatted integration of process values (e.g. in HEX or decimal value representation) can be selectively controlled with this script language.
- **OPC XML-DA web service:** Dynamic, formatted display using JavaScript in the browser (Page 24). The OPCXML.JS library offers elegant use of process values for JavaScript when dynamizing HTML pages.

The creation of user-defined pages requires knowledge of HTML and JavaScript programming.

Key words for advanced programming: XML, HTTP Request, Ajax, and web services.

The following figures shows an example of a simple user-defined page.

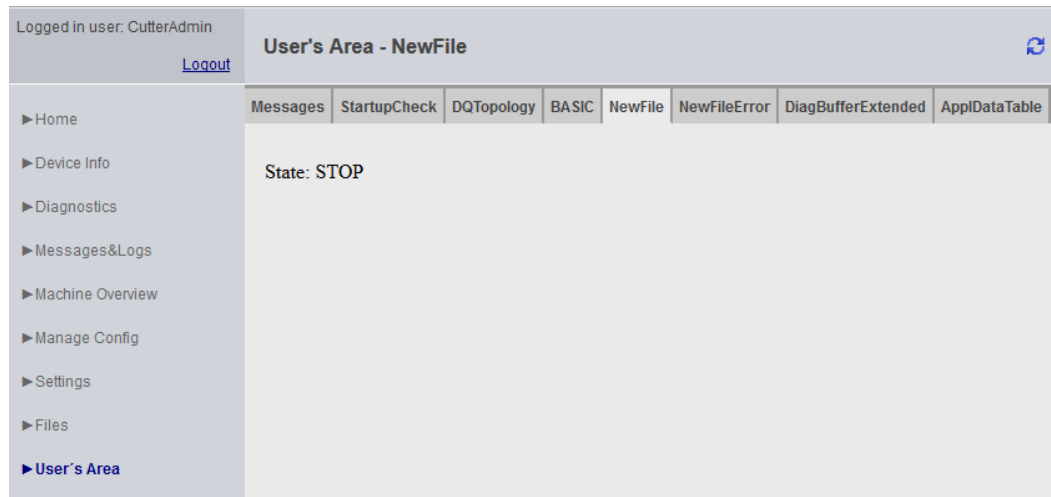


Figure 3-1 Example of a simple user-defined page

3.1.3 Loading of MWSL pages into the controller

The source code of an MWSL page must be created as a file with the extension .mws1. An MWSL file is an HTML file with MWSL extensions to be able to copy the control information.

The source code files is converted to the internal format of the controller in different ways:

- Load the files via the **Files > Files** page.
- Copy the files onto the card using a card reader.
- Load the files via FTP.

When loading via a card reader or FTP, the files must be stored in directory /USER/SIMOTION/HMI/FILES.

See also

Structure of a MWSL file (Page 48)

Embedded, user-defined pages (Page 19)

3.1.4 Compiling MWSL files

The .mwsl files have the same basic structure as standard HTML pages. The source code file should be coded with UTF-8 if special characters are to be displayed correctly.

1. Create the MWSL pages with a tool of your choice. The pages are assigned the file extension .mwsl
2. Compile the page. The converted files will appear in the target directory with the same name as the associated source files having the file extension .mwsl.cms
3. Check and test the page in the **User's Area**.
4. Errors that occur when compiling the MWSL file are output in the logfile. Errors that occur when the web page is called are written in the source text of the MWSL page as error messages.

The /FILES directory and all subdirectories are examined for the corresponding file types. The compiler replaces the original file by the compiled code. The compiled files have the extension *.cms

Compilation of the MWSL pages

There are three ways of compiling MWSL pages:

1. Click the **Compile** button. All files of the FILES directory are compiled. This option is recommended, for example, after an FTP upload.
2. When the controller boots. All files of the FILES directory are compiled.
3. Compile the first time the page is called via its URL. This is also the case when the users area is called.

Note

Backing up the originals

Because the .mwsf files are deleted during conversion, the originals should always be backed up on the PC and only a copy put on the card.

Successfully compiled pages with the .cms extension

The following example shows compilation via the **Files** page. On the **Files** page, you will find the Directory Operations with which the MWSL pages can be loaded into the controller via the **Send** button and compiled.

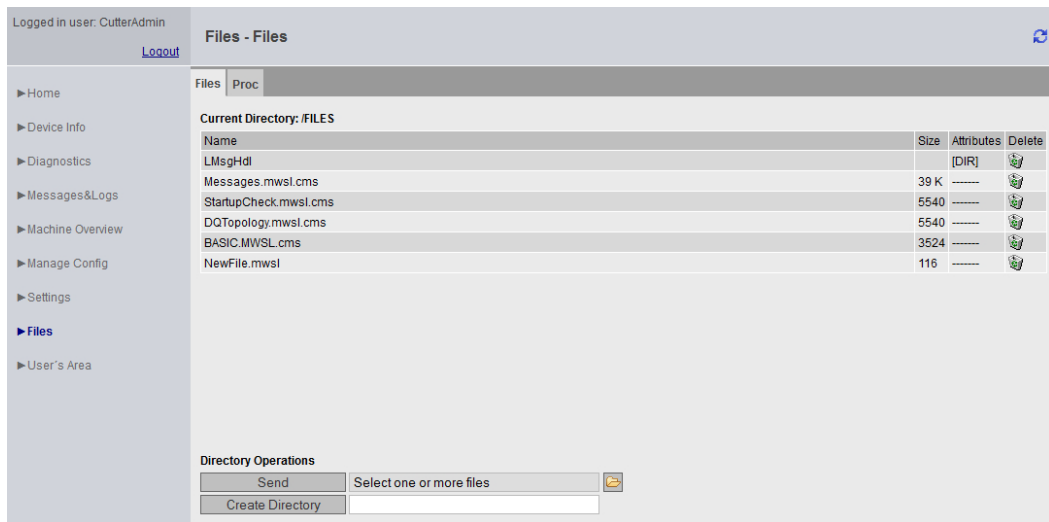


Figure 3-2 Loading of MWSL files

If a file has been loaded without error, it is given the extension .cms

Note

Files with errors do not have an .cms extension. In the above example, there is a problem with the file NewFile.mwsf

Example of compilation of a user-defined page

An MWSL page can be created using any text editor.

```
<html>
<head/>
<body>
<br>&nbsp; State: <MWSL>WriteVar("DeviceInfo.BZU");</MWSL>
</body>
```



```
</html>
```

NewFile.mwsl

In this example, the source text is saved as a file with the name `NewFile.mwsl`.

The `MWSL` is used in the example to output device data. Structure of a `MWSL` file (Page 48)

The device data is accessed in the `MWSL`-Ausdrücken by means of the variables provider. See Section Variables Provider in the SIMOTION IT Diagnostics and Configuration Manual.

The page can only be displayed if the `User's Area` settings are correct. In this example, the `EmbeddedSimple` version has been selected. Embedded, user-defined pages (Page 19)



Figure 3-3 User's Area Output `NewFile.mwsl.cms`

The page outputs can be checked in the `User's Area`. The screenshot shows the output of the sample file. If there are errors in compilation, the file is named `NewFile.mwsl.cms`

Error analysis

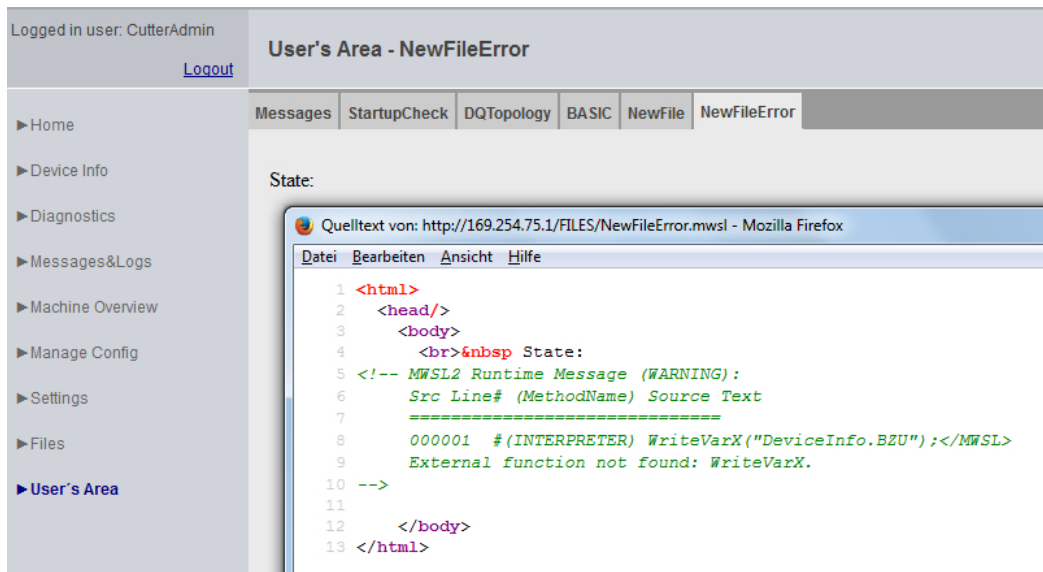


Figure 3-4 User's Area MWSL error message

Error messages that occur when the page is executing are inserted in the source text of the MWSL page as a comment. You can access the source text in the browser by displaying the source text of the current frame. In the example, an attempt is made to access the function `WriteVarX()`, which does not exist.

Errors occurring during compilation are output in a log file formed by appending ".log" to the file name.

Files in the old MBS and MCS format

Interpretation of MCS and MBS files (e.g. user-defined files) is still supported in Version 4.4 to ensure downward compatibility. A diagnostic buffer entry indicates that the format is obsolete. For future versions, this downward compatibility is not ensured. For this reason, old pages should be stored in the new format.

3.1.5 Embedded, user-defined pages

Integrating user-defined pages

As from Version 4.1.3, user-defined pages can be embedded in the framework of the SIMOTION IT DIAG standard pages.

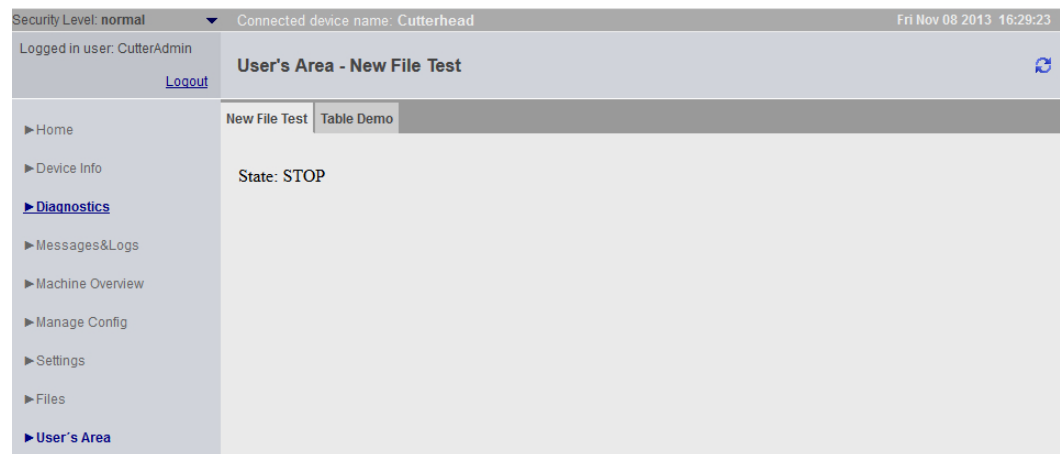


Figure 3-5 User's Area with embedded page

The menu in the **User's Area** links the files in the FILES folder in two different ways:

- EmbeddedSimple: The **User's Area** page loads all web pages contained in the FILES folder as tabs. The file name is shown without an extension.
- Embedded: The page loads a user-definable tab.

You can switch between EmbeddedSimple and Embedded on the **Settings** page. See the Manual SIMOTION IT Diagnostics and Configuration, Section Standard pages.

Settings in WebCfg.xml

In WebCfg.xml, the appearance of the User's Area can be set with the configuration constants `<UserArea>` and `<UserDir>`.

The `<UserArea>` tag can be used to set how the tab is displayed (the default setting is shown in bold):

```
<UserArea>( StandAlone | Embedded | EmbeddedSimple )</UserArea>
```

`<UserDir>` denotes the directory for the tab files relative to the FILE directory.

```
<UserDir></UserDir>
```

StandAlone

```
<UserArea>StandAlone</UserArea>
```

3.1 User-defined pages

Access to the **User's Area** is permanently linked to the user.mwsl file. To display the **User's Area** , this file must exist and be available for calling.

EmbeddedSimple

```
<UserArea>EmbeddedSimple</UserArea>
```

Select this option to use all the files found in the directory labeled <UserDir> to create the tab.

The respective file name (without the extension) is used as the title, and the corresponding menu link refers to this file.

Embedded - Using the menu editor

```
<UserArea>Embedded</UserArea>
```

If, on the **Settings** page, the checkbox **Enable user editor** has been selected, in the **User's Area** , the menu editor (Page 20) will be displayed in which menus can be customized.

WebCfg.xml example:

```
<SERVERPAGES version="78.00">
  [...]
  <CONFIGURATION_DATA>
    <USERCONFIG>
      <UserArea>Embedded</UserArea>
      <UserDir/>
    </USERCONFIG>
  </CONFIGURATION_DATA>
  [...]
</SERVERPAGES>
```

3.1.6 Menu editor

Creating individual menus using the menu editor

With Version 4.1.3 and higher, you can use the **Menu editor** link to call the menu editor, which enables you to configure custom menus for the User's Area.

Prerequisites for using the menu editor

To use the menu editor, in WebCfg.xml, the configuration constant `<UserArea>` must be set to `Embedded`. Alternately, on the **Settings** page, **User pages** can be set to **Embedded**.

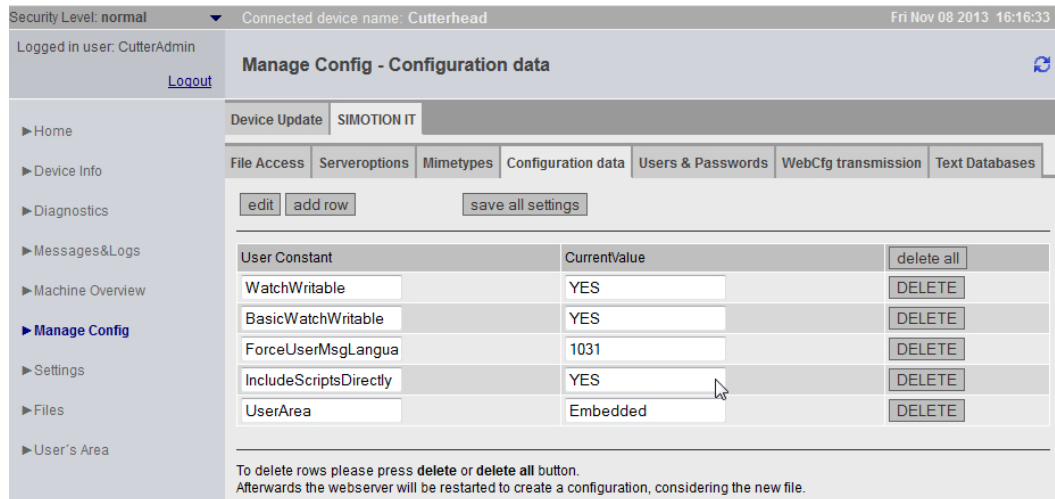


Figure 3-6 Configuration data menu editor

The **Enable user menu editor** option must then be selected on the **Settings** page.

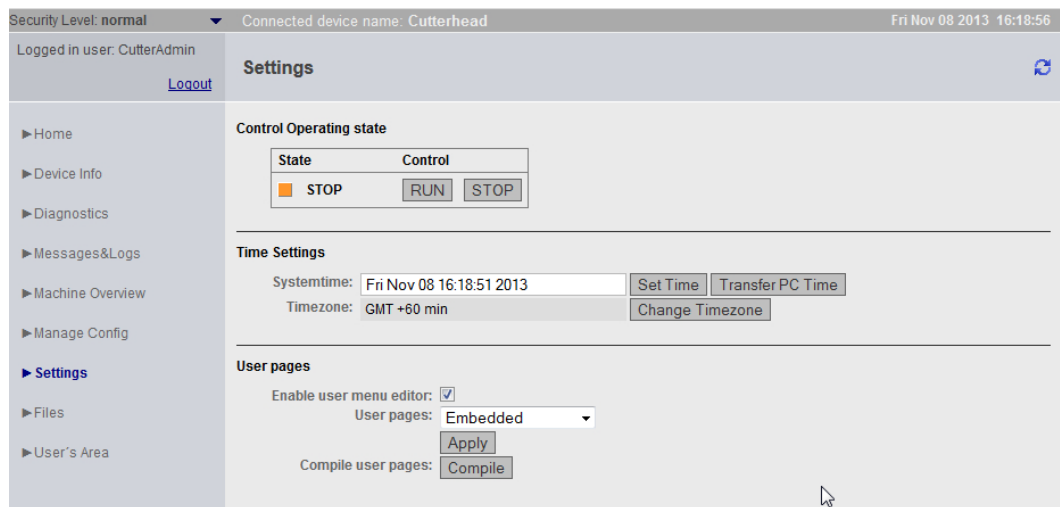


Figure 3-7 Settings menu editor

Working with the menu editor

The **User's Area** page now contains the **Menu editor** tab.

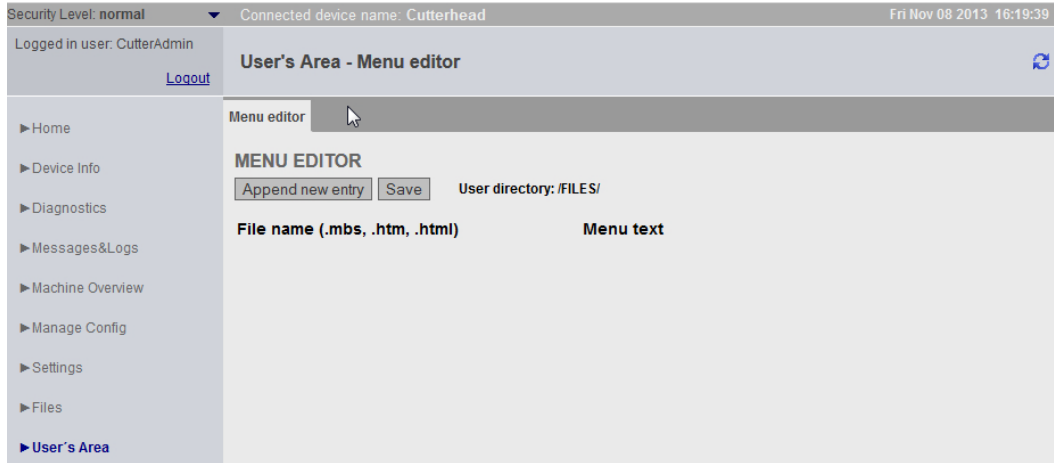


Figure 3-8 Starting the menu editor for the first time

The first time the menu editor is launched, a largely blank page appears.

New menu items can be created with the **Append new entry** button.

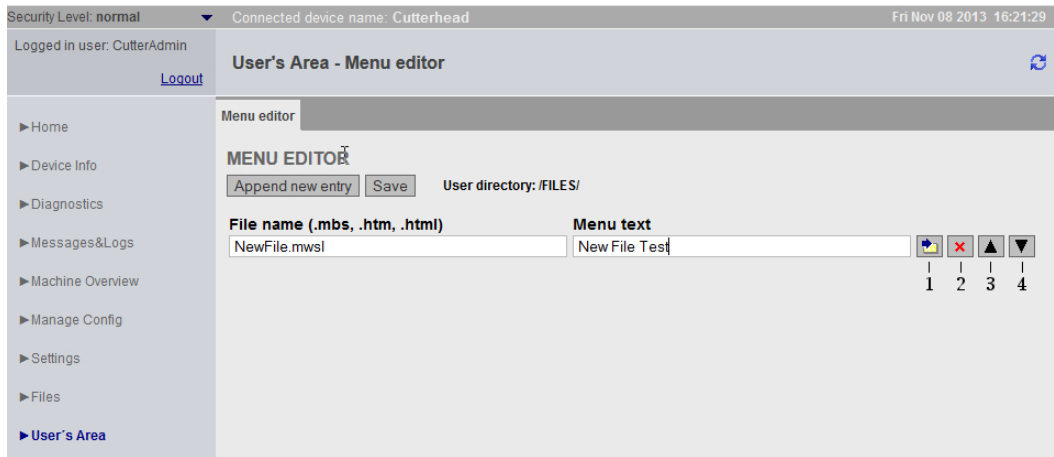


Figure 3-9 Menu editor with several items

In the screenshot above, the **NewFile.mwsl** file has been added. The buttons can be used to add or delete and change the position of files.

The names of the files to be displayed for the corresponding menu commands are entered in the **File name** column.

The **Menu text** column contains the name of the menu item.

Button ① is used to create new menu items; these are inserted before the current item in each case.

Button ② deletes the corresponding menu item.

Button ③ moves the menu item up.

Button ④ moves the menu item down.

Example

Two files are entered in the menu editor.

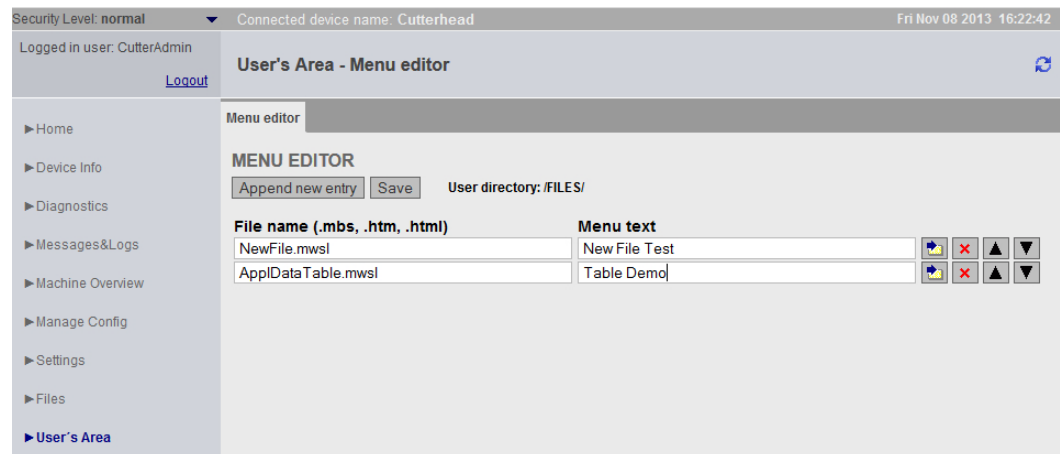


Figure 3-10 Example menu editor

The **User's Area** now only shows the selected pages.

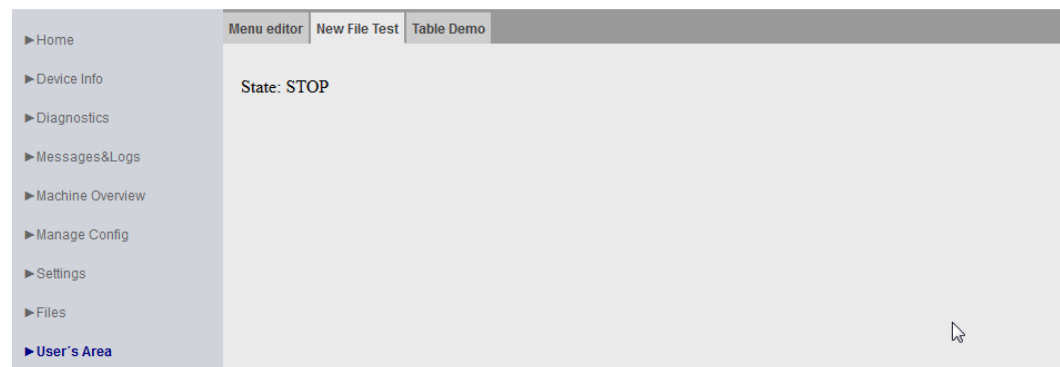


Figure 3-11 Example Menu Editor - display of Users's Area

To hide the tab **Menu editor** on the page, on the **Settings** page, the option **Enable user menu editor** must be deactivated.

3.1.7 JavaScript and web services

3.1.7.1 Variable access with JavaScript and web services

Access to a device with the JavaScript library

Using the DOM functionality of JavaScript, it is possible to implement simple web service clients. This opens up a multitude of new options within a browser, e.g.:

- Reading and cyclic updating of variable contents using an OPC XML-DA Read command
- Writing of variables using an OPC XML-DA Write command
- Browsing of the entire SIMOTION variable management area
- Setting up and querying an OPC XML-DA Subscription

The functionality is provided by several JavaScript files:

- `opcxml.js`: Contains functions for the structure of the necessary XML documents and for communication with an OPC XML-DA server.
- `apl.js`: Based on `opcxml.js` and implements the following objects:
 - Variable browser: Representation of the SIMOTION variable management area in a tree topology within the browser
 - Property viewer: Representation of variable properties (value, data type, access rights, Enums) in the form of a table within a browser. For writable variables, the table contains an entry field for changing the variable content.
 - Watch table: Representation of a watch table in the browser

3.1.7.2 Communication with the OPC XML DA server (`opcxml.js`)

OPCReadRequest

The `OPCReadRequest` class can be used to read the values for a list of variables.

```
function OPCReadRequest(parLocaleId,parResultCB)
```

Transfer parameters:

- `parLocaleId`: Language identifier ("DE", "EN")
- `parResultCB`: Callback function that must be provided by the caller
This function is called by `OPCReadRequest` when a response has arrived from the OPC XML DA server. The `OPCReadRequest` object is disposed of automatically (by calling the "destructor" method) if the callback function returns the value "true."

```
function OPCReadRequestCB (parResponse)
```

Transfer parameters:

- `parResponse`: Array of `ItemValues` with the result of the read request.

```
function OPCItemValue()
{
    this.mItemPath;
    this.mItemName;
    this.mItemHandle;
    this.mItemValue;
    this.mItemResultId;
}
```

If `mItemResultId` is defined, an error occurred during reading. In this case, `mItemResultId` is assigned the OPC XML DA error ID.

Example:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html;
        charset=ISO-8859-1">
    <script type="text/javascript" src="/common.js"></script>
    <script type="text/javascript" src="/opcxml.js"></script>
    <script type="text/javascript">
        function read()
        {
            var tmpReadCB = function(parResponse)
            {
                tmpResultStr = "";
                for (var tmpIndex = 0; tmpIndex < parResponse.length;
                    tmpIndex++)
                {
                    var tmpItemValue = parResponse[tmpIndex];
                    var tmpValue = (tmpItemValue.mItemValue) ?
                        tmpItemValue.mItemValue :
                        tmpItemValue.mItemResultId;
                    tmpResultStr += tmpItemValue.mItemPath
                        + " : "
                        + tmpItemValue.mItemName
                        + " = "
                        + tmpValue
                        + "\n";
                }
            }
        }
    </script>
</head>
</html>
```

```
        alert(tmpResultStr);
        return true;
    }
    var tmpReadRequest = new OPCReadRequest("DE",tmpReadCB);
    tmpReadRequest.addItem("SIMOTION","var/userdata.user1");
    tmpReadRequest.addItem("SIMOTION","var/userdata.user2");
    tmpReadRequest.addItem("SIMOTION","var/userdata.user10");
    tmpReadRequest.sendReadRequest();
}
</script>
<title>Insert title here</title>
</head>
<body>
    <input type="button" onclick="read();" value="Read"/>
</body>
</html>
```

OPCGetPropertiesRequest

The `OPCGetPropertiesRequest` class can be used to read the properties of variables:

- Values
- Data types
- Access rights
- Enum components

```
function OPCGetPropertiesRequest(parLocaleId,parResultCB)
```

Transfer parameters:

- `parLocaleId`: Language identifier ("DE", "EN")
- `parResultCB`: Callback function that must be provided by the caller. This function is called by `OPCGetPropertiesRequest` when a response has arrived from the OPC XML DA server. The `OPCGetPropertiesRequest` object is disposed of automatically (by calling the "destructor" method) if the callback function supplies "true" as a return value.

`function OPCGetPropertiesRequestCB(parResponse)`
`parResponse`: Array of `PropertyResults` that contain the properties of variables:

```
function OPCPropertyResult ()
{
    this.mItemPath;
    this.mItemName;
    this.mName;
    this.mIsItem;
    this.mHasChildren;
    this.mResultId;
    this.mValue;
    this.mType;
    this.mAccessRights;
    this.mEffectiveness;
    this.mEnums;
    this.mIsEnum;
}
```

User interface:

- `addItem(parItemPath, parItemName)` adds a variable to the variables list
- `removeItem(parItemHandle)` deletes a variable from the variables list
- `sendGetPropertiesRequest()` sends the read request
- `destructor()` releases the entire request object

Example:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html;
        charset=ISO-8859-1">
    <title>GetProperties</title>
    <script type="text/javascript" src="/common.js"></script>
    <script type="text/javascript" src="/opcxml.js"></script>
    <script type="text/javascript">
function getProperties()
{
    var tmpGetPropertiesCB = function(parResponse)
    {
        tmpResultStr = "";
        for (var tmpIndex = 0; tmpIndex < parResponse.length;
```

```

        tmpIndex++)
    {
        var tmpPropertyResult = parResponse[tmpIndex];
        var tmpEnums = "";
        if (tmpPropertyResult.mEnums &&
            (tmpPropertyResult.mEnums.length > 0))
        {
            for (var tmpIndex = 0;
                tmpIndex < tmpPropertyResult.mEnums.length;
                tmpIndex++)
            {
                tmpEnums += " " +
                    tmpPropertyResult.mEnums[tmpIndex] + "\n";
            }
        }
        if (!tmpPropertyResult.mResultId)
        {
            tmpResultStr += tmpPropertyResult.mItemPath +
                "::~" +
                tmpPropertyResult.mItemName +
                ":\n Type = " +
                tmpPropertyResult.mType +
                "\n value = " +
                tmpPropertyResult.mValue +
                "\n AccessRights = " +
                tmpPropertyResult.mAccessRights +
                "\n Enums = \n" + tmpEnums + "\n";
        }
        else
        {
            tmpResultStr += tmpPropertyResult.mItemPath +
                "::~" + tmpPropertyResult.mItemName
                + ":\n ResultId = " +
                tmpPropertyResult.mResultId +
                "\n\n";
        }
    }
    alert(tmpResultStr);

    return true;
}
var tmpGetPropertiesRequest =
    new OPCGetPropertiesRequest("DE", tmpGetPropertiesCB);
tmpGetPropertiesRequest.addItem("SIMOTION",
    "var/userdata.user1");
tmpGetPropertiesRequest.addItem("SIMOTION",
    "var/userdata.user20");
tmpGetPropertiesRequest.addItem("SIMOTION",
    "dev/Service.BZU.value");
tmpGetPropertiesRequest.sendGetPropertiesRequest();
}
</script>
</head>
<body>
    <input type="button" onclick="getProperties();"
        value="getProperties"/>
</body>

```

```
</html>
```

OPCWriteRequest

OPCWriteRequest writes three values of one or more variables.

```
function OPCWriteRequest(parLocaleId,parResultCB)
```

Transfer parameters:

- `parLocaleId`: Language identifier ("DE", "EN")
- `parResultCB`: Callback function that must be provided by the caller. The function is called on conclusion of the send request.

```
function OPCWriteRequestCB(parResultList)
```

`parResultList` is an array of `ItemValues` that contains the results of the write request.

```
function OPCItemValue()
{
    mItemPath
    mItemName
    mItemHandle
    mItemValue
    mItemResultId
}
```

The OPCWriteRequest object is disposed of automatically (by calling the "destructor" method) if the callback function returns the value "true."

User interface:

- `addItem(parItemPath,parItemName,parType)` adds a variable to the variables list and returns a variable handle, which can be used to reference the variable within the request. `ParType` designates the OPC XML DA data type to be used for writing. If `parType` is not passed, the "xsi:string" data type is applied.
- `removeItem(parItemHandle)` removes a variable from the variables list
- `setItemValue(parItemHandle,parValue)` sets the value that is to be written for a variable
- `sendWriteRequest()` sends the write request
- `destructor()` releases all resources occupied by the write object

Example:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
      charset=ISO-8859-1">
    <script type="text/javascript" src="/common.js"></script>
    <script type="text/javascript" src="/opcxml.js"></script>
```

```

<script type="text/javascript">
function writeValues()
{
    var tmpWriteCB = function(parWriteResult)
    {
        var tmpString = "";
        for (var tmpIndex = 0; tmpIndex < parWriteResult.length;
            tmpIndex++)
        {
            var tmpItemValue = parWriteResult[tmpIndex];
            var tmpValue = (tmpItemValue.mItemResultId) ?
                tmpItemValue.mItemResultId : tmpItemValue.mItemValue;
            tmpString += tmpItemValue.mItemPath + ":@" +
                tmpItemValue.mItemName + " = " + tmpValue + "\n";
        }
        alert(tmpString);
    }
    var tmpWrite = new OPCWriteRequest("DE",tmpWriteCB);
    var tmpItemHandle = tmpWrite.addItem("SIMOTION",
        "var/userdata.user1");
    tmpWrite.setItemValue(tmpItemHandle,"123");
    tmpItemHandle = tmpWrite.addItem("SIMOTION",
        "var/userdata.user2");
    tmpWrite.setItemValue(tmpItemHandle,"234");
    tmpItemHandle = tmpWrite.addItem("SIMOTION",
        "var/userdata.user10");
    tmpWrite.setItemValue(tmpItemHandle,"345");
    tmpWrite.sendWriteRequest();
}
</script>
<title>Write</title>
</head>
<body>
<input type="button" value="Write" onclick="writeValues()"/>
</body>
</html>

```

OPCBrowseRequest

The `OPCBrowseRequest` class can be used to browse the variable management area of a control.

```
function OPCBrowseRequest(parclaaLocaleId,parResultCB)
```

Transfer parameters:

- `parLocaleId`: Language identifier ("DE", "EN")
- `parResultCB`: Callback function that must be provided by the caller
The function is called on conclusion of the send request.

```
function OPCBrowseRequestCB (parResult, parItemPath, parItemName)
```

`parResult` is an array of type `BrowseResult` and contains the Browse information.

```
function BrowseResult ()
{
    mItemPath;
    mItemName;
    mName;
    mIsItem;
    mHasChildren;
}
```

`parItemPath` and `parItemName` are the path and name of the directory whose content is displayed in `BrowseResult`.

User interface:

- `sendBrowseRequest (parItemPath, parItemName)` sends the Browse request.
`parItemPath` and `parItemName` are the path and name of the directory to be browsed.

Example:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://
www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
      charset=ISO-8859-1">
    <script type="text/javascript" src="/common.js"></script>
    <script type="text/javascript" src="/opcxml.js"></script>
    <script type="text/javascript">
      function browse()
      {
        var tmpBrowseRequestCB = function(parBrowseResult,
          parItemPath,
          parItemName)
        {
          var tmpString = parItemPath + ":@" + parItemName + "\n";
          for (var tmpIndex = 0; tmpIndex < parBrowseResult.length;
            tmpIndex++)
          {
            var tmpBrowseResult = parBrowseResult[tmpIndex];
            tmpString += tmpBrowseResult.mItemName + "\n";
          }
          alert(tmpString);
        }
        var tmpBrowseRequest =
          new OPCBrowseRequest ("DE", tmpBrowseRequestCB);
        tmpBrowseRequest.sendBrowseRequest ("SIMOTION", "var/");
      }
    }
  }
</html>
```

```
        </script>
        <title>Browse</title>
    </head>
    <body>
        <input type="button" value="Browse" onclick="browse();" />
    </body>
</html>
```

OPCSubscriptionRequest

The `OPCSubscriptionRequest` class can be used to set up, poll, and delete an OPC XML DA subscription.

```
function OPCSubscriptionRequest(parLocaleId,parResultCB,parCancelCB)
```

Transfer parameters:

- `parLocaleId` Language identifier ("DE", "EN").call
- `parResultCB` Callback function that must be provided by the user. This callback function is called following setup and a refresh action.

```
function OPCSubscriptionRequestCB(parResultList,parResult)
parResultList is an array of type OPCItemValue, which contains the variable values
which have been determined.
```

```
function OPCItemValue()
{
    mItemPath
    mItemName
    mItemHandle
    mItemValue
    mItemResultId
}
```

- `parCancelCB` Bback function that must be provided by the user. This callback function is called after a subscription is released.

```
function OPCSubscriptionCancelCB()
The function has no transfer parameters to be passed.
```

User interface:

- `addItem(parItemPath,parItemName)` adds the passed variable to the internal list of subscription variables
- `removeItem(parItemHandle)` deletes the passed variable from the list of subscription variables.
- `cancel()` logs out an active subscription on the server

- `refresh()` reads the current variable values.
Only variables whose values have changed since the preceding query are passed. The first call of `refresh` after generating the OPCSubscription object or the first call after a cancel causes the subscription to log in with the current internal variable list on the server. Refresh must be called cyclically. The hold time mechanism of the OPC XML DA subscription is not supported, i.e. a wait time must be programmed between each 2 refresh cycles by means of a JavaScript timer. However, the wait time of the OPC XML DA subscription is active, i.e. a response to a refresh call is sent only after the wait time elapses, provided a variable value has not changed in the meantime.
- `destructor()`
Logs out the subscription on the server and releases the resources occupied by the subscription object. Destructor must be called before releasing the object.

Example:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://
www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
      charset=ISO-8859-1">
    <script type="text/javascript" src="/common.js"></script>
    <script type="text/javascript" src="/opcxml.js"></script>
    <script type="text/javascript">
      var gloSubscription;
      var gloItemHandle_1;
      var gloItemHandle_2;
      function subscription()
      {
        if (!gloSubscription)
        {
          var tmpSubscriptionCB = function(parValues)
          {
            for (var tmpIndex = 0;
              tmpIndex < parValues.length;
              tmpIndex++)
            {
              var tmpItemHandle =
                parValues[tmpIndex].mItemHandle;
              var tmpItemValue =
                parValues[tmpIndex].mItemValue;
              if (tmpItemHandle == gloItemHandle_1)
              {
                var tmpValueNode =
                  document.getElementById("user1");
                tmpValueNode.firstChild.nodeValue =
                  tmpItemValue;
              }
              else if (tmpItemHandle == gloItemHandle_2)
              {
                var tmpValueNode =
                  document.getElementById("user2");
                tmpValueNode.firstChild.nodeValue =
                  tmpItemValue;
              }
            }
          }
        }
      }
    }
  }

```

```
        var tmpTimerCB = function()
        {
            gloSubscription.refresh();
        }
        setTimeout(tmpTimerCB,300);
    }
    var tmpCancelCB = function()
    {
        if (gloSubscription)
        {
            gloSubscription.destructor();
            gloSubscription = null;
        }
    }
    gloSubscription =
        new OPCSubscriptionRequest("DE",
            tmpSubscriptionCB,tmpCancelCB);
    gloItemHandle_1 =
        gloSubscription.addItem("SIMOTION",
            "var/userdata.user1");
    gloItemHandle_2 =
        gloSubscription.addItem("SIMOTION",
            "var/userdata.user2");
    gloSubscription.refresh();
}
}
function cancel()
{
    if (gloSubscription)
        gloSubscription.cancel();
}
</script>
<title>Subscription</title>
</head>
<body>
<div>
<input type="button" value="Start"
    onclick="subscription();" />
<input type="button" value="Cancel" onclick="cancel();" />
</div>
<table>
<tr>
<td>user1</td>
<td id="user1">user1</td>
</tr>
<tr>
<td>user2</td>
<td id="user2">user2</td>
</tr>
</table>
</body>
</html>
```

OPCSubscriptionAutoRefresh

OPCSubscriptionAutoRefresh provides the same function as OPCSubscriptionRequest, with the exception that the timer-controlled call of the refresh function occurs automatically.

```
function
OPCSubscriptionAutoRefresh(parLocaleId,parResultCB,parCancelCB,
    parCycleTime)
```

Transfer parameters:

- parLocaleId
- parResultCB
- parCancelCB
See OPCSubscriptionRequest
- parCycleTime
Cycle time in which the refresh function is called, in ms

User interface:

- startRefresh() starts the query cycle
- cancel() See OPCSubscriptionRequest
- addItem(parItemPath,parItemName,parItemHandle) See OPCSubscriptionRequest
After the variable is added, the refresh cycle is restarted automatically.
- removeItem(parItemHandle)
- destructor()

Example:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html;
        charset=ISO-8859-1">
    <script type="text/javascript" src="/common.js"></script>
    <script type="text/javascript" src="/opcxml.js"></script>
    <script type="text/javascript">
        var gloSubscription;
        var gloItemHandle_1;
        var gloItemHandle_2;
        var gloItemHandle_3;
        function subscription()
        {
            if (!gloSubscription)
            {
                var tmpSubscriptionCB = function(parValues)
```

```
{
  for (var tmpIndex = 0; tmpIndex < parValues.length;
      tmpIndex++)
  {
    var tmpItemHandle =
      parValues[tmpIndex].mItemHandle;
    var tmpItemValue =
      parValues[tmpIndex].mItemValue;
    if (tmpItemHandle == gloItemHandle_1)
    {
      var tmpValueNode =
        document.getElementById("user1");
      tmpValueNode.firstChild.nodeValue =
        tmpItemValue;
    }
    else if (tmpItemHandle == gloItemHandle_2)
    {
      var tmpValueNode =
        document.getElementById("user2");
      tmpValueNode.firstChild.nodeValue =
        tmpItemValue;
    }
    else if (tmpItemHandle == gloItemHandle_3)
    {
      var tmpValueNode =
        document.getElementById("user3");
      tmpValueNode.firstChild.nodeValue =
        tmpItemValue;
    }
  }
}
var tmpCancelCB = function()
{
  if (gloSubscription)
  {
    gloSubscription.destructor();
    gloSubscription = null;
  }
}
gloSubscription =
  new OPCSubscriptionAutoRefresh("DE",
    tmpSubscriptionCB, tmpCancelCB, 300);
gloItemHandle_1 =
  gloSubscription.addItem("SIMOTION",
    "var/userdata.user1");
gloItemHandle_2 =
  gloSubscription.addItem("SIMOTION",
    "var/userdata.user2");
gloSubscription.startRefresh();
}
function addVar()
{
  if (gloSubscription)
  {
    gloItemHandle_3 =
      gloSubscription.addItem("SIMOTION",
```

```

        "var/userdata.user3");
    }
}
function removeVar()
{
    if (gloSubscription)
    {
        gloSubscription.removeItem(gloItemHandle_3);
    }
}
function cancel()
{
    if (gloSubscription)
        gloSubscription.cancel();
}
</script>
<title>Auto refresh</title>
</head>
<body>
<div>
    <input type="button" value="Start"
        onclick="subscription();"/>
    <input type="button" value="Add variable"
        onclick="addVar();"/>
    <input type="button" value="Remove variable"
        onclick="removeVar();"/>
    <input type="button" value="Cancel" onclick="cancel();"/>
</div>
<table>
    <tr>
        <td>user1</td>
        <td id="user1">user1</td>
    </tr>
    <tr>
        <td>user2</td>
        <td id="user2">user2</td>
    </tr>
    <tr>
        <td>user3</td>
        <td id="user3">user3</td>
    </tr>
</table>
</body>
</html>

```

3.1.7.3 Representation of OPC XML-DA data in the browser (appl.js)

appl.js

The JavaScript library **appl.js** assembles classes for the representation of the data determined with OPC XML-DA requests.

AppIDataTable

AppIDataTable implements a dynamic table in which process variables can be displayed. The variable values are updated cyclically with an OPC XML-DA subscription.

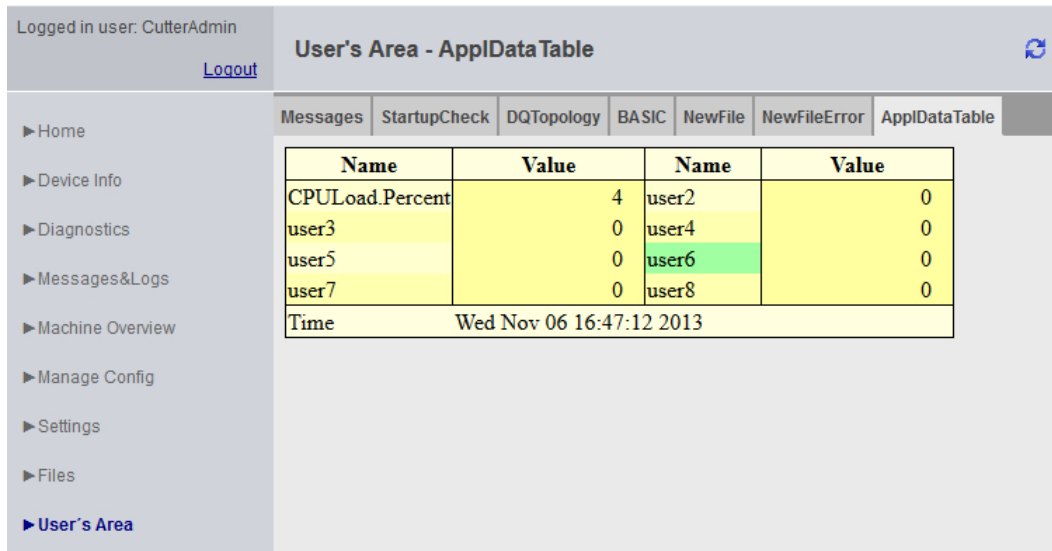


Figure 3-12 Example of implementing a dynamic table (AppIDataTable)

```
function
AppIDataTable (parDocument, parClassName, parColumnClasses, parColumnIds
, parHeader)
```

Transfer parameters:

- parDocument: JavaScript document used to generate elements
- parClassName: Entered as "class" attribute in the "Table" tag of the HTML table
- parColumnClasses: Array whose "length" attribute specifies the number of table columns. The values of the array are used as a "class" attribute for the table columns (<td class="...">).
- parColumnIds : Array whose values are used together with the parRowId parameter (described below) for the "id" attributes of the table columns. The value of the "id" attribute is produced by chaining together the ColumnId and RowId (in that order).
- parHeader: Array containing the column headings of the table

The transfer parameters passed are used to set the "class" and "id" attributes such that the table display can be specified using style sheets.

User interface:

The user interface of class `ApplDataTable` consists of a series of functions that are useful for general use in the web site and their passed and returned parameters.

- `addRow (parRowId, parRowClass)` appends a new row to the table
 - `parRowId`: Used as a value for the `id` attribute of row (`<tr id="...">`)
 - `parRowClass`: Used as a value for the `"class"` attribute of row (`<tr class="...">`)
- `addElement (parElement, parDestructor, parColSpan, parColClass)`
Inserts the HTML element passed using `parElement` in the table.
 - `parElement`: HTML element to be inserted in the table
 - `parDestructor` (optional)
Function that is called if the HTML element is deleted from the table (used in Internet Explorer to prevent memory leaks).
 - `parColSpan` (optional)
: Specifies the number of columns the element is to span.
 - `parColClass` (optional)
`parColClass` can be used to overwrite the `ColClass` assigned when the `ApplDataTable` object is created, if special formatting is to be applied for the current element.
- `addVariable (parPath, parName, parColSpan, parColClass)`
: Inserts a variable in the table. The value of the variables is updated cyclically.
 - `parPath`
Variable path (e.g. "SIMOTION" or "SIMOTION diagnostics")
 - `parName`
Variable name (e.g. "var/userdata.user1")
 - `parColSpan` (optional)
See `addElement`
 - `parColClass` (optional)
See `addElement`
- `addText (parText, parColSpan, parColClass)`: Inserts text in the table.
 - `parText`
: Text to be inserted.
 - `parColSpan` (optional)
See `addElement`
 - `parColClass` (optional)
See `addElement`

- `addRemoveButton (parRemoveCB, parRemoveData, parImage)`
Inserts a button in the table for removing the current row.
 - `parRemoveCB (parData)` (optional)
Transfer parameter passed to the Callback function. Data passed when `addRemoveButton` is called
 - `parRemoveData` (optional)
Data passed as a parameter when `parRemoveCB` is called.
 - `parImage`: (optional)
Optional parameters: DOM object of an HTML `img` element (e.g. created with `document.createElement("img")`), which is to appear on the button
- `addImage (parImage)`
Inserts an image in the table.
 - `parImage`: URL of the image to be inserted
- `getVariables ()` supplies an array of all variables of the table
Each entry of the array consists of an object with the elements `mItemPath`, `mItemName`, and `mItemHandle`.
- `addRefreshCB (parRefreshCB)` registers a Callback function on the table object, which is called on completion of a refresh cycle; an array of `OPCItemValue`-Objekten is passed to the Callback function.

```
OPCItemValue
{
```

```
    mItemPath
    mItemName
    mItemHandle
    mItemValue
    mItemResultId
```

```
}destructor () must be called if the table is no longer required.
```

This function releases all the resources occupied by the table. In particular, the subscription used by the table is logged off on the OPC XML DA server. The function must also be called when the page is exited (onunload).

Example:

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
      charset=ISO-8859-1">
    <script type="text/javascript" src="/common.js"></script>
    <script type="text/javascript" src="/opcxml.js"></script>
    <script type="text/javascript" src="/appl.js"></script>
    <script type="text/javascript">
      var gloApplDataTable = null;
      function init()
      {
        if (gloApplDataTable == null)
        {
          var tmpColumnIds      = new Array("Name_0", "Value_0",
                                             "Name_1", "Value_1");
          var tmpColumnClasses = new Array("Name", "Value",
                                             "Name", "Value")
```



```

var tmpHeader          = new Array("Name", "Value",
                                   "Name", "Value");
gloApplDataTable = new ApplDataTable(document,
                                     "ReadTableClass",
                                     tmpColumnClasses,
                                     tmpColumnIds,
                                     tmpHeader);
gloApplDataTable.addRow("Row_0", "RowClass_0");
gloApplDataTable.addText("CPULoad.Percent");
gloApplDataTable.addVariable("SIMOTION diagnostics",
                              "CPULoad.Percent");
gloApplDataTable.addText("user2");
gloApplDataTable.addVariable("SIMOTION",
                              "var/userdata.user2");
gloApplDataTable.addRow("Row_1", "RowClass_1");
gloApplDataTable.addText("user3");
gloApplDataTable.addVariable("SIMOTION",
                              "var/userdata.user3");
gloApplDataTable.addText("user4");
gloApplDataTable.addVariable("SIMOTION",
                              "var/userdata.user4");
gloApplDataTable.addRow("Row_2", "RowClass_0");
gloApplDataTable.addText("user5");
gloApplDataTable.addVariable("SIMOTION",
                              "var/userdata.user5");
gloApplDataTable.addText("user6");
gloApplDataTable.addVariable("SIMOTION",
                              "var/userdata.user6");
gloApplDataTable.addRow("Row_3", "RowClass_1");
gloApplDataTable.addText("user7");
gloApplDataTable.addVariable("SIMOTION",
                              "var/userdata.user7");
gloApplDataTable.addText("user8");
gloApplDataTable.addVariable("SIMOTION",
                              "var/userdata.user8");
gloApplDataTable.addRow("Time", "Time");
gloApplDataTable.addText("Time", 1, "Time");
gloApplDataTable.addVariable("SIMOTION diagnostics",
                              "DeviceInfo.Systemtime",
                              3, "Time");

var tmpTableRoot =
    document.getElementById("TableRoot");
if (tmpTableRoot)
{
    tmpTableRoot.appendChild(gloApplDataTable.mTable);
}
}
function close()
{
    if (gloApplDataTable != null)
    {
        gloApplDataTable.destructor();
    }
}
</script>

```

```

<style type="text/css">
  table.ReadTableClass {background-color:#FFFFE0;
                        border:1px solid black;
                        border-collapse:collapse;}
  th.Name {border:1px solid black;}
  th.Value {border:1px solid black;}
  td.Name {width:80px;border-right:1px solid black;}
  td.Value {background-color:#FFFFA0;width:120px;
           text-align:right;
           border-right:1px solid black;padding-right:15px;}
  td.Input {width:200px;}
  td.Time {border-top:1px solid black;}
  tr.RowClass_0 {background-color:#FFFFD0;border:0px;}
  tr.RowClass_1 {background-color:#FFFFB0;border:0px;}
  #Name_1Row_2 {background-color:#A0FFA0;}
</style>
<title>Table</title>
</head>
<body onload="init();" onunload="close();">
  <div id="TableRoot"></div>
</body>
</html>

```

In the final section of the source code, the `<style>` tag demonstrates how the colors of table rows and individual table cells can be changed using CSS formatting.

This means that the `tr.RowClass_0` declaration, for example, can ensure that a yellow background is produced when `gloAppDataTable.addRow("Row_0","RowClass_0").` is called.

ApplBrowser

`ApplBrowser` enables the variable management area of the control to be queried.

After you have created an object of this type, the 1st browse operation is started automatically. If the browse information of a subtree has been received, the callback function `parNewTreeFct` is first called. Afterwards, the callback function `parNewNodeFct` or `parNewLeafFct` is called, depending on the type of information (node or leaf). All callback functions receive an object of type `ApplBrowseElement` as the first parameter.

User interface `ApplBrowseElement`:

- `getElement()` supplies an HTML anchor object (`<a ...>`) for displaying information
- `setCB(parCB)`
: Registers a Callback function at the `ApplBrowseElement`-Objekt; this function is called if a user clicks the anchor object This function also receives an `ApplBrowseElement`-Objekt as a passed transfer parameter.
- `destructor()` must be called if the object is no longer required. This also applies to exiting the page (`onunload`).

```

ApplBrowser(parDocument,
           parItemPath,
           parItemName,
           parNewTreeFct,

```

```
parNewNodeFct,
parNewLeafFct)
```

Transfer parameters:

- `parDocument`: JavaScript document used to generate elements
- `parItemPath`, `parItemName` specifies the starting point for browsing the variable management area
- `parNewTreeFct (parBackElement, parItemPath, parItemName)`: Callback function that is called when the structure of a new subtree begins.
 - `parBackElement` object of type `ApplBrowseElement`
The content describes the start node. If a user clicks the HTML anchor of this object, the elements of the preceding subtree (if present) are determined.
 - `parItemPath`, `parItemName`: Path and name of the current subtree
- `parNewNodeFct (parBrowseElement)`: Callback function that is called for a node element
 - `parBrowseElement` object of type `ApplBrowseElement`
The content describes a node. If a user clicks the HTML anchor of this object, a browse operation is started for the subtree connected to this node.
- `parNewLeafFct (parBrowseElement)` Callback function that is called for a leaf element
 - `parBrowseElement` object of type `ApplBrowseElement`
The content describes a leaf.

User interface:

- `destructor` releases all resources occupied by the browser

Example:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/
html4/strict.dtd">
<html>
  <head>
    <title>Browser demo</title>
    <script type="text/javascript" src="/common.js"></script>
    <script type="text/javascript" src="/opcxml.js"></script>
    <script type="text/javascript" src="/appl.js"></script>
    <script type="text/javascript">
      var gloBrowseTable = null;
      var gloBrowser = null;
      function addItem(parBrowseElement)
      {
        var tmpBodyElement = gloBrowseTable.firstChild;
        var tmpTableRowElement = document.createElement("tr");
        var tmpTableDataElement = document.createElement("td");
        var tmpLinkElement = parBrowseElement.getElement();
        tmpTableDataElement.appendChild(parBrowseElement.getElement());
        tmpTableRowElement.appendChild(tmpTableDataElement);
        tmpBodyElement.appendChild(tmpTableRowElement);
      }
      function browse()
```

```

{
  var tmpNewTreeFct =
    function(parBrowseElement, parItemPath, parItemName)
    {
      var tmpBrowseTableHook =
        document.getElementById("BrowseTable");
      if (tmpBrowseTableHook.firstChild)
        tmpBrowseTableHook.removeChild(
          tmpBrowseTableHook.firstChild);
      gloBrowseTable = document.createElement("table");
      var tmpBodyElement = document.createElement("tbody");
      gloBrowseTable.appendChild(tmpBodyElement);
      tmpBrowseTableHook.appendChild(gloBrowseTable);
      if (parBrowseElement)
      {
        parBrowseElement.getElement().firstChild.nodeValue =
          "< " +
            parBrowseElement.getElement().firstChild.nodeValue;
        addItem(parBrowseElement);
      }
      alert("Path: " + parItemPath + "\nName: " + parItemName);
    };
  var tmpNewNodeFct = function(parBrowseElement)
  {
    parBrowseElement.getElement().firstChild.nodeValue =
      "+ " +
        parBrowseElement.getElement().firstChild.nodeValue;
    addItem(parBrowseElement);
  };
  var tmpNewLeafFct = function(parBrowseElement)
  {
    var tmpCB = function(parBrowseElement)
    {
      var tmpText = "Path: " +
        parBrowseElement.mItemPath + "\n" +
        "Name: " +
        parBrowseElement.mItemName + "\n";
      alert(tmpText);
    }
    parBrowseElement.setCB(tmpCB);
    addItem(parBrowseElement);
  };
  gloBrowser = new ApplBrowser(document, "", "",
                                tmpNewTreeFct,
                                tmpNewNodeFct,
                                tmpNewLeafFct);
}
function leave()
{
  if (gloBrowser)
    gloBrowser.destructor();
}
</script>
</head>
<body onload="browse();" onunload="leave();">
  <div id="BrowseTable"></div>
</body>

```

</html>

ApplBrowseTree

`ApplBrowseTree` builds upon the `ApplBrowser` and displays the browse result in tree format.

```
ApplBrowseTree (parDocument,  
                parItemPath,  
                parItemName,  
                parTablePrefix,  
                parLeafCB,  
                parNodeCB,  
                parBackCB,  
                parFilterCB,  
                parLeafImg,  
                parNodeImg,  
                parBackImg)
```

Transfer parameters:

- `parDocument`
: JavaScript document used to generate elements
- `parItemPath`, `parItemName`
: Specifies the starting point for browsing the variable management.
- `parTablePrefix`
Prefix for referencing elements in CSS
- `parLeafCB`
Callback function that is called if a leaf is clicked
- `parNodeCB`
Callback function that is called if a node is clicked. Each of the callback functions receives an `ApplBrowseElement` as a first parameter.
- `parBackCB`
Callback function that is called if the first element of the tree is clicked
- `parLeafImg`, `parNodeImg`, `parBackImg`
: Symbols that are displayed by a back, node, or leaf element.

User interface:

- `getElement()`
Supplies a table element that contains the browse results.
- `destructor()`
Releases the table and all resources connected to it

Example:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <title>Browse table demo</title>
    <style type="text/css">
      table.BrowseTree {table-layout:fixed;}
      td.BrowseTreeImg0 {width:20px;}
      td.BrowseTreeImg1 {width:20px;}
      td.BrowseTreeBrowse {overflow:visible;text-align:left;}
      a.refLeaf {cursor:pointer;}
      a.refNode {cursor:pointer;}
    </style>
    <script type="text/javascript" src="/common.js"></script>
    <script type="text/javascript" src="/opcxml.js"></script>
    <script type="text/javascript" src="/appl.js"></script>
    <script type="text/javascript">
      var gloBrowseTree = null;
      function browse()
      {
        var tmpLeafCB = function(parBrowseElement)
        {
          var tmpText = "Path: " + parBrowseElement.mItemPath + "\n"
                      + "Name: " + parBrowseElement.mItemName +
                      "\n";
          alert(tmpText);
        }
        var tmpFilterCB = function(parBrowseElement)
        {
          var tmpItemName = parBrowseElement.mItemName;
          var tmpPos = tmpItemName.indexOf("unit/");
          if (tmpPos != 0)
          {
            tmpPos = tmpItemName.indexOf("to/");
            if (tmpPos != 0)
              tmpPos = tmpItemName.indexOf("var/");
          }
          return (tmpPos == 0);
        }
        gloBrowseTree = new ApplBrowseTree(document,
                                           "SIMOTION",
                                           "",
                                           "BrowseTree",
                                           tmpLeafCB,
                                           undefined,
                                           undefined,
                                           tmpFilterCB
                                           );

        var tmpBrowseTreeHook =
```

```
        document.getElementById("BrowseTree");
        tmpBrowseTreeHook.appendChild(gloBrowseTree.getElement());
    }
    function leave()
    {
        if (gloBrowseTree)
            gloBrowseTree.destructor();
    }
</script>
</head>
<body onload="browse();" onunload="leave();">
    <div id="BrowseTree"></div>
</body>
</html>
```

3.1.8 MiniWeb Server Language (MWSL)

3.1.8.1 Mode of operation of the MWSL

The web server language is a scripting language that is interpreted on the web server. This scripting language is fairly similar to JavaScript, but is only a small subset of the complete language.

The MWSL enables the client to be operated with a simple browser without scripting, because the web server generates the pages dynamically.

MWSL enables access and processing of variables. Among other things, MWSL allows access to process variables that are present on the web server system. MWSL and the integrated template mechanism can then be used very effectively to process and evaluate these variables.

The template mechanism used to produce the dynamic pages is similar to a very simplified XSLT process. See W3C XSL transformation (<http://www.w3.org/standards/xml/transformation>)

The client requests a URL on the web server. The address is converted to the access to an MWSL file in the web server. From this file, the MWSL service generates a temporary HTML file on the web server. This file is then sent to the client and displayed there.

Note

Enabling access to the process variables

Access to the process variables with the MWSL functions GetVar and SetVar requires activation of the setting **Enable OPC_XML (load symbols to RT)** in SCOUT.

You can find more information in the manual SIMOTION IT Diagnostics and Configuration, in the chapters 'Accessing the global variables (V4.2 and higher)' and 'Making unit variables available'.

3.1.8.2 Structure of a MWSL file

An MWSL file is basically an HTML file which also contains MWSL tags. To distinguish them from HTML files, MWSL files have the extension ".mwsl." Loading of MWSL pages into the controller (Page 14)

Example:

```
<HTML>
  <HEAD>
    ...
  </HEAD>
  <BODY>
    <table>
      <tr>
        [...]
        <td>
          <MWSL>
            //MWSL code to be executed
          </MWSL>
        </td>
        [...]
        <td>
          <MWSL>
            //MWSL code to be executed
          </MWSL>
        </td>
        [...]
      </tr>
    </table>
  </BODY>
</HTML>
```

If the MWSL functionality is required, the following tags must be added:

- The `<MWSL>`-tag introduces an MWSL script
- The `</MWSL>`-tag marks the end of the script

For reasons of clarity, the examples below do not always include the HTML code and begin directly with the `<MWSL>`" tag.

3.1.8.3 Error messages

MWSL error messages

Errors that occur while an MWSL page is being used are output in two different ways:

1. Errors that occur when compiling the MWSL file are output in the logfile. The name of the log file is formed by appending ".log" to the file name.
2. Errors that occur when executing a web page, that is, when the page is called from the server, are written into the source text of the MWSL page as error messages.

Error output in the MWSL page

The source text of a page can be loaded to the editor in the Internet Explorer by right-clicking with the mouse and selecting the menu command **Show Source Text**.

Example

This example shows the comment associated with the query relating to an unavailable variable.

```
exec.mwsl
<html>
  <head>
    <title>SIMOTION
      <MWSL>
        WriteVar("DeviceInfo.Board");
      </MWSL>
    </title>
    <meta name="DC.Subject" content="SIMOTION">
    <meta name="DC.Publisher" content="Siemens AG">
    <meta name="DC.Format" content="text/html">
    <meta name="DC.Language" content="en">
    <meta name="DC.Rights" content="Copyrights Siemens AG 2003">
  </head>
  <body style="font-family: Arial">
    <p>
      <MWSL>WriteVar("var/userData.user8");</MWSL>
    </p>
    <p>
      <MWSL>WriteVar("var/userData.user9");</MWSL>
    </p>
  </body>
</html>
```

In `exec.mwsl` file, the statement `WriteVar("var/userData.user9");` is used to query a non-existent variable.

Source text for the output page:

```
<html>
  <head>
    <title>SIMOTION D435</title>
    <meta name="DC.Subject" content="SIMOTION">
    <meta name="DC.Publisher" content="Siemens AG">
    <meta name="DC.Format" content="text/html">
    <meta name="DC.Language" content="en">
    <meta name="DC.Rights" content="Copyrights Siemens AG 2003">
  </head>
  <body style="font-family: Arial">
    <p>
      495399
    </p>
    <p>
      <!-- MWSL2 Runtime Message (WARNING):
           Src Line# (MethodName) Source Text
           =====>
```

3.1 User-defined pages

```
000003  #(INTERPRETER) WriteVar("var/userData.user9");</MWSL>
External function returned an error: E6B20014.
-->

</p>
</body>
</html>
```

The MWSL2 Runtime Message comment contains a description of the cause of the error.

3.1.8.4 Variable types

MWSL distinguishes between script variables and global variables:

- Script variables are defined within the script
- Global variables are provided by variable sources

Note

Global variables are not part of the script engine. Rather, they represent information from the web server environment. Variables are accessed exclusively via access functions. The global variables are grouped into variable sources according to their origin.

3.1.8.5 Script variables

Script variables are variables that are only valid on the current page.

The variables apply beyond MWSL tags, i.e. they can be created in one MWSL tag and be used starting in the next MWSL tag.

For these variables, no distinction is made between variable types, i.e. there is no Int, Char, etc.

A variable is created as follows:

```
var <Variablennamen> = <Wert>;
```

The variable type is determined internally by the variable assignment.

Example:

```
<MWSL>
  var string1 = "Hello";
  var string2 = "World";
  write(string1 + " " + string2);
</MWSL>
```

Two variables are created in the example shown above: `string1` and `string2`.

The two strings are strung together (with spaces).

The `write` command outputs the result. See `write` (Page 122)

Output: Hello World

Example:

```
<MWSL>
  var num1 = 5;
  var num2 = 7;
  var Result;
  Result = num1 + num2;
</MWSL>
```

Two variables are created in the example shown above: `num1` and `num2`.

The two numbers are added, and the result is stored in the `Result` variable.

`Result` contains the value 12.

The data type is converted in the same way as ECMA Script 262. See ECMA script (<http://www.ecma-international.org/publications/standards/Ecma-262.htm>)

Keyword var

The keyword `var` introduces a variable declaration. In ECMA script, variables do not have to be explicitly declared.

Syntax:

```
var VarName = InitialValue, VarName2 = InitialValue2, ...;
```

Multiple variables are declared and (optionally) initialized with initial values.

You can specify several declarations, separated by commas.

For further information, please refer to the ECMA Script Definitions.

Ranges of visibility and validity

The visibility and validity of variables is analogous to ECMA script. (However, MWSL currently does not recognize any functions.)

Example:

```
<MWSL>
  var MyVar = 10;
  {
    MyVar = 20;
    write ("Inner:" + MyVar + "," );
  }
  write ("Outer:" + MyVar + "\n" );
</MWSL>
```

Output: Inner: 20, Outer: 20

In this example, the variable `MyVar` of the outer level will be accessed in the operation block, because a variable named `MyVar` was not declared on the operation block level.

Therefore, the `MyVar = 20` operation changes the value of the variable on the outer level.

```
<MWSL>
  var MyVar = 10;
```

```
{
    var MyVar = 20;
    write ("Inner:" + MyVar + "," );
}
write ("Outer:" + MyVar + "\n" );
</MWSL>
```

Output: Inner: 20, Outer: 10

In this example, the variable `MyVar` is created new at the inner level, which leaves the value of the variable `MyVar` having the same name at the outer level unchanged.

3.1.8.6 Global variables

Definition

Global variables enable access to the variable management area of the web server. There are different types of global variables:

- PROCESS variables enable access to normal variables of the web server. This is known as standard access.
- DEFAULT variables are identical to the PROCESS variables.
- URL variables provide access to variables contained in a URL.
- HTTP variables return the content of variables in the HTTP header.
- COOKIE variables allow access to cookies.

A variable can be accessed using the following command:

```
GetVar ("var/userData.user1", "PROCESS");
```

The variable source `PROCESS` must be written in upper case. If the `var/userData.user1` variable is not present, "null" is returned.

`PROCESS` is the standard variable source. Therefore, `PROCESS` can also be omitted.

```
GetVar ("var/userData.user1");
```

If a variable provider is to be addressed directly, the name of the desired provider can be specified instead of the variable source `PROCESS`.

Format string for the GetVar and WriteVar functions

The format string always starts with a % sign, followed by the specification of the number of characters. This is then followed by the type information.

The following type information is available:

- %d for Integer values
- %f for Float values
- %e for representing the value with exponent
- %s for Strings

Examples

```
GetVar("var/systemClock", "PROCESS", "%d");
GetVar("var/modeOfOperation", "PROCESS", "%s");
```

Table 3-1 Format strings

Example	Meaning
"%.6s"	Outputs the first 6 characters of the specified variable (as a string).
"%.3s"	Outputs the first 3 characters of the specified variable (as a string).
"%.s"	Outputs the complete variable (as a string).
"%3.2f"	Outputs the variable interpreted as Float. The 3 indicates that 3 total places are output. The 2 indicates that, of the 3 places, 2 places after the decimal point will be displayed.
"%4d"	Outputs the variable interpreted as Integer. Four places are output. This parameter can only be passed if the variable source "PROCESS" has also been passed.

If the format string is omitted, the complete variable content is returned.

Float numbers are rounded at output. For example, the number pi is output with the formatting "%4.3f" as 3,142

See also

GetVar (Page 112)

WriteVar (Page 123)

3.1.8.7 Special variables

Via the variable source HTTP, the value of the special variable `Username` can be queried.

```
GetVar("Username", "HTTP")
```

This call of `GetVar` returns the user currently logged on to the Web server.

The special variable `HTTP` supplies information on the HTTP connection.

```
GetVar("HTTP", "HTTP")
```

Output:

```
Host: 192.168.1.1User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:
28.0) Gecko/20100101 Firefox/28.0Accept: text/html,application/xhtml
+xml,application/xml;q=0.9,*/*;q=0.8Accept-Language: de,en-
US;q=0.7,en;q=0.3Accept-Encoding: gzip, deflateReferer: http://
192.168.1.1/index.mwslConnection: keep-alive
```

3.1.8.8 Configuration constants

Accessing constants of the WebCfg.xml configuration file

It is possible to create constant variables in the WebCfg.xml file. Accessing these constants enables more flexible programming, meaning that user-defined pages can be controlled using relevant configuration parameters.

A constant is defined in the following section of WebCfg.xml:

/SERVERPAGES/CONFIGURATION_DATA/UserConfig

```
<SERVERPAGES>
  <CONFIGURATION_DATA>
    <USERCONFIG>
      <MyParam>MyParamValue</MyParam>
    </USERCONFIG>
  </CONFIGURATION_DATA>
</SERVERPAGES>
```

It is possible to access the MyParam constant in an HTML page by specifying the "constants/MyParam" path.

```
<html>
  <head>
  </head>
  <body style="font-family: Arial">
    <table width="100%" height="100%" bgcolor="#00349A">
      <tr>
        <td style="color: #FFFFFF">
          <b><MWSL>WriteVar("constants/MyParam");</MWSL></b>
        </td>
      </tr>
    </table>
  </body>
</html>
```

Result:

```
<html>
  <head>
  </head>
  <body style="font-family: Arial">
    <table width="100%" height="100%" bgcolor="#00349A">
      <tr>
        <td style="color: #FFFFFF">
          <b>MyParamValue</b>
        </td>
      </tr>
    </table>
  </body>
</html>
```

3.1.8.9 Variables and URL parameters

MWSL offers the option of editing URL parameter values using the `WriteVar`, `GetVar`, `SetVar`, and `ExistVariable` functions.

Example of a URL with appended parameters. The line break has been added for better readability:

```
http://localhost/MWSL/StringOperationtest.mwsl?
Parameter1=Hallo&Parameter2=du!&StartValue=2&EndValue=5
```

The URL points to the page `StringOperationtest.mwsl` and transfers the parameters `Parameter1`, `Parameter2`, `StartValue` and `EndValue`.

The following command outputs the URL variable `Parameter1`:

```
WriteVar("Parameter1", "URL");
```

Note that "URL" must be written in upper case.

If a URL variable that is not present in the URL is requested, an empty string ("") is always returned. This return is not a script error.

Parameters in URLs

In a URL, parameter passing begins after the "?" character. Individual parameters are separated by "&" characters. The value is assigned after the "=" character.

Certain characters require a coding in order to be passed correctly. The following table provides an overview of the most commonly used escape codes.

Table 3-2 URL escape codes

Character	Escape code
Blank	%20
<	%3C
>	%3E
#	%23
%	%25
{	%7B
}	%7D
	%7C
\	%5C
^	%5E
~	%7E
[%5B
]	%5D
`	%60
;	%3B
/	%2F
?	%3F

Character	Escape code
:	%3A
@	%40
=	%3D
&	%26
\$	%24

The coding consists of the '%' character followed by the ASCII hexadecimal value of the desired character.

You can find more information on the Internet, e.g. at <http://de.selfhtml.org/>.

3.1.8.10 COOKIES

The following example shows how a cookie can be created in an MWSL file.

For this purpose, insert the <META> tag into the <HEAD> tag:

```
http-equiv="SET-COOKIE" content="siemens_automation_language=de;"
```

Example:

```
<HTML>  
  <HEAD>  
    <META http-equiv="SET-COOKIE"  
          content="siemens_automation_language=de;">  
  </HEAD>  
  <BODY>  
    [...]   
  </BODY>  
</HTML>
```

For more information on cookies: <http://de.selfhtml.org/>

Setting a cookie as an HTTP header

It is possible to set HTTP header for the HTTP response from MWSL. For example, this allows a cookie to be set via the HTTP header and not as a <META> tag.

Example:

```
<MWSL>  
  var strCookie;  
  strCookie = "Set-cookie: siemens_automation_language=";  
  strCookie = strCookie + GetVar( "Language", "URL");  
  strCookie = strCookie + ", path=\\r\\n";  
  AddHTTPHeader( strCookie );  
  
  write("HTTP lang: " + GetVar("Cookie", "HTTP"));  
</MWSL>
```

In this example, a cookie for detection of the language setting is set.

3.1.8.11 Variables and access to COOKIES

The access occurs similarly as for the URL parameters. The single difference is that the variable type is now `COOKIE` instead of `URL`.

The variable specified in the example can be accessed with the following command, for example:

```
GetVar("siemens_automation_language", "COOKIE");
```

Note that `COOKIE` must be written in upper case.

If the `COOKIE siemens_automation_language` is set with `en`, for example, the above call would return this value.

For more information on cookies: <http://de.selfhtml.org/>

If the variable is not present, there is no output.

3.1.8.12 Variables and HTTP header information

Using the variable source `HTTP`, the values are read out of the HTTP query header fields. These are, for example, `Accept`, `Content-Type`, or `User-Agent`.

Table 3-3 Example output of an HTTP header field

```
WriteVar("User-Agent", "HTTP");
```

Output of the Mozilla Firefox browser:

```
Mozilla/5.0 (Windows NT 6.1; WOW64; rv:25.0) Gecko/20100101 Firefox/25.0
```

Table 3-4 Example of generation of an HTTP HEADER with the HTML META tag

Example:

```
<HTML>
  <HEAD>
    [...]
    <META http-equiv="Accept-Language" content="de">
    [...]
  </HEAD>
  <BODY>
    [...]
  </BODY>
</HTML>
```

In this example, the HTTP variable `Accept-Language` is defined in the META tag using the `"http-equiv="Accept-Language"` command. It is initialized by the `content` attribute with the value `de`.

Additional information on META information: <http://de.selfhtml.org/html/kopfdaten/meta.htm#allgemeines>

This variable is accessed similarly as for the URL parameters. The difference is that the variable source is `HTTP` and not `URL`. The variable specified in the example can be accessed with the following command:

```
GetVar ("Accept-Language", "HTTP");
```

The variable source `HTTP` must be written in upper case.

Format string for the GetVar and WriteVar functions

As the third parameter of the functions `GetVar` and `WriteVar`, a format string can be specified. The format string defines from which position how many characters of the variable source will be returned.

```
GetVar ("Accept-Language", "HTTP", "[3,5]")
```

In the above example, 5 characters of the `HTTP` variables `Accept-Language` are returned starting from the third character.

The format string can only be passed if a variable source, such as `HTTP` in the above example, has also been passed.

If the format string is omitted, the complete variable content is returned.

See also

Global variables (Page 52)

GetVar (Page 112)

WriteVar (Page 123)

3.1.8.13 Operators

MWSL operators

All operators presented here behave as defined in ECMA 262. For more information, refer to the ECMA 262 Specification.

Boolean values are converted to numerical values 0 (for false) and 1 (for true) where applicable.

Table 3-5 Relational operators

Operator	Comment
<code>x < y</code>	This operator returns true if the <code>x</code> variable is less than the <code>y</code> variable. Otherwise, the value returned is false.
<code>x <= y</code>	This operator returns true if the <code>x</code> variable is less than or equal to the right <code>y</code> variable. Otherwise, the value returned is false.
<code>x > y</code>	This operator returns true if the <code>x</code> variable is greater than the <code>y</code> variable. Otherwise, the value returned is false.
<code>x >= y</code>	This operator returns true if the <code>x</code> variable is greater than or equal to the <code>y</code> variable. Otherwise, the value returned is false.

Operator	Comment
<code>x == y</code>	This operator returns true if the <code>x</code> variable is equal to the <code>y</code> variable. Otherwise, the value returned is false.
<code>x != y</code>	This operator returns true if the <code>x</code> variable is not equal to the <code>y</code> variable. Otherwise, the value returned is false.

Table 3-6 Logic operators

Operator	Remark
<code>!x</code>	Logical NOT This operator returns a false if <code>x</code> is true. if <code>x</code> is false, the value returned by the operator will be true.
<code>x && y</code>	Logical AND This operator returns true if <code>x</code> and <code>y</code> have the value true. Otherwise, the value returned is false.
<code>x y</code>	Logical OR This operator returns a false if <code>x</code> and <code>y</code> have the value false. Otherwise, the value returned is true.

Table 3-7 Object operators

Operator	Remark
<code>new</code>	Creating an object This operator creates an object. The object can be of a pre-integrated or a user-defined type.
<code>delete</code>	Deleting an object This operator deletes an object created with <code>new</code> .

Table 3-8 Bit-by-bit operators

Operator	Remark
<code>x & y</code>	Bit-by-bit AND If <code>x</code> and <code>y</code> are one at a bit position, this bit position will have the value one. All bit positions are checked.
<code>x y</code>	Bit-by-bit OR If <code>x</code> or <code>y</code> are one at a bit position, this bit position will have the value one. All bit positions are checked.
<code>x ^ y</code>	Bit-by-bit XOR If <code>x</code> or <code>y</code> is one at a bit position, but they are not both one, this bit position will have the value one. All bit positions are checked.
<code>x >> y</code>	Right shift considering the sign In <code>x</code> , the bits are shifted right by <code>y</code> digits. If <code>x</code> is positive, zeroes are filled in from the left; if <code>x</code> is negative, ones are filled in from the left.

Operator	Remark
$x \ggg y$	Right shift without considering the sign In x , the bits are shifted right by y digits. Zeroes are filled in from the left.
$x \lll y$	Left shift In x , the bits are shifted left by y digits. Zeroes are filled in from the right.
$\sim x$	Bit-by-bit NOT Inverted bits of x .

Table 3-9 Arithmetic operators

Operator	Remark
$x + y$	Plus operator This operator adds x and y .
$++x$	Increment Increments the value of x by one. $x++$ or $++x$ is possible.
$x - y$	Minus operator This operator subtracts the value y from x .
$--x$	Decrement Decrements the value of x by one. $x--$ or $--x$ is possible.
x / y	This operator divides x by y . The return value is a float.
$x \% y$	Modulo This operator returns the integer remainder of a division of x and y .
$x * y$	This operator multiplies x by y .

Table 3-10 Assignment operators

Operator	Remark
$x = y$	Assignment operator This operator assigns the value of the right-hand expression to x , in this case the variable y .
$x += y$	Addition Assigns x the value of $x + y$.
$x -= y$	Subtraction Assigns x the value of $x - y$.
$x *= y$	Multiplication Assigns x the value of $x * y$.
$x /= y$	Division Assigns x the value of x / y .
$x ^= y$	Bit-by-bit OR Assigns x the value of $x \wedge y$.
$x = y$	Bit-by-bit OR Assigns x the value of $x y$.

Operator	Remark
<code>x &= y</code>	Bit-by-bit AND Assigns <code>x</code> the value of <code>x y</code> .
<code>x %= y</code>	Modulo Assigns to <code>x</code> the integer remainder of a division with <code>y</code> .
<code>x >>= y</code>	Bit-by-bit right shift Assigns <code>x</code> the value of <code>x >> y</code> .
<code>x <<= y</code>	Bit-by-bit left shift Assigns <code>x</code> the value of <code>x << y</code> .

Table 3-11 Conditional operator

Operator	Remark
<code>Condition ? x : y</code>	Conditional operator This operator has the value <code>x</code> , if the <code>condition</code> is true. Otherwise, the value of the operator is <code>y</code> .

3.1.8.14 Conditional operations

if Conditions

The `if` condition is familiar from Ecma 262.

Syntax

```
if(<condition>)
  Operation1
else
  Operation2;
```

If the condition is true, instruction 1 is run.

If the condition is false, instruction 2 is run. If no `else` part is present, processing continues in accordance with the `if` operation.

Example:

```
<MWSL>
[...]
if (ExistVariable("Parameter", "PROCESS"))
{
  WriteVar("Parameter");
}
[...]
```

If the `Parameter` process variable exists, its content is output.

If not, the instruction is skipped and the program execution is then resumed. In the example above, this would be the code that follows after the closing curly bracket.

If an `else` branch is present, it is run provided that the condition has not been fulfilled.

Example:

```
<MWSL>
[...]
if (ExistVariable("Parameter") && GetVar("Parameter")>=3)
{
    write("The parameter value is:");
    WriteVar("Parameter");
    write("This is a valid value!");
}
else
{
    write("Parameter process variable is not permitted or is not available");
}
[...]
```

If the `Parameter` variable is not present, the `else` part is executed. A message is then output, indicating that no variable with the specified name exists.

As the examples show, an operation can be replaced by an operation block.

An operation block is a list of operations that is enclosed in curly brackets.

Example:

```
<MWSL>
[...]
```

```
if (ExistVariable("Parameter") && GetVar("Parameter")>=3)
{
    WriteVar("Parameter");
}
else
{
    write("Parameter process variable is not permitted or is not
available");
}
[...]
```

If the `Parameter` process variable exists and the content is greater than or equal to 3, it is output. Otherwise, a corresponding output is made.

switch Condition

The `switch` operation makes it possible to compare an `<expression>` with many conditions `<value1>`, `<value2>`, etc.

```
switch (<expression>)  
{  
  case <value1>:  
    //Program block  
    break;  
  case <value2>:  
    //Program block  
    break;  
    //etc.  
  default:  
    //Program block  
}
```

3.1.8.15 Loops

for Loop

The MWSL provides a loop mechanism, such as is already familiar from JavaScript.

For a detailed description, refer to the ECMA 262 Specification.

Syntax

```
for( Start statement; End condition; Run statement)  
{  
    Loop body, code to be executed  
}
```

Sequence:

1. The start instruction is executed
2. The loop body is executed
3. The run operation is executed
4. As long as the end condition is true, the processing is repeated starting from the loop body (2.).

Example:

```
for(i=1; i<5; i++)  
{  
    write(i);  
}
```

Output: 1234

do while Loop

In the `do while` loop, the body of the loop is first run then the `while <condition>` is tested.

3.1 User-defined pages

```
do
{
  //Operation block
} while (<condition>;
```

break continue

To break or resume loop execution, two operations `break` and `continue` are available.

With the `break` operation, the loop is always exited immediately without testing the `<condition>` again.

The `continue` operation jumps immediately to the head of the loop.

3.1.8.16 Functions

A user-defined function consists of the function operation and a program block of operations. No or multiple parameters can be passed.

A `<Wert>` can be returned. If no return value is specified, the function will return `undefined`.

```
function Function name([<parameter1>,
<parameter2>, ...])
{
  //Program block
  return <Wert>;
}
```

3.1.8.17 Comments

Comments can comprise one or more lines in the MWSL.

```
// One-line comment. All characters are
ignored up to the line break.

/*
Multi-line comment starts with '/*' and
must be ended with '*/'.
*/
```


3.1.8.18 Overview of MWSL functions

The MWSL provides a variety of functions, which are presented in the following overview table. For detailed descriptions of the functions, refer to the appendix.

Table 3-12 MWSL functions

Function name	Explanation
AddHTTPHeader(<Http header>) (Page 109)	Insert <Http Header> in a page.
createGUID() (Page 109)	Generates a unique alphanumeric ID in the system.
DecodeString(<string>) (Page 110)	Converts a string encoded with EncodeString back to its original.
die(<Param0>,<Param1>,...) (Page 110)	Abort program execution.
EncodeString(<string>) (Page 111)	Replaces special characters by their URL-coded hex value (%hh).
ExistFile(<file name>) (Page 111)	Checks whether a file with the name <parFileName> exists. The function returns file length as returned value.
ExistVariable(<variable name>, <variable source>) (Page 112)	Query of the existence of a variable.
GetLanguage() (Page 112)	Returns the currently set language in English.
GetVar(<variable name>, <variable source>, <format string>) (Page 112)	Return of the value of a variable of the corresponding variable source
InsertFile(<text file>) (Page 114)	Import of a <Test File>. A path can be specified.
IsAuthAlgo(<parAuthMethod>) (Page 115)	Returns true if the user logged on at the time of calling could be identified by the authentication method specified in parAuthMethod.
isFinite(<value>) (Page 115)	Returns false if the passed value is NaN or infinite.
isNaN(<value>) (Page 115)	Checks whether the passed value is an invalid double.
IsSSL() (Page 116)	Returns true if the client is connected to the server via an SSL connection.
parseFloat(<string>) (Page 116)	Conversion of a string to a double value.
parseInt(<value>,[<basis>]) (Page 116)	Conversion of a string to an integer value.
ProcessXMLData(<DATA>, <TEMPLATE>) (Page 118)	Generation of dynamic HTML files with special XML files.
string ReadFile(<file name>) (Page 119)	Returns the content of the file as the return value.
ReplaceString<variable name>,<search pattern>,<replacement string>) (Page 120)	Replacement of strings matching the search pattern.
SetVar(<variable name>, <value>) (Page 120)	Sets values of variables.
ShareRealm(<group name>) (Page 120)	Indicates whether the current user is a member of the group that is passed as a parameter. The return value can be true or false.
write(<text>) (Page 122)	Writes <Text> strings to the HTML page. <Text> can also be the return value of functions.
WriteToTab(<parTabPos>, <parFillChar>) (Page 122)	Write <parFillChar> up to position <parTabPos>.

Function name	Explanation
WriteVar(<variable name>, <variable source>, <format string>) (Page 123)	Output of a variable value. The syntax is identical to the GetVar() function.
WriteXMLData (<DATA>, <TEMPLATE>) (Page 125)	Outputs the data directly in contrast to ProcessXMLData().

Table 3-13 MWSL process variables

Process variable	Explanation
NodeIndex (Page 126)	Parse process variable for the template. This variable outputs the number of nodes that have already been run.
NodeLevel (Page 127)	Parse process variable for the template. This variable outputs the hierarchy level of the current node.

3.1.8.19 Mode of operation of the template mechanism

A Template is applied to data elements of a data source. This mechanism allows separate implementation of data and processing.

The Template-mechanism is started by the `ProcessXMLData()` and `WriteXMLData()` commands.

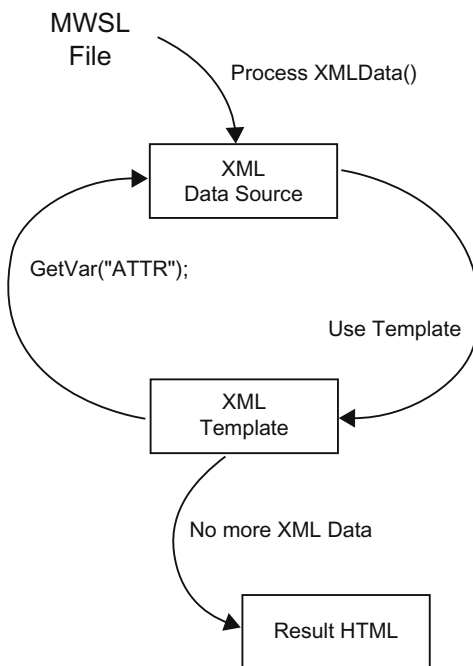


Figure 3-13 Basic overview of the template mechanism

The data file contains structured data that can be output as transformed data using the Template-mechanism.

An XML file consists of a series of XML nodes. A name as well as a set of attributes are assigned to each XML node. An attribute, in turn, consists of a name and a value.

The Template file contains a set of transformation operations.

A transformation operation can be assigned to an XML node (with a certain name).

During the transformation process, the data fragment is read node-by-node from top to bottom. If a Template can be assigned to a node, this Template is executed and the attributes of the current node are available to the Template as variables.

3.1.8.20 Structure of the template file

The Template file is an XML file whose data nodes contain the transformation operations for processing a data file.

Example

```
<?xml version="1.0" ?>
<TEMPLATES>
  [...]
  <TEMPLATE NAME="Variable">
    [...]
    <POSITION NAME="LINE">
      <![CDATA[
        [...]
        <MWSL>Name ()</MWSL>
        [...]
      ]]>
    </POSITION>
    [...]
  </TEMPLATE>
  [...]
</TEMPLATES>
```

The Template file evaluates the data nodes of the data file.

The individual Template tags are defined in the `<TEMPLATES>` tag (in the example above, only one template tag is defined).

The line `<TEMPLATE NAME="Variable">` defines that this Template (the subsequent code) is only run for data nodes of the "Variable" type. The type of a data node is specified either through the name of the data node or through a special attribute named "Template".

```
<POSITION NAME="LINE">
  [...]
</POSITION>
```

This tag is used to specify the area of the web page in which the content of the template is to be output.

The positions `HEAD`, `LINE`, and `FOOT` are available and are run in that order during processing.

The content of the `POSITION` is encapsulated in a `CDATA`-block in order to protect it from the XML parser:

```
<![CDATA[ [...] ]>
```

MWSL can now be used to access the attributes of the current data node and to output them or use them in other operations.

Example:

```
<MWSL> WriteVar("Name")</MWSL>
```

3.1.8.21 Structure of a data source

A data source is an XML fragment, in which the nodes of the XML fragment form the data elements.

Note

If the XML fragment does not contain a root node, MWSL itself generates a root node so that the data source conforms to XML.

The XML fragment can also be a complete XML document.

Example:

```
<?xml version="1.0" standalone="yes"?>
  [...]
  <Variable Name="ZUFUEHRUNG.STATE"
            Type="String"
            InitialValue="good"
            Behavior="Manual"
            Description="Status of part infeed."
  />
  [...]
  <Variable Name="Language"
            Type="String"
            InitialValue="de_DE"
            Behavior="Manual"
            Description="Language setting of web page"
  />
  [...]
```

The example defines data nodes (in the example: `Variable`) with the associated attributes.

The different nodes can be evaluated using a corresponding template file.

A further option is the creation of a data structure (hierarchy). That is, data nodes can be created that, in turn, contain other data nodes.

Example:

```
<?xml version="1.0" standalone="yes"?>
  [...]
  <StructVariable Name="Farbe"
                 Type="String"
```

```

                InitialValue="gelb"
                Behavior="Manual"
                Description="Fictitious value">
    [...]
    <Subvar Name="Rotteil"
        Type="Integer"
        InitialValue="128"
        Behavior="Manual"
        Description="The red proportion of the color"
    />
    <Subvar Name="Blauteil"
        Type="Integer"
        InitialValue="128"
        Behavior="Manual"
        Description="The blue proportion of the color"
    />
    <Subvar Name="Grunteil"
        Type="Integer"
        InitialValue="128"
        Behavior="Manual"
        Description="The green proportion of the color"
    />
    [...]
</StructVariable>
    [...]

```

This example uses the data node of type `StructVariable`. This data node contains several data nodes of type `Subvar`.

Different templates can be used for the different types of data nodes.

3.1.8.22 Template transformation

The `ProcessXMLData()` command dynamically generates an HTML file (or just a text fragment) from an XML data file and an XML template file.

For this purpose, the parser runs through the data file step by step, from top to bottom.

The parser reads in a data node. Following this, a search is performed in the template file for a matching template for this data node. If a template is found, this template will be applied to the data node. The template is an MWSL fragment. The sole difference is that the attributes of the XML data node in the template are available as standard process variables if a variable source is not specified. If there are identical names, the XML attributes overlay the corresponding process variables. If the variable source `PROCESS` is specified explicitly, the process variables are always used.

Example:

```

Prozessvariable
Color Value "Green"

```

Data file:

```

<?xml version="1.0" standalone="yes"?>
[...]
<Variable Name="ZUFUEHRUNG.STATE"
    Farbe="Red"

```

```

/>
[...]
<Variable Name="Language"
/>
[...]

```

Template file:

```

<?xml version="1.0" ?>
<TEMPLATES>
  <TEMPLATE NAME="Variable">
    <POSITION NAME="LINE">
      <![CDATA[
        <MWSL>write (GetVar("Name")+":");</MWSL>
        <MWSL>write (GetVar("Color")+"\r\n");</MWSL>
      ]]>
    </POSITION>
  </TEMPLATE>
</TEMPLATES>

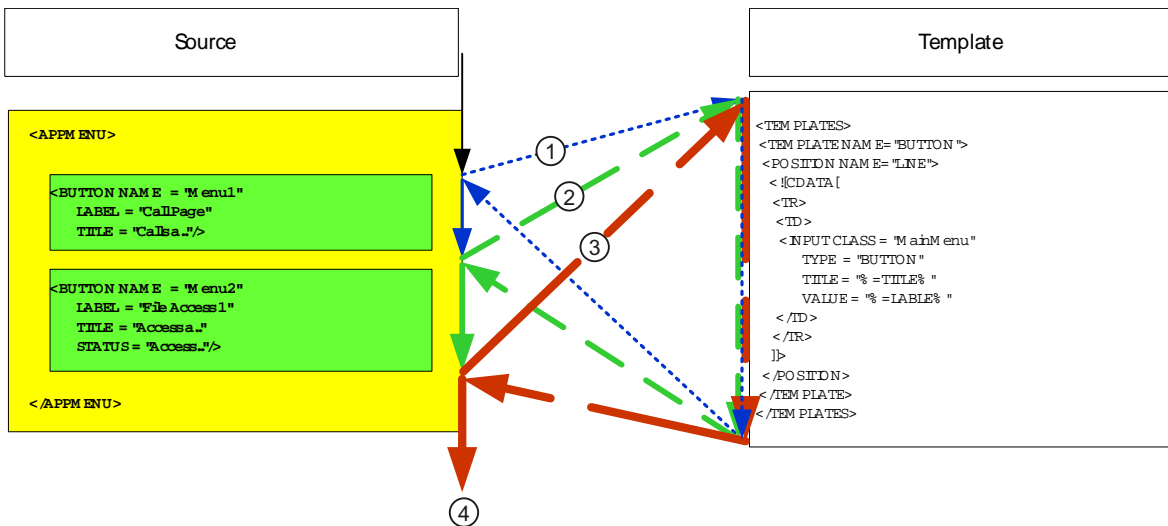
```

Output:

ZUFUEHRUNG.STATE: Red

Language: Green

Sequence of a Template-process



The figure shows the sequence of the Template Parse-process.

The Template is run through one time for each data element (yellow and green blocks). The attributes of the current data element can thereby be accessed as variables.

Chronological sequence

1. The <APPMENU> tag is found in the Source and the Template is run. No processing operation is found for this tag in the Template, therefore, no data element is written to the output.
2. The first <BUTTON> tag is found and processed by the Template .
3. The second <BUTTON> tag is found and processed by the Template .
4. No other tags are found; processing is complete.

Another example:**Calling MWSL file**

```

<HTML>
  <HEAD>
    <TITLE>
      MWSL Template Test Page
    </TITLE>
  </HEAD>
  <BODY >
    <MWSL>
      write(ProcessXMLData (
        "<EXTERNAL SRC=\"/FILES/MWSL/variables.xml \"/>",
        "<TEMPLATES><EXTERNAL SRC=\"/FILES/MWSL/variablesTemplate.xml\"/"
      >
        </TEMPLATES>")
      );
    </MWSL>
  </BODY>
</HTML>

```

variables.xml

```

<?xml version="1.0" standalone="yes"?>
<Provider Name ="MyVarProvider">
  <Variable Name="ZUFUEHRUNG.STATE"
    Type="String"
    InitialValue="good"
    Behavior="Manual"
    Description="Status of part infeed."
  />
  <Variable Name="Language"
    Type="String"
    InitialValue="de_DE"
    Behavior="Manual"
    Description="Language setting of web page"
  />
</Provider>
<!-- End of File -->

```

variablesTemplate.xml

```

<?xml version="1.0" ?>
<TEMPLATES>
  <TEMPLATE NAME="Provider">
    <POSITION NAME="HEAD">

```

```

<![CDATA[
  <TABLE BORDER="1">
    <TR>
      <TH>
        Variablenname
      </TH>
    </TR>
  ]]>
</POSITION>
<POSITION NAME="FOOT">
<![CDATA[
  <TR>
    <TD>
      <MWSL><!--GetVar("Name");--></MWSL>
    </TD>
  </TR>
</TABLE>
]]>
</POSITION>
</TEMPLATE>
<TEMPLATE NAME="Variable">
  <POSITION NAME="LINE">
    <![CDATA[
      <TR>
        <TD>
          <MWSL>
            <!--
              GetVar("Name");
            -->
          </MWSL>
        </TD>
      </TR>
    ]]>
  </POSITION>
</TEMPLATE>
</TEMPLATES>

```

Call the `ProcessXMLData` command to start the parser with the data file `variables.xml`. The `<Provider>` tag is found first.

In the `variablesTemplate.xml` template file, a search is performed in order to ascertain whether a template has been defined for this type.

The `Position="HEAD"` area is executed.

The parser reads the next tag from the data file and generates the additional lines of the HTML file to be output based on the appropriate template in the template file.

The same happens with the next tag.

The footer part of the template provider is executed with the end tag `</Provider>`.

In the following expression, the generated parts are shown bold.

Generated file:

```

<HTML>
  <HEAD>
    <TITLE>

```



```
MWSL Template Test Page
</TITLE>
</HEAD>
<BODY >
  <TABLE BORDER="1">
    <TR>
      <TH>Variablenname</TH>
    </TR>
    <TR>
      <TD>ZUFUEHRUNG.STATE</TD>
    </TR>
    <TR>
      <TD>Language</TD>
    </TR>
    <TR>
      <TD>Hallo</TD>
    </TR>
  </TABLE>
</BODY>
</HTML>
```

3.1.8.23 MWSL in XML attributes

As part of template parsing, it is also useful to write MWSL-statements to XML attributes of the data file, which are then evaluated at the time of parsing.

Example:

```
<?xml version="1.0" standalone="yes"?>
<Motor Name="M1"
  Nummer="1"
  Type="Dreh"
  Nennleistung = "7"
  Drehzahl="3"
  Alter="2"
  Farbe="RED"
  Prozess="&MWSL;WriteVar (&quot;CPULoad.Percent&quot;; , &quot;PROCESS&qu
ot;;
                                     &quot;%e&quot;); &END_MWSL;"
/>
```

The example uses the following MWSL command:

```
"&MWSL;WriteVar (&quot;CPULoad.Percent&quot;; , &quot;PROCESS&quot;; , &quot;
%e&quot;);
&END_MWSL;"
```

The command would appear as follows in a template file or an MWSL file:

```
<MWSL>WriteVar ("CPULoad.Percent", "PROCESS", "%e");</MWSL>
```

This command outputs the value of the `CPULoad.Percent` variables from the `PROCESS` variable source.

Please note that `<MWSL>` must be replaced with `&MWSL;`, and `</MWSL>` with `&END_MWSL;`. In addition, double quotation marks `"` must be replaced with `"`.

The rest conforms to the MWSL syntax.

3.1.8.24 Examples

Examples of how MWSL can be used

The examples shown here outline the options for using MWSL.

Setting variable values using the SetVar function

```
<MWSL>
  SetVar (GetVar ("VARNAME"), GetVar (GetVar ("VARNAME"), "URL"));
</MWSL>
```

In this example, the variable whose name is saved in the `VARNAME` process variable is initialized with the value of the `VARNAME URL` variable.

For illustration

`GetVar ("VARNAME")` supplies the content of the `VARNAME` process variable. This value will be viewed, in turn, as a variable name.

If we assume that the content of the `VARNAME` process variable is "Jack", the overall call already looks much simpler:

Overall call: `SetVar ("Jack", GetVar ("Jack", "URL"));`

If we were to assume that the URL variable "Jack" had the content "is a great guy", this would be equivalent to the following expression:

```
SetVar("Jack", "is a great guy");
```

TestTemplate.mwsl

This example will be used to briefly explain the template mechanism again.

For illustrative purposes, a few passages in the respective files are marked.

The template produces a table.

The called file

```
TestTemplate.mwsl
<HTML>
  <HEAD>
    <TITLE>
      MWSL Template Test Page
    </TITLE>
  </HEAD>
  <BODY>
  <MWSL>
```

```

        write(ProcessXMLData("<EXTERNAL SRC=\" /FILES/MWSL/variables.xml\" />",
            "<TEMPLATES><EXTERNAL SRC=\" /FILES/MWSL/variablesTemplate.xml\" />
            </TEMPLATES>")
        );
    </MWSL>
</BODY>
</HTML>

```

VariablesTemplate.xml

```

<?xml version="1.0" ?>
<TEMPLATES>
    <TEMPLATE NAME="Provider">
        <POSITION NAME="HEAD">
            <![CDATA[
                <TABLE BORDER="1">
                    <TR>
                        <TH>
                            Varname
                        </TH>
                        <TH>
                            Type
                        </TH>
                        <TH>
                            Description
                        </TH>
                        <TH>
                            Value
                        </TH>
                        <TH>
                            NI/NL
                        </TH>
                    </TR>
                </TABLE>
            ]]>
        </POSITION>
        <POSITION NAME="FOOT">
            <![CDATA[
                <TR>
                    <TD COLSPAN="5">
                    </TD>
                </TR>
            ]]>
        </POSITION>
    </TEMPLATE>

    <TEMPLATE NAME="Variable">
        <POSITION NAME="LINE">
            <![CDATA[
                <TR>
                    <TD align=center>
                        <A HREF="<MWSL>Link () </MWSL>"><MWSL>Name () </MWSL></A>
                    </TD>
                    <TD>
                        <MWSL>Type () </MWSL>
                    </TD>
                    <TD>
                    </TD>
                </TR>
            ]]>
        </POSITION>
    </TEMPLATE>

```

```

        <MWSL>Description()</MWSL>
    </TD>
    <TD align=right>
        <MWSL>InitialValue()</MWSL>
    </TD>
    <TD>
        <MWSL> NodeIndex() </MWSL> / <MWSL> NodeLevel() </MWSL>
    </TD>
</TR>
]]>
</POSITION>
</TEMPLATE>
</TEMPLATES>

```

In this case, the template file consists of multiple templates. The `Provider` template places the header and footer parts for the data (table header and footer).

The `Variable` template is appointed for the lines of data output (one table row per variable entry).

The data are inserted after the corresponding attributes.

The data file variables.xml

```

<?xml version="1.0" standalone="yes"?>
<Provider Name ="MyVarProvider">
  <Variable Name="ZUFUEHRUNG.STATE"
  Type="String"
  InitialValue="good"
  Behavior="Manual"
  Description="Status of part infeed"
  />
  <Variable Name="Language"
  Type="String"
  InitialValue="de_DE"
  Behavior="Manual"
  Description="Language setting of web page"
  />
</Provider>

```

Generated HTML file

```

<HTML>
  <HEAD>
    <TITLE>
      MWSL Template Test Page
    </TITLE>
  </HEAD>
  <BODY>
    <TABLE BORDER="1">
      <TR>
        <TH>

```

```

        Varname
    </TH>
    <TH>
        Type
    </TH>
    <TH>
        Description
    </TH>
    <TH>
        Value
    </TH>
    <TH>
        NI/NL
    </TH>
</TR>
<TR>
    <TD align=center>
        <A HREF="">ZUFUEHRUNG.STATE</A>
    </TD>
    <TD>
        String
    </TD>
    <TD>
        Status of part infeed
    </TD>
    <TD align=right>
        good
    </TD>
    <TD>
        2 / 2
    </TD>
</TR>
<TR>
    <TD align=center>
        <A HREF="">Language</A>
    </TD>
    <TD>
        String
    </TD>
    <TD>
        Language setting of web page
    </TD>
    <TD align=right>
        de_DE
    </TD>
    <TD>
        3 / 2
    </TD>
</TR>
</TABLE>
</BODY>
</HTML>

```

MainNavigation.mwsl

```

<html>
  <head>
    <title>
      MiniWeb Main Navigation
    </title>
  </head>
  [...]
  <body>
    <table>
      <tr>
        <MWSL>
          write(ProcessXMLData (
            "<EXTERNAL SRC=\""/XML/MainNavigation.xml\"/>",
            "<TEMPLATES><EXTERNAL
              SRC=\""/Templates/MainNavigation.xml\"/>
            </TEMPLATES>") );
          </MWSL>
        </tr>
      </table>
    </body>
  </html>

```

This file contains the actual body of the HTML page to be generated.

The dynamic part is generated with the `ProcessXMLData()` command and inserted in `<table>`.

/XML/MainNavigation.xml

MainNavigation.xml contains the data part for the generation.

```

<?xml version="1.0" standalone="yes"?>
<MAINNAVIGATION>
  <APPLICATION NAME = "Entrance"
    CLIENTAREA = "/Portal/Entrance.mwsl"
    TITLE = "Back to Entrance Page." />
  <APPLICATION NAME = "MWSL Test"
    CLIENTAREA = "/MWSL/Start.mwsl"
    TITLE = "Test environment for MWSL." />
  <APPLICATION NAME = "File Browser"
    REALM = "Administrator"
    CLIENTAREA = "/www"
    TITLE = "Browse the Filesystem" />
  [...]
  <APPLICATION NAME = "CSSA"
    REALM = "User"
    CLIENTAREA = "/CSSA/Main.mwsl"
    TITLE = "PKI Interface." />
  <APPLICATION NAME = "VarSimulator"
    CLIENTAREA = "/Simulator/Simulator_index.mwsl"
    TITLE = "Simulate several variables." />
</MAINNAVIGATION>

```

/Templates/MainNavigation.xml

```
<?xml version="1.0" standalone="yes"?>
<TEMPLATES>
  <TEMPLATE NAME="APPLICATION">
    <POSITION NAME="LINE">
      <![CDATA[
        <td>
          <input class = "MainMenu"
            type = "BUTTON"
            title = "<MWSL> WriteVar("TITLE")</MWSL>"
            value = "<MWSL> WriteVar("NAME")</MWSL>"
            OnClick =
              "NavigateApp('<MWSL>WriteVar("CLIENTAREA")</MWSL>') "
          />
        </td>
      ]]>
    </POSITION>
  </TEMPLATE>
</TEMPLATES>
```

This file is run for each data node and the appropriate variables are inserted.

Generated HTML file

```
<html>
  <head>
    <title>
      MiniWeb Main Navigation
    </title>
  </head>
  <body>
    <table>
      <tr>
        <td>
          <input class = "MainMenu"
            type = "BUTTON"
            title = "Back to Entrance Page."
            value = "Entrance"
            OnClick = "NavigateApp('/Portal/Entrance.mwsl') "
          />
        </td>
        <td>
          <input class = "MainMenu"
            type = "BUTTON"
            title = "Test environment for MWSL."
            value = "MWSL Test"
            OnClick = "NavigateApp('/MWSL/Start.mwsl') "
          />
        </td>
        <td>
          <input class = "MainMenu"
            type = "BUTTON"
            title = "Browse the Filesystem"
            value = "File Browser"
            OnClick = "NavigateApp('/www') "
          />
        </td>
      </tr>
    </table>
  </body>
</html>
```

```

        </td>
        [...]
        <td>
            <input class = "MainMenu"
                type = "BUTTON"
                title = "PKI Interface."
                value = "CSSA"
                OnClick = "NavigateApp('/CSSA/Main.mwsl')"/>
        </td>
        <td>
            <input class = "MainMenu"
                type = "BUTTON"
                title = "Simulate several variables."
                value = "VarSimulator"
                OnClick = "NavigateApp
                    ('/Simulator/Simulator_index.mwsl')"/>
        </td>
    </tr>
</table>
</body>
</html>

```

AppNavigation.mwsl

Call:

```

<MWSL>
    WriteXMLData("<EXTERNAL SRC=\" + GetVar("XML", "URL") + "\"/>",
        "<TEMPLATES><EXTERNAL SRC=\" +
        GetVar("TEMPLATE", "URL") + "\"/></TEMPLATES>");
</MWSL>

```

When the call is made, the data and the template file are transferred from the URL.

In this example, the transferred template file is assumed to be AppNavigation.xml and the transferred data file MWSLTestMenu.xml.

MWSLTestMenu.xml

```

<?xml version="1.0" ?>
<APPMENU>
    <MENU>
        <BUTTON NAME = "Menu1"
            LABEL = "Variablentest"
            TITLE = "Tooltip"
            STATUS = "Statusline"
            CLIENTAREA = "/MWSL/Variablentest.mwsl?Parameter=4711"/>
        />
        [...]
        <BUTTON NAME = "Menu9"
            LABEL = "Versuch"
            TITLE = "Tooltip"
            STATUS = "Statusline"

```



```

        CLIENTAREA = "/MWSL/Versuch.mwsl" />
    </MENU>
</APPMENU>

```

In the following template file, some attributes are queried as to their existence and the corresponding lines executed.

In this case, only the condition for `CLIENTAREA` is run since no other queried attribute is present.

AppNavigation.xml

```

<?xml version="1.0" standalone="yes"?>
<TEMPLATES>
    <TEMPLATE NAME="BUTTON">
        <POSITION NAME="LINE">
            <![CDATA[
                <TR>
                    <TD>
                        <INPUT CLASS = "MainMenu"
                            TYPE = "BUTTON"
                            TITLE = "<MWSL>TITLE ()</MWSL>"
                            VALUE = "<MWSL>LABEL ()</MWSL>"
                        <MWSL>
                            if ( ExistVariable( "CLIENTAREA" ))
                            {
                                write("OnClick=
                                    \"top.ClientArea.window.navigate('\"+
                                        GetVar("CLIENTAREA") + "')\"");
                            }
                            if ( ExistVariable ( "TOP" ))
                            {
                                write("OnClick = \"top.window.navigate('\" +
                                    GetVar("TOP") + "')\"");
                            }
                            if ( ExistVariable ( "WIN" ))
                            {
                                write("OnClick = \"window.open('\" +
                                    GetVar("WIN") + "')\"");
                            }
                            if ( ExistVariable ( "ACTION" ))
                            {
                                write("OnClick = \"top.ClientArea.window.\" +
                                    GetVar("ACTION") + "\"");
                            }
                        </MWSL>
                    </TD>
                </TR>
            ]]>
        </POSITION>
    </TEMPLATE>
</TEMPLATES>

```

The template file contains `if` conditions, in which attributes from the data file are queried.

This results in different statements for each data node during runtime.

The generated HTML file will then only contain the line corresponding to the respective data node.

generierte Ausgabe

```
<TR>
  <TD>
    <INPUT CLASS = "MainMenu"
           TYPE = "BUTTON"
           TITLE = "Tooltip"
           VALUE = "Variablentest"
           OnClick = "top.ClientArea.window.navigate(
                     '/MWSL/Variablentest.mwsl?Parameter=4711') "
    />
  </TD>
</TR>
[... ]
<TR>
  <TD>
    <INPUT CLASS = "MainMenu"
           TYPE = "BUTTON"
           TITLE = "Tooltip"
           VALUE = "Versuch"
           OnClick =
             "top.ClientArea.window.navigate('/MWSL/Versuch.mwsl') "
    />
  </TD>
</TR>
```

3.1.9 Server Side Includes (SSI)

3.1.9.1 Integration of process values

You can include process values in the user-defined HTML pages using Server Side Includes (SSI).

Note

In SIMOTION controls V4.1 and higher, HTML pages with SSI must be available as a binary file to display process values. A standard HTML page can be converted to a binary file using the supplied conversion tool (Page 14).

HTML pages with static content only do not have to be converted.

Integration of process values

The variables are integrated in the HTML page using the `<%=IDENTIFIER %>` character string. `IDENTIFIER` is a placeholder, which you must replace with variables from the variable providers. For example, the variable `<%=DeviceInfo.Board%>` returns the name of the control. On a D435, for example, the value is "D435".

Details of the variables and syntax can be found in the *Variable providers* chapter of the *SIMOTION IT Diagnostics and Configuration* manual.

The source text below shows an example for integrating the variable `userData.user1`. First, the value of the variable is output (system variable `userData.user1: <%=var/userData.user1 %>`). The value of the variable is used as a default in the input field and can be overwritten by a user input.

```
<html>
  <head>
    <title>Demo Seite</title>
  </head>
  <body text="#000000" bgcolor="#FFFFFF" link="#FF0000"
  alink="#FF0000" vlink="#FF0000">
    Demoseite<br>
    Systemvariable userData.user1 : <%= var/userData.user1 %> <br>
    <form method="post" action="/VarApp">
      SIMOTION C: userData.user1:
      <input type="TEXT" name="var/userData.user1" value="<
      %= var/userData.user1 %>" />
      <input type="submit" value="Wert schreiben" />
    </form>
  </body>
</html>
```

3.2 OPC XML-DA web service

3.2.1 Web services introduction

A SIMOTION IT web service supports application programs in text-based access to process values. The application program performs strictly symbolic access independently of the programming language, the operating system, and the time of programming. Details of the international standard of web services can be researched on the Internet or in technical literature.

The SIMOTION control offers the OPC XML-DA web service for data access in accordance with the OPC XML-DA definition V1.01 of the OPC Foundation and the Trace Via SOAP web service (TVS) for the use of the SIMOTION Runtime Trace.

- Addon\4_Accessories\SIMOTION_IT\7_Webservices\WSDL\OPC XMLDA 1.01.wsdl
- Addon\4_Accessories\SIMOTION_IT\7_Webservices\WSDL\TVS.wsdl

An example of a client using the OPC XML-DA service is supplied.

- Addon\4_Accessories\SIMOTION_IT\7_Webservices\Example

Web services can also be used with JavaScript in HTML pages. The standard diagnostics pages of SIMOTION IT use these. See Variable access with JavaScript and web services (Page 24)

Required knowledge for programming: OPC XML-DA, web services, XML, SOAP, Javascript/AJAX.

Key words for advanced programming: Structuring of applications with model, view, controller architecture, if, for example, dynamic variable values are to be displayed and updated in the background.

3.2.2 Overview

The SIMOTION IT OPC XML DA server enables access via Ethernet to data and operating modes of the SIMOTION device.

What is OPC XML DA?

OPC stands for Open Connectivity and denotes a standard interface for communication in automation technology

With OPC XML DA, it is possible to communicate with a controller using Ethernet-based standard telegrams.

Commands are transmitted via the SOAP (Simple Object Access Protocol) communication protocol.

The interface is defined in a configuration file using a description language (WSDL) based on XML vocabulary. It describes the format of the HTTP request and response telegrams with which function calls are output (see OPC XML-DA R1.0 Specification: OPC Foundation Download (<http://www.opcfoundation.org/Downloads.aspx>)).

This interface can only be used by client applications.

The figure below shows an example client of the OPC Foundation . The client enables browsing via the system, interface, IO, and global device variables.

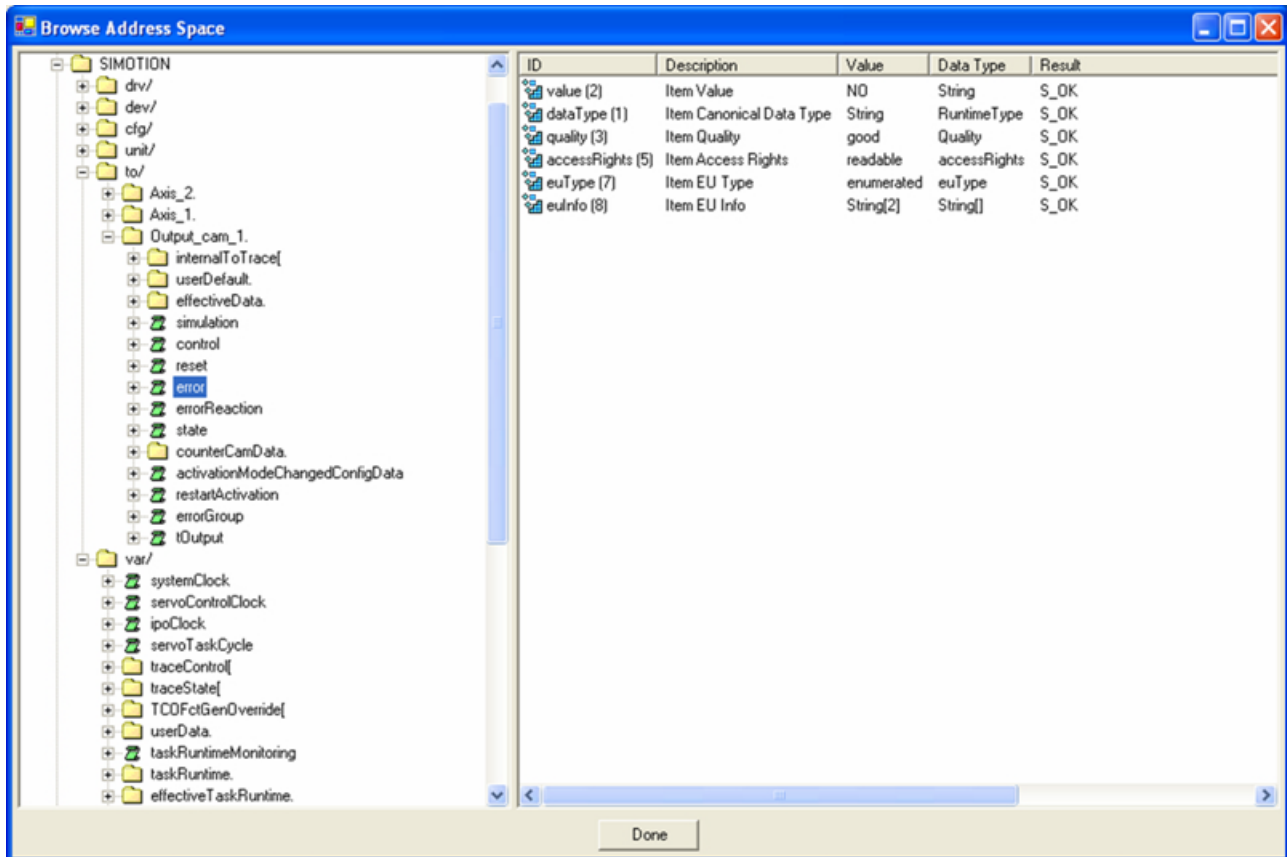


Figure 3-14 OPC Foundation Client

Purpose and benefits

The purpose and benefits of SIMOTION IT OPC XML-DA server are as follows:

- Symbolic access (without project information) to the data of the controller. Only knowledge of the variable names is required.
- Non-dependence on the engineering and project versions. The client can still access the data even if the control program has been modified.
- The server can be addressed by any client application which conforms to the OPC XML-DA V1.0 standard, regardless of its operating system (e.g. Linux).

What previous knowledge is required?

For the user to understand the SIMOTION IT OPC XML-DA server described in this chapter, prior knowledge of the terms associated with OPC XML-DA (see OPC XML-DA R1.0 Specification) is necessary.

3.2.3 Comparison of OPC XML DA / SIMATIC NET OPC DA

Comparison

The "SIMATIC NET OPC Server for SIMOTION" product exists in addition to the SIMOTION IT OPC XML DA server. This package also allows access to data and operating modes of the SIMOTION device via SIMATIC NET OPC DA.

The following table compares the two packages and describes the basic procedure:

Table 3-14 Basic procedure for accessing data

SIMOTION IT OPC XML DA	SIMATIC NET OPC DA
No configuration (OPC export) necessary with SIMOTION SCOUT.	OPC export with SIMOTION SCOUT required, which has to be repeated for every project change.
Symbols are resolved in the SIMOTION device, communication by means of text format (XML).	Symbols are resolved during OPC export and stored in the OPC server on the Windows system in binary format; binary communication -> higher data throughput.
At present only SIMOTION with OPC XML DA. Access to S7 devices not possible at present.	Simultaneous access to SIMOTION and S7 devices.
Client can run on any operating system.	Based on Windows COM/DCOM technology; client and server can only run on Windows operating systems.
Communication using standard protocols (TCP/IP, XML, SOAP) does not require vendor-specific (SIEMENS) tools or drivers on the client system.	S7 protocol used for communication, appropriate manufacturer-specific drivers required on the client.
Communication is only possible via Ethernet (e.g. PROFINET).	Communication is possible via PROFIBUS/MPI and Ethernet (e.g. PROFINET).
Direct addressing via firewalls is possible.	Generally, DCOM communication is not released for firewalls.

3.2.4 Schematic representation of creating the client application

Example arrangement

The figure below shows an arrangement example of the relevant software for the creation of a client application on a PC. The PC and the SIMOTION device are networked via Ethernet.

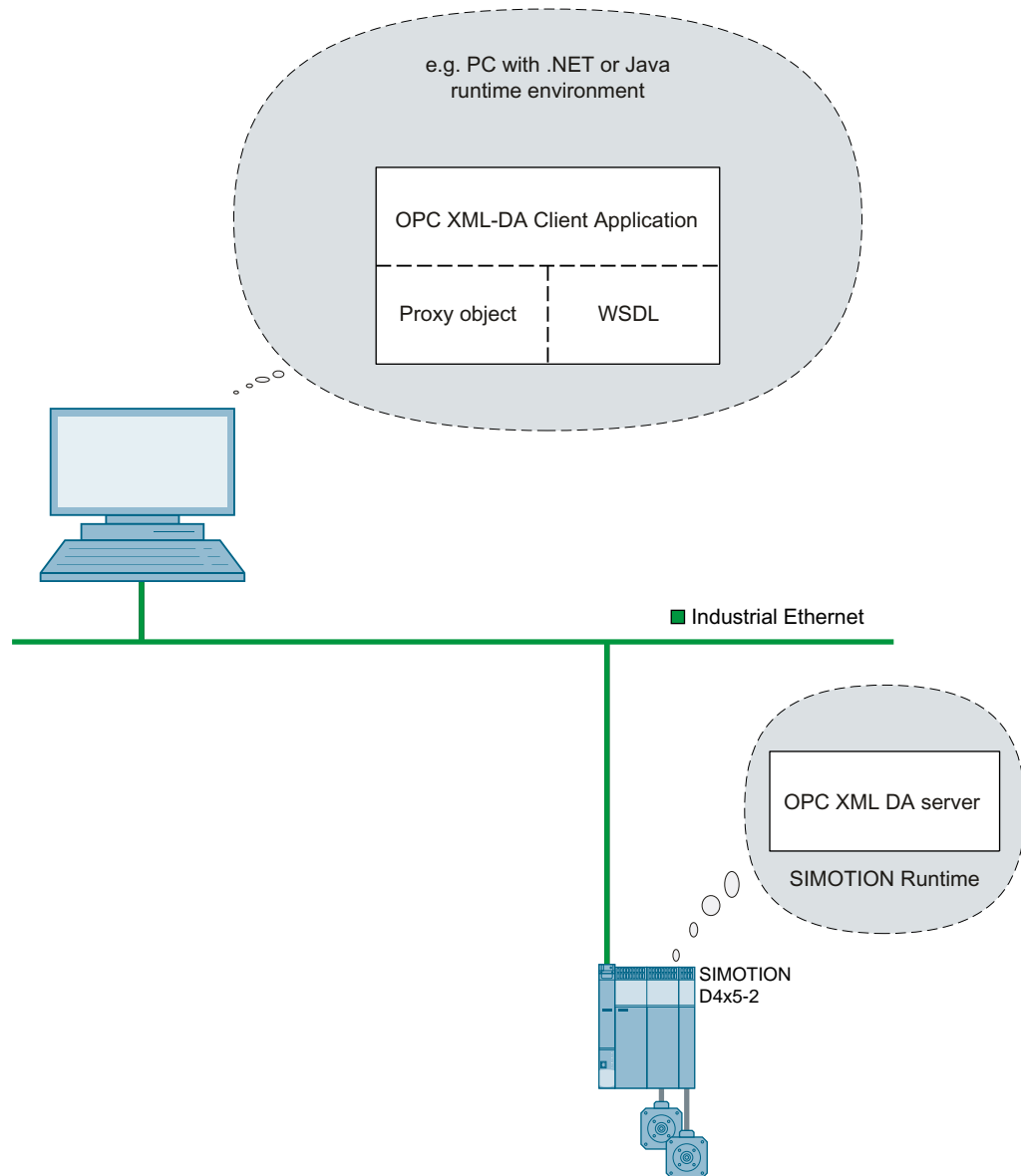


Figure 3-15 Design stage overview (example)

3.2.5 Schematic representation at runtime of the client application

Example arrangement

The figure below shows an example of accessing the OPC XML-DA server of a SIMOTION device via Ethernet during runtime. The example shows other devices connected to the SIMOTION device via PROFIBUS.

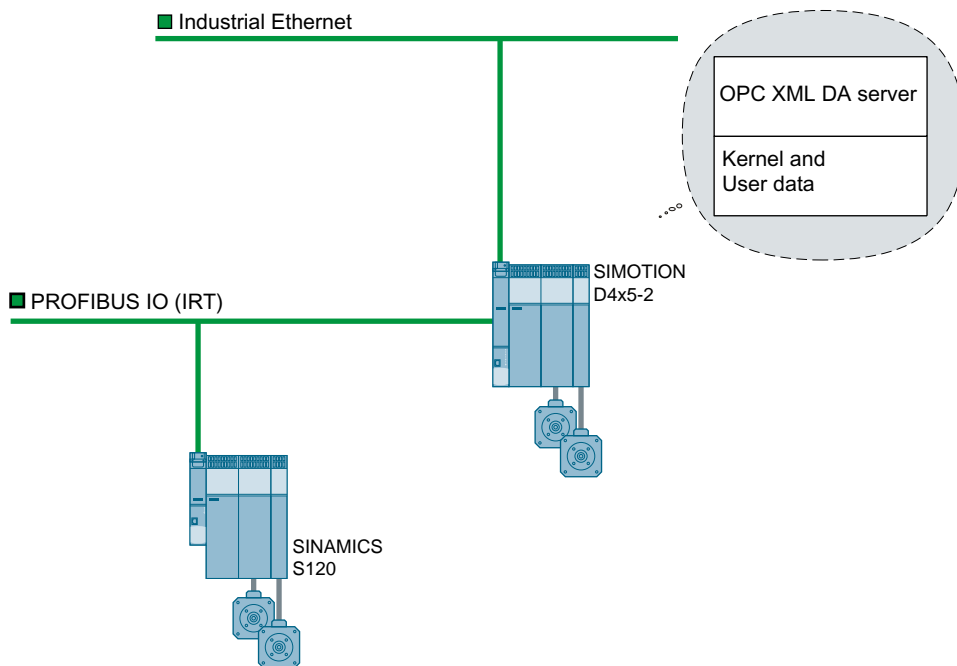


Figure 3-16 Overview at runtime (example)

3.2.6 Installation

3.2.6.1 Hardware and software requirements for creating the client application

Hardware requirements at the design stage

Note

You can freely select the programming environment. The following requirements are examples of Microsoft Visual Studio .NET, but are not mandatory.

Table 3-15 Hardware requirements at the design stage

Feature	Minimum requirement
Processor	Intel Pentium III or compatible, 800 MHz
Main memory	128 MB RAM

Software requirements at the design stage

Note

You can freely select the programming environment. The following requirements are an example for Microsoft Visual Studio .NET, but they are not binding.

- Microsoft Visual Studio .NET:
<http://msdn.microsoft.com/vstudio/> (<http://msdn.microsoft.com/en-us>)
<http://www.microsoft.com/net/> (<http://www.microsoft.com/net/>)
- Configuration file (WSDL), according to OPC XML- R1.0 Specification.

3.2.6.2 Configuring the SIMOTION device interface for using the client application

Configuring the interface

In order to establish a connection between a PC and a SIMOTION device when the system is running, you must carry out the following steps for the configuration of the Ethernet interface:

Table 3-16 Configuring the interface

Step	Procedure
1	You must activate the functionality in the SIMOTION project when configuring the control hardware via the "Ethernet Extended / Webserver" properties in the "Simotion IT - Web server settings" function.
2	The IP address of the SIMOTION control via which the OPC XML-DA server is accessed must be known. The IP address can be configured using the interface settings in HW Config.

3.2.6.3 OPC XML DA access protection

Settings for the OPC XML-DA access protection

WebCfg.xml provides a way of setting up access protection for OPC XML-DA, by means of a REALM.

```
<SOAPAPP>
  <STATIC>
    <!--
      place all statically linked WebServices here
```

```
-->
<WEBSERVICE NAME="OpcXml" URL="/SOAP/OPCXML"
              REALM="OPC_XML_USER_GROUP" />
<WEBSERVICE NAME="TVS" URL="/SOAP/TVS" />
</STATIC>
</SOAPAPP>
```

3.2.7 OPC XML-DA variable access

Access to the OPC XML-DA data uses the same access syntax as that described in the Variable providers chapter of the SIMOTION IT Diagnostics and Configuration manual.

Table 3-17 Examples of accessing variables

Variables	Access syntax
Global device variables	glob/<var name>
I/O variables	io/_direct.<var name> io/_image.<var name> io/_quality.<var name>
Unit (MCC/ST/LAD-FBD)	unit/<unit name>.<var name>

Access to the I/O variables (access syntax)

"_direct" addresses the direct I/O access (current values) of the I/O variables

"_image": addresses the process image of the I/O variables

"_quality": addresses the Quality, i.e. the detailed status of the I/O variables

3.2.8 Example of a client application

Example

An example showing what a minimal client application for an OPC client can look like is included on the AddOn DVD in the \VOL2\Addon\4_Accessories\SIMOTION_IT\7_Webservices\Example directory. The example explains the most important programming steps for the "Read" method with the Microsoft Visual Studio development environment.

The application example displays a **Read** button in a dialog box. When the button is activated, the client connects to the SIMOTION IT OPC XML DA server and reads a variable. The result is displayed in the output field of the dialog box.

The dialog box of the application example is shown in the following figure:

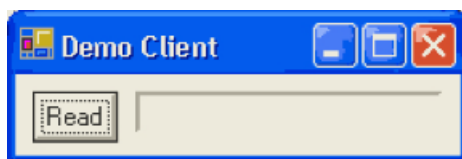


Figure 3-17 Demo client

Programming steps

The following programming steps are needed:

1. Create a new project with Microsoft Visual Studio .NET and import the WSDL file as the interface description ("Add Web Reference" menu).
2. Create a dialog box with a text field and a **Read** button.
3. Enter the name assigned for the reference, e.g. "OPCXMLServer", in the program (using `DemoClient.OPCXMLServer`).
4. Declare the server URL in the program as follows:
`http://<IP address>/soap/opcxml`
Enter the IP address of your SIMOTION control in place of <IP address>.
5. Instantiate the server proxy object in accordance with the code example and provide the call-up with the required parameters.
6. The required data is returned.

3.2.9 SIMOTION IT OPC XML-DA server interface

3.2.9.1 Overview

Introduction

This section describes the methods you can run across the interface to the OPC XML-DA V1.0 server. The server itself is integrated in the SIMOTION device.

For firmware version V4.2 and higher, a license is no longer required for using the OPC XML-DA server.

This is just a brief overview. These methods are described in detail in the document "OPC XML-DA Specification R1.0" of OPC Foundation.

You can find an up-to-date and detailed interface description on the home page of the OPC Foundation: <http://www.opcfoundation.org> (<http://www.opcfoundation.org>)

3.2.9.2 Methods which can be called synchronously

The SIMOTION IT OPC XML DA server provides the following methods, which can be called synchronously, under the "OpcXmlDaService" type:

Description of methods

Browse

The "Browse" method allows you to navigate through the available variables.

GetProperties

The "GetProperties" method can query the settings for a specific variable (e.g. access rights, time stamp, data type).

GetStatus

The "GetStatus" method supplies information about the server status, the program version and the supported interface version.

Read

The "Read" method reads out variable lists.

Subscribe

The "Subscribe" method passes a list of variable names and receives a handle for the subscription. This handle can be used in the SubscriptionPolledRefresh method to poll the values of the previously defined variables again. See Basics of subscriptions (Page 97).

The buffering property, which corresponds to the "EnableBuffering" attribute, is not supported. This has an effect on the "SubscribeRequestItem" and "SubscribeRequestItemList" methods.

SubscriptionPolledRefresh

The "SubscriptionPolledRefresh" method returns the values of the variables written beforehand using the Subscribe method. The handle specifying the subscription is used as a parameter.

The "Holdtime" parameter defines the earliest possible response time. This limits the frequency of data transmission.

The "ReturnAllterms" parameter determines how the "WaitTime" parameter is used.

- True
"WaitTime" is ignored, all requested values are returned immediately.
- False
For the period set in the "WaitTime" parameter, the server checks whether one of the requested values has changed since the last call.
If the specified time expires without a value having been changed, an empty response is returned.
If values change during the specified time, the changed values are returned immediately and the polling ended.

SubscriptionCancel

The "SubscriptionCancel" method cancels the subscription and returns the subscription handle.

Which subscription is to be canceled, must be specified at the call.

If an asynchronous call form is used, the client is informed later of which subscription has been canceled, via a client handle.

Note

Once the subscription has been canceled, the subscription handle ceases to be valid for the client.

Write

The "Write" method writes variable lists.

3.2.9.3 Access to variables

Variable access using methods

Variables can be accessed via the methods which can be called synchronously and asynchronously.

Note

Information on accessing the variables of the different providers can be found in the *Variable providers* chapter of the *SIMOTION IT Diagnostics and Configuration* manual.

To make variables available on the SIMOTION IT OPC XML DA server, you have to declare them as VAR_GLOBAL. See the *Making unit variables available* chapter of the *SIMOTION IT Diagnostics and Configuration* manual.

3.3 Trace Interface via SOAP (TVS) web service

3.3.1 Trace overview

Introduction

The SOAP-based service provides a trace service option.

WebTrace uses the same runtime mechanisms as SIMOTION SCOUT Trace. The useful options are described in the chapters *Trace (device trace)* and *Trace (system trace)* of the *SIMOTION IT Diagnostics and Configuration* manual.

Trace-Service

The "Trace Interface via SOAP" web service enables variable values to be written to a buffer. The values are packed in files and can be retrieved asynchronously via an HTTP request.

This interface can only be used by client applications. The client enables the time characteristic of variables to be traced.

A WSDL file is available for creating the application.

3.3.2 Trace sequence

Introduction

When working with a trace, the trace can assume various states. The following graphic shows the possible states and transitions. The methods named are described in chapter "Trace interface".

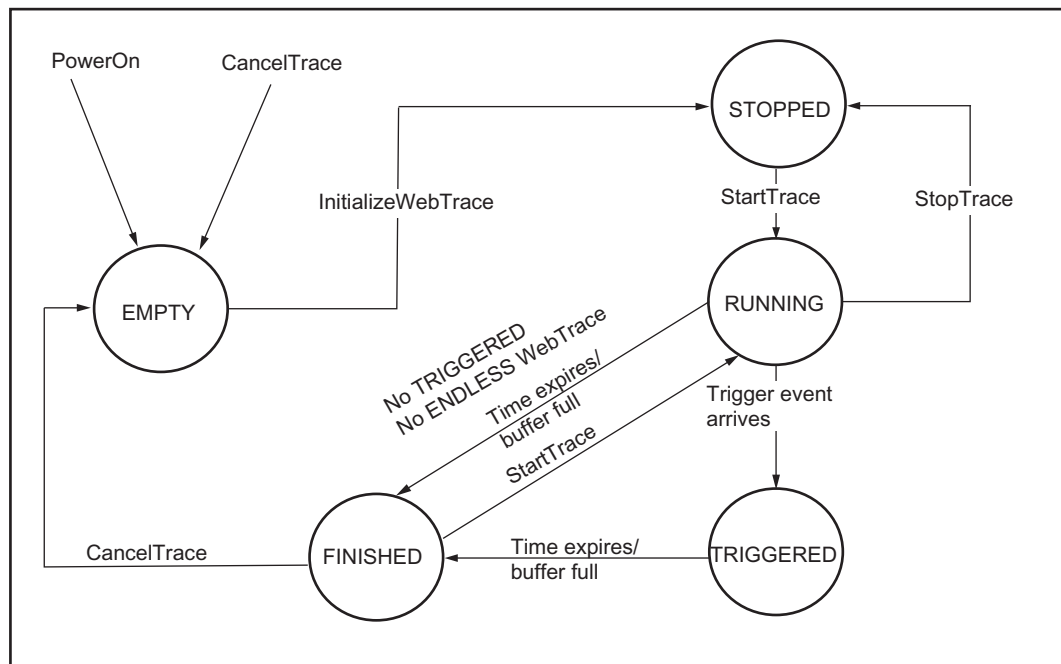


Figure 3-18 WebTrace

States

After a trace has been created with "InitializeWebTrace", this trace is in the STOPPED state. With "StartTrace", a trace starts up and writes the desired data to the buffer. Accordingly, a trace can be stopped again with "StopTrace". After the start, a trace switches to the RUNNING state. If the time specified in the call has expired, a trace assumes the FINISHED state. A trace can be deleted at any time with "CancelTrace" in order to create a new trace, for example.

3.3.3 Procedure/terms

HTTP methods - data exchange

The trace data are stored as compiled data on the RAM disk using the ReadData method. This data must be retrieved via ordinary HTTP requests.

Note

The data are not deleted by retrieval alone! To prevent the RAM disk from overflowing, an HTTP DELETE call to this URL must follow an HTTP GET call. (Reason: The use case under consideration is one in which a client may have to request the same trace data more than once, e.g. to compare traces that have already been executed.) These temporary data are completely deleted only after a CancelTrace operation, regardless of whether they have already been retrieved or not.

TRIGGERED

The trace offers a triggering option. Depending on the trigger method, different constants or variable symbols must be specified for this. The trace starts with:

- A rising edge (RE),
if the variable exceeds the value of a constant.
- A falling edge (FE),
if the variable falls below the value of a constant.
- Within a tolerance band (WIB),
if the variable lies between two constants.
- Outside of a tolerance band (OOB),
if the variable lies outside of a tolerance band.
- Bit mask has value (BHV),
if the variable has a specified value after masking with a constant.

If the trace is set up in TRIGGERED mode, a trigger condition as described below must be specified. This trigger acts as a SingleShot. However, the MatchCountTriggerPoint parameter can be used to set the trigger for repeated occurrences (e.g. five: start trace/data recording only on the fifth time appearance of the trigger condition).

In this case, the trace takes place only after the trigger. The data are recorded for the duration specified during setup.

IMMEDIATE / ENDLESS

The counterpart to the TRIGGERED trace is the IMMEDIATE trace, which begins the trace immediately after the "StartTrace" call has occurred. In this case as well, the data are recorded for the duration specified during setup.

The ENDLESS Trace uses a ring buffer trace. Trigger conditions are not evaluated. ENDLESS Trace starts as soon as the StartTrace event arrives. However, it is terminated only when StopTrace is called explicitly. The size of the ring buffer must also be specified using the

duration for the initialization call. Thus, an appropriate value must be found that uses fewer resources, but is sufficient to retrieve data in a timely manner via HTTP.

The size of the ring buffer (B) is determined from the number of variables (N), their size (S), the transferred time duration (t) and the cycle clock (T) in which they are traced.

$$B = t/T * \sum_{i=0}^{n-1} S^i$$

Within the transferred time duration, the buffer must be discharged at least once by calling the "readData" function in order to prevent the oldest trace data from being overwritten each time.

If the parameters require a larger memory area, the recording duration is reduced such that no more than the available memory space is occupied.

512 KB is available for SIMOTION C, SIMOTION D410-2, and 1024 KB is available as ring buffer for all other SIMOTION modules.

3.3.4 Error handling

All implemented methods of the TVS (trace via SOAP) supply either the requested data or status information, or an SOAP_FAULT. This behavior enables the use of the SoapFaultError in the .NET framework. The Try-Catch mechanism enables convenient error handling.

3.3.5 Basics of subscriptions

Introduction

"GetStatus" must be called in order to query the status of a trace. The fastest possible detection of a status change requires extremely frequent polling, which places an unnecessary load on the CPU in the control and causes heavy traffic on the network.

To optimize this operation, OPC XML DA provides "subscriptions". With subscriptions, a query does not receive a response until the required variable changes or a timeout occurs. Thus, the connection is kept open without causing traffic. As soon as relevant data are available for the client, these data are sent to the client.

The trace supports this mechanism via the SOAP web service also. However, in this case, only the status of the trace object is checked, as this is the only valuable information in this environment.

As soon as the status changes (e.g. RUNNING -> FINISHED), the clients that issued the query receive a response accordingly. In essence, any number of clients is possible (as long as there are sufficient resources).

Operational sequence

The operational sequence of a subscription is as follows:

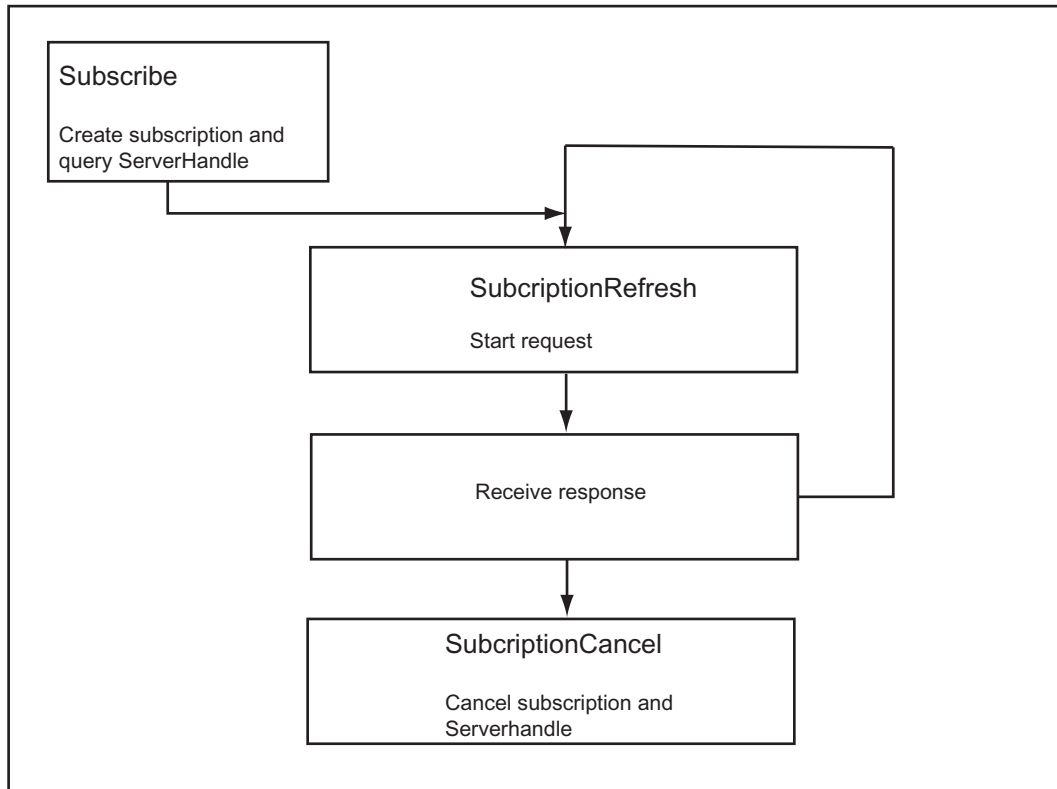


Figure 3-19 Subscription

First, a subscription must be created. It is answered with a unique ServerHandle, which is required for further communication.

SubscriptionRefresh can be called as often as necessary to start a new query. This request receives two time specifications in milliseconds as parameters:

- **HoldTime:**
This time indicates the minimum hold time for the response, irrespective of whether the status has changed.
- **WaitTime:**
The WaitTime begins after the HoldTime has expired. If the trace status has changed, the response to the current status is sent immediately. If there is no change, the response is sent once the WaitTime has expired.

The exact method calls are explained in the next section.

3.3.6 Interface

3.3.6.1 Global definitions

TraceStateEnum

Enumerator that indicates the status of the trace object.

Declaration:

```
public enum TraceStateEnum
{
    RUNNING,
    STOPPED,
    ERROR,
    EMPTY,
    FINISHED,
    TRIGGERED
}
```

TraceDataCycleEnum

Enumerator that specifies the cycle clock in which the data are to be traced. It must be noted here that large traces may cause a level overflow.

Declaration:

```
public enum TraceDataCycleEnum
{
    IPO1,
    IPO2,
    SERVO
}
```

Structure VDSC

Structure that contains information about the traced variables. These are:

- Variable name `VarName`
- Variable type `VarType` in S7 notation (e.g. DINT or BYTE)
- `VarOffset` specifies the offset of the variable within the data stream (relative to the start of the I/O container)
- Variable length `VarLen` (optional)

Declaration:

```
public class VDSC
{
    public string VarName;
    public string VarType;
    public System.UInt32 VarOffset;
    public System.UInt32 VarLen;
}
```

TriggerCondition

Structure indicating the trigger of a trace. The TriggerCondition contains the variable to be compared in symbolic names according to VarProvider notation.

- Variable refers to the variable for comparison.
- Constant1 is a constant which is set according to the trigger type.
- Constant2 is a constant which is set according to the trigger type.
- Operation indicates the comparison type as enumerator TriggerOperationType.
- MatchCountTriggerPoint indicates how many times the trigger condition must apply before the trigger is activated and data tracing starts.
- GlobalTriggerID is optional and contains the unique ID, which is generated by the browser when a distributed trace is set up.

Declaration:

```
public class TriggerCondition
{
    public string Variable;
    public string Constant1;
    public string Constant2;
    public TriggerOperationType Operation;
    public System.UInt32 MatchCountTriggerPoint;
    public System.UInt32 GlobalTriggerID;
}
```

For this purpose, the comparison type:

Call:

```
public enum TriggerOperationType {
    RE,
    FE,
    WIB,
```

```

OOB,
BHV
}

```

The following table gives an overview of various possible combinations of trigger types and constants.

Trigger type	Symbol	Description	Constant1	Constant2
Rising Edge	RE	Triggered if variable exceeds the value of Constant1 in positive direction	Limit exceeded	-Not used-
Falling Edge	FE	Triggered if variable exceeds the value of Constant1 in negative direction	Limit exceeded	-Not used-
Within a tolerance Band	WIB	Triggered if variable is located within the interval spanned by Constant1 and Constant2	Lower limit	Upper limit
Out of tolerance Band	OOB	Triggered if variable is located outside the interval spanned by Constant1 and Constant2	Lower limit	Upper limit
Bit pattern	BHV	The bit pattern triggers if the relevant bit is 1 both in the bit mask and in the comparison result. ((v&c1)==c2)	Bit mask	Result of comparison

Overview of trigger types and constants

Enumerator that determines the type of trace.

Call:

```

public enum TraceStartTypeEnum
{
    IMMEDIATE,
    ENDLESS,
    TRIGGERED
}

```

3.3.6.2 Methods

StartTrace

The `StartTrace` method starts an initialized trace. The SoapFault "No Trace available" is returned if a trace has not yet been initialized. `StartTrace` is ignored (with a positive result) if the trace is already in progress.

The `GlobalTriggerID` must be transferred for a distributed trace. The trigger ID is a unique ID, which unambiguously identifies the station currently responsible for triggering; the same ID is used for all stations.

When a trace is restarted, the trigger ID must be reassigned so that the trigger events can be distinguished from one another.

Call:

```
public TVS_Client.TVS.StartTrace_Response StartTrace
(
    uint GlobalTriggerID
)

public class StartTrace_Response
{
    public TraceStateEnum TraceState;
}
```

StopTrace

The `StopTrace` method stops a trace in progress. The "No Trace available" SoapFault is returned when a trace has not yet been initialized. This is ignored (with a positive result) if the trace has already stopped.

Call:

```
public TVS_Client1.TVS.StopTrace_Response StopTrace ( )

public class StopTrace_Response
{
    public TraceStateEnum TraceState;
}
```

CancelTrace

The `CancelTrace` method deletes an active trace. The traces switches to EMPTY status, and all trace data are deleted. (Note: Data blocks of the `WebTrace` that have been requested but have not yet been retrieved are also deleted (see `WebTrace::ReadData()`)

The "No Trace available" SoapFault is returned when a trace has not yet been initialized.

Call:

```
public TVS_Client.TVSI0.CancelTrace_Response CancelTrace ( )
```

```
public class CancelTrace_Response
{
    public TraceStateEnum TraceState;
}
```

GetStatus

The `GetStatus` method returns the current status of the trace. When a trace object is deleted or has become invalid, then `TraceIsValid` will contain "false". In this case, the trace must be deleted via `CancelTrace`.

Call:

```
public TVS_Client.TVSIIO.GetStatus_Response GetStatus ( )

public class GetStatus_Response
{
    public bool TraceIsValid;
    public TraceStateEnum TraceState;
}
```

ReadData

The `readData` method saves trace data on the RAM disk and supplies the URLs of the files in the return value. These data can be retrieved from the client with an HTTP-GET request.

The "No Tracedata available" `SoapFault` is returned if no trace data are available.

Call:

```
public ReadData_Response ReadData()

public class ReadData_Response
{
    public TraceStateEnum TraceState;
    public string[] URL;
}
```

InitializeWebTrace

A trace is created with `InitializeWebTrace`. `VariablesToTrace` is the list of symbolic names in accordance with `VarProvider` notation. `TraceDataCycle` determines the cycle clock in which the data is to be recorded. `TraceStartType` determines the type of trace. `Duration` specifies the duration of the recording in milliseconds. With an endless trace, this parameter specifies the size of the ring buffer in milliseconds.

MatchCountTriggerPoint in TriggerInformation determines how often the trigger must occur before it actually performs a trigger operation and, as a result, starts the recording. Pretrigger specifies the number of values which are to be recorded prior to triggering ("history").

Call:

```
public InitializeWebTrace_Response InitializeWebTrace
(
    string[] VariablesToTrace,
    TraceDataCycleEnum TraceDataCycle,
    TraceStartTypeEnum TraceStartType,
    uint Pretrigger,
    uint Duration,
    TriggerCondition TriggerInformation,
    string[] DevicesInvolved
)

public class InitializeWebTrace_Response
{
    public VDSC[] CurrentlyTracedVariables;
    public TraceStateEnum TraceState;
    public string UID;
    public string[]DevicesInvolved;
}
```

InitializeWebTraceEx

InitializeWebTraceEx is identical to InitializeWebTrace apart from the return value, where the variables are sorted according to their particular offset.

GetTraceParameters

GetTraceParameters can be used to read out an existing trace configuration.

Only one WebTrace is returned (if one actually exists).

Call:

```
public GetTraceParameters_Response GetTraceParameters
(
    string UID
)

public class GetTraceParameters_Response
{
```



```

public TraceTypeEnum TraceType;
public VDSC[] CurrentlyTracedVariables;
public TraceStateEnum TraceState;
public TraceDataCycleEnum TraceDataCycle;
public string UID;
public TraceStartTypeEnum TraceStartType;
public unsignedInt Pretrigger;
public unsignedInt Duration;
public TriggerCondition TriggerInformation;
public unsignedInt IOContainerOffset;
public unsignedInt IOContainerLength;
public hexBinary ClientHandle;
public string[] DevicesInvolved;
}

```

EnableTrigger

Only for the distributed trace.

When a distributed trace has been set up and started, this activates the triggers. The `TriggerID` must be unique so that the stations can differentiate between them and do not lose their way on the network. The sequence is important: All traces should be running before the trigger is activated, to avoid loss of trigger events.

Return value: `TriggerState` activated or not activated.

Call:

```

public EnableTrigger_Response EnableTrigger
(
    uint TriggerID
)

public class EnableTrigger_Response
{
    public TraceStateEnum TraceState;
}

```

ReadData

With `ReadData`, the TVS service is requested to read out the trace buffer and pack the data in temporary files. These can then be accessed via HTTP under the relative paths specified in URL. If the buffer is empty, a response is made to the request with the "No Tracedata available" `SoapFault`. Currently, a maximum of 8 compiled files with a maximum of 8,192 recording points are provided for each request.

ReadDataArchive

If a trace is in the STOPPED state, this function can be used to request the recorded data.

The function supplies a URL, from which a WTRC data archive can be downloaded and then displayed in the WebTraceViewer.

Note

The WTRC file is deleted as soon as it has been downloaded.

Call:

```
public ReadDataArchive_Response ReadDataArchive ()

public class ReadDataArchive_Response
{
    public TraceStateEnum TraceState;
    public string URL;
}
```

ReadDataArchives

Only for the distributed trace.

ReadDataArchives automatically retrieves the trace data for all stations involved in a distributed trace and combines them in a WTRC file. The URLField array is used to transfer a list of the URLs for all the stations involved in the trace. The trace must be in the STOPPED state in order to use this method.

The return value is identical to that for ReadDataArchive.

Call:

```
public ReadDataArchives_Response ReadDataArchives
(
    public string[] URLField;
)

public class ReadDataArchives_Response
{
    public TraceStateEnum TraceState;
    public string URL
}
```

3.3.6.3 Subscriptions

Introduction

The subscription methods are listed below.

Subscribe

A subscription is created using the `Subscribe` method. The response is a `ServerHandle` that can be used to uniquely identify a subscription operation. In addition, the current `TraceStatus` is supplied.

Call:

```
public TVS_Client.TVS.Subscribe_Response Subscribe ( )

public class Subscribe_Response {
    public System.UInt32 ServerHandle;
    public TraceStateEnum TraceState;
}
```

SubscriptionRefresh

`SubscriptionRefresh` is used to query the status of the trace again; the trace was previously queried with the `Subscribe` method. The server response is received after `HoldTime` (milliseconds) + `WaitTime` (milliseconds), if the status has not changed during this time, or the response is received (at the earliest) after the `HoldTime` has expired and before the `WaitTime` has expired, if the status of the trace changes during the `WaitTime`. As such, the response is never expected before the `HoldTime` has elapsed.

In the response, `StateChanged` indicates whether the status has changed between request and response (true) or whether the `TraceState` status matches the status during the request (false = `WaitTime` expired).

Call:

```
public TVS_Client.TVS.SubscriptionRefresh_Response SubscriptionRefresh (
    System.UInt32 ServerHandle ,
    System.UInt32 WaitTime ,
    System.UInt32 HoldTime
)

public class SubscriptionRefresh_Response {
    public bool StateChanged;
    public TraceStateEnum TraceState;
}
```

SubscriptionCancel

With `SubscriptionCancel`, a subscription is canceled and the resource is enabled. The response indicates whether the Cancel operation was successful. Any current `SubscriptionRefreshes` are cancelled and responses sent immediately.

Call:

```
public TVS_Client.TVS.SubscriptionCancel_Response SubscriptionCancel (
    System.UInt32 ServerHandle )

public class SubscriptionCancel_Response {
    public bool SubscriptionCanceled;
}
```

Appendix

4.1 MWSL functions

4.1.1 AddHTTPHeader

Syntax	<p>AddHTTPHeader (<Http-Header>)</p> <p>This command can be used to add HTTP headers from MWSL . These are then not transmitted as part of the document but rather in the protocol portion of HTTP.</p> <p>The AddHTTPHeader command must therefore come before the HTML tag of a page.</p> <p>However, it is important to make sure that no MWSL functions that result in output into the page are used before the HTML tag.</p>	
Parameters	<Http-Header>	<p>Character string that ends with <code>\r\n</code>.</p> <p>If multiple HTTP headers are to be entered (only possible with <code>Set-cookie</code>), the individual headers must be separated by <code>\r\n</code>.</p>
MWSL example	<pre> <MWSL> var strCookie; strCookie = "Set-cookie: siemens_automation_language=de"; AddHTTPHeader(strCookie); </MWSL> <html> <head> <title> MWSL Function AddHTTPHeader </title> </head> <body style="background-color:#DCDCDC"> <h2>Testpage</h2> </body> </html> </pre>	

4.1.2 createGUID

Syntax	<p>String createGUID()</p> <p>Generates a unique alphanumeric ID in the system.</p>	
Parameters		

Example	<pre><MWSL> write(createGUID()); </MWSL></pre>
Output	5022420B-02A7-0000-B362-3B7F4E87148D
Valid as of	V4.4

4.1.3 DecodeString

Syntax	string DecodeString(<string>) Converts a string encoded with EncodeString back to its original.	
Parameters	<string>	String in which URL-coded special characters are converted back to normal characters.
Example	<pre><MWSL> var tmpString = "Straße Flüsse Gelände Vögel"; write("Original: " + tmpString + "
"); var tmpEncodedString = EncodeString(tmpString); write("Encoded: " + tmpEncodedString + "
"); var tmpDecodedString = DecodeString(tmpEncodedString); write("Decoded: " + tmpDecodedString); </MWSL></pre>	
Output	Original: Straße Flüsse Gelände Vögel Encoded: Straße Flüsse Gelände Vögel Decoded: Straße Flüsse Gelände Vögel	
Valid as of	V4.4	

4.1.4 die

Syntax	die(<Param0>,<Param1>,...) Break program execution.	
Parameters	<Param0>,<Param1>,...	Concatenation and output of the parameters.
Example	<pre><MWSL> function dieTest() { write("Is there a life after die?"); die("--- ",123," ---"); }; dieTest(); write("There is a life after die"); </MWSL></pre>	
Output	Is there a life after die?--- 123 ---	
Valid as of	V4.4	

4.1.5 EncodeString

Syntax	string EncodeString(<string> Replaces special characters by their URL-coded hex value (%hh).	
Parameters	<string>	String in which the replacement will be performed.
Example	<pre><MWSL> var tmpString = "Straße Flüsse Gelände Vögel"; write("Original: " + tmpString + "
"); var tmpEncodedString = EncodeString(tmpString); write("Encoded: " + tmpEncodedString + "
"); var tmpDecodedString = DecodeString(tmpEncodedString); write("Decoded: " + tmpDecodedString); </MWSL></pre>	
Output	Original: Straße Flüsse Gelände Vögel Encoded: Straße Flüsse Gelände Vögel Decoded: Straße Flüsse Gelände Vögel	
Valid as of	V4.4	

4.1.6 ExistFile

Syntax	long ExistFile(<file name> Checks whether a file with the name <file name> exists. The function returns the file length as the returned value. Special aspect due to the MWSL Compiler: Because all files that are associated with the MWSL Compiler (*.mws, *.msl, *.xml, *.js, *.xmlf, *.css) are only present on the memory card in a compiled form, a ExistFile call must always be made to the compiled file (*.cms) in this case.	
Parameters	<file name>	Name of the sought file. The file path refers to the root directory of the user USER/SIMOTION/HMI.
Example	<pre><MWSL> var tmpLength = ExistFile("/files/test.mws.cms"); write("File length:"+ tmpLength); </MWSL></pre>	
Output	File length: 38	
Example	<pre><MWSL> var tmpLength = ExistFile("/files/mydata.txt"); write("File length:"+ tmpLength); </MWSL></pre>	
Output	File length: 22	
Valid as of	V4.4	

4.1.7 ExistVariable

Syntax	<pre>bool ExistVariable(<Variable Name>, <Variable Source>)</pre> <p>This function queries the presence of a variable. It returns <code>true</code> or <code>false</code>.</p>	
Parameters	<variable name>	Variable name
	<variable source>	See GetVar (Page 112)
Example	<pre>ExistVariable("DeviceInfo.BZU", "PROCESS")</pre> <p>If the process variable "DeviceInfo.BZU" exists, <code>true</code> is returned, otherwise <code>false</code>.</p>	

See also

Conditional operations (Page 61)

4.1.8 GetLanguage

Syntax	<pre>String GetLanguage()</pre> <p>Returns the currently set language.</p> <p>The return value depends on whether one of the following values is found in the displayed sequence:</p> <ol style="list-style-type: none"> 1. Content of the cookie <code>siemens_automation_language</code> if the cookie exists. 2. Value of the HTTP header <code>Accept-Language</code> if the value exists. 3. Return of the <code>SERVEROPTIONS</code> value <code>language</code> set in <code>WebCfg.xml</code>. 	
Parameters		
Example	<pre><MWSL> write("The currently set language is'" + GetLanguage() + "'"); </MWSL></pre>	
Output	The currently set language is 'de'	
Valid as of	V4.4	

4.1.9 GetVar

Syntax	<pre>GetVar(<Variable Name>, <Variable Source>, <Format String>);</pre> <p>This function returns the value of a variable from a variable source.</p> <p>If a parameter does not exist, "null" will be returned.</p>	
Parameters	<variable name>	Variable name

	<variable source>	<p>Name of variable source</p> <p>Valid values:</p> <ul style="list-style-type: none"> • URL • HTTP • PROCESS Read variables from the providers. • COOKIE Read variables from the HTTP header of the cookie • XML • SENDPAGE • DEFAULT The default setting is PROCESS. <p>If not source is stated, DEFAULT is selected, that is, the variable provider.</p> <p>The name of the variable providers, such as SIMOTION, MINIWEB, etc., are not designations for variable sources.</p> <p>The suitable provider is searched for in the web server based on the variable name.</p>
	<Format String>	<p>The handling of the format string depends on the variable source.</p> <p>Thus, this property is not possible for the variable sources COOKIE and URL.</p> <p>Syntax of an HTTP variable: Variables and HTTP header information (Page 57)</p> <p>Syntax of a process variable: Global variables (Page 52)</p>
Example	<p><code>GetVar ("var/userData.user1");</code> Returns the content of the variable <code>var/userData.user1</code>.</p> <p>Because PROCESS is the default variable source, the result corresponds to that of the following call.</p> <p><code>GetVar ("var/userData.user1", "PROCESS");</code> Returns the content of the variable <code>var/userData.user1</code>, the variable source PROCESS.</p> <p><code>GetVar ("Parameter", "URL");</code> Returns the content of the <code>Parameter</code> variable from the URL.</p> <p><code>GetVar ("Accept-Language", "HTTP", "?-")</code> Returns the content of the HTTP variable <code>Accept-Language</code>.</p> <p>The format string <code>"?-"</code> indicates that all characters up to the first occurrence of the <code>"-"</code> character will be returned.</p> <p><code>GetVar ("var/userData.user1", "PROCESS", "[2,3]");</code> Returns three characters, starting from position 2 of the process variable <code>var/userData.user1</code>. The result is characters 2-5 of the process variable.</p> <p><code>GetVar ("Accept-Language", "HTTP", "[3,0]")</code> Returns the content of the HTTP variable <code>Accept-Language</code> starting from the third character.</p>	

4.1.10 InsertFile

<p>Syntax</p>	<p>InsertFile(<File name>)</p> <p>This command allows an existing text file to be imported individually.</p> <p>The text file is interpreted before insertion with MWSL and embedded into the existing source text at the insertion point in the target file.</p> <p>If the file has an ending (*.mws, *.m, *.xsl, *.js, *.xmlf, *.css) associated with the MWSL compiler, the MWSL scripts it contains will be run.</p> <p>URL parameters can be passed with usual syntax (<file name>?<parameter>=<value>).</p>	
<p>Parameters</p>	<p><file name></p>	<p>Name of text file, including path.</p>
<p>Example</p>	<pre> <HTML> <HEAD> </HEAD> <BODY> [...] <table> [...] <tr> <td> An HTML file is now displayed on the right-hand page. </td> <td>
 </td> <td> <MWSL> if(ExistFile("/FILES/TMPL/Output.mws.cms") > 0) { InsertFile("/FILES/TMPL/Output.mws?myparam=123"); } </MWSL> </td> </tr> [...] </table> [...] </BODY> </HTML> </pre> <p>In the right-hand column of the table, the content of the file <code>Output.mws</code> is inserted and displayed in HTML format.</p>	

4.1.11 IsAuthAlgo

Syntax	<code>bool IsAuthAlgo(<parAuthMethod>)</code> Returns true if the user logged on at the time of calling was successfully identified by the authentication method specified in <code>parAuthMethod</code> .	
Parameters	<code><parAuthMethod></code>	Possible specifications: FormulaAuthentication, DigestAuthentication, BasicAuthentication, CertificateAuthentication
Example	<pre><MWSL> write(IsAuthAlgo("FormulaAuthentication")); </MWSL></pre>	
Output	1	
Valid as of	V4.4	

4.1.12 isFinite

Syntax	<code>bool isFinite(<value>)</code> Returns false if the passed value is NaN or infinite.	
Parameters	<code><value></code>	Value for the check.
Example	<pre><MWSL> write("Test of the number 123456 - isFinite: "); write(isFinite(123456) + "
"); write("Test of NaN - isFinite: "); write(isFinite(parseInt("ABC", 2))); </MWSL></pre>	
Output	Test of the number 123456 - isFinite: 1 Test of NaN - isFinite: 0	
Valid as of	V4.4	

4.1.13 isNaN

Syntax	<code>bool isNaN(<value>)</code> Checks whether the passed value is an invalid double.	
Parameters	<code><value></code>	Value for the check.
Example	<pre><MWSL> write("Test of 123456 - isNaN: "); write(isNaN(123456) + "
"); write("Test of NaN - isNaN: "); write(isNaN(parseInt("ABC", 2))); </MWSL></pre>	

4.1 MWSL functions

Output	Test of 123456 - isNaN: 0 Test of NaN - isNaN: 1
Valid as of	V4.4

4.1.14 IsSSL

Syntax	bool IsSSL() Returns true if the client is connected to the server via an SSL connection.	
Parameters		
Example	<pre><MWSL> write("If the client is connected to the server via an SSL connection:"); write(IsSSL()); </MWSL></pre>	
Output	If the client is connected to the server via an SSL connection: 1	
Valid as of	V4.4	

4.1.15 parseFloat

Syntax	double parseFloat(<string>) Conversion of a string to a double value.	
Parameters	<string>	String to be converted
Example		
Output		
Valid as of	V4.4	

4.1.16 parseInt

Syntax	int parseInt(<value>, [<base>]) Conversion of a string to an integer value.	
Parameters	<value>	String to be converted. If a value starts with 0x, it will be interpreted as hexadecimal. Values starting 0 will be interpreted as octal. All other values are shown in decimal format. Maximum value: 0x7FFFFFFF. If values exceed the upper limit, the maximum value 2147483647 is returned. If the value shall be shown as a negative number, put "-" in front.
	<base>	Basis to which the string shall be converted. Values: "2" = binary, "8" = octal, "16" = hexadecimal. No value = decimal interpretation.

Example	<pre> <MWSL> var tmpVar0 = "0x1"; var tmpVar1 = "0x2"; var tmpSum = tmpVar0 + tmpVar1; write(tmpSum + "
"); var tmpVarInt0 = parseInt(tmpVar0); var tmpVarInt1 = parseInt(tmpVar1); tmpSum = tmpVarInt0 + tmpVarInt1; write(tmpSum + "
"); tmpVar0 = "101"; tmpVar1 = "100"; tmpSum = tmpVar0 + tmpVar1; write(tmpSum + "
"); tmpVarInt0 = parseInt(tmpVar0,"2"); tmpVarInt1 = parseInt(tmpVar1,"2"); tmpSum = tmpVarInt0 + tmpVarInt1; write(tmpSum + "
"); tmpVar0 = "A"; tmpVar1 = "B"; tmpSum = tmpVar0 + tmpVar1; write(tmpSum + "
"); tmpVarInt0 = parseInt(tmpVar0,"16"); tmpVarInt1 = parseInt(tmpVar1,"16"); tmpSum = tmpVarInt0 + tmpVarInt1; write(tmpSum + "
"); </MWSL> </pre>
Output	<pre> 0x10x2 3 201 9 AB 21 </pre>
Valid as of	V4.4

4.1.17 ProcessXMLData

<p>Syntax</p>	<p>ProcessXMLData(<DATA>, <TEMPLATE>)</p> <p>With this command, dynamic HTML files can be generated based on a data and template file. The parameter <DATA> contains the data that are interpreted with the template in parameter <TEMPLATE>.</p> <p>ProcessXMLData combines the two files into one HTML file. The data nodes of the data file are evaluated by the template file to be displayed.</p> <p>This produces a separation of the data from the content. With a subsequent change to the template file, the appearance of the pages can be altered without changing the data.</p> <p>This makes it easier to added to data. Using different templates, it is possible to generate pages with the same data but completely different appearances.</p> <p>Additional information about the template mechanism: Mode of operation of the template mechanism (Page 66)</p>	
<p>Parameters</p>	<p><DATA></p>	<p>Data for the dynamic HTML file</p> <p>A file or a variable containing the data can be passed as a parameter.</p> <p>File: "<EXTERNAL SRC=\""/datafile.xml \"/>", in which datafile.xml is the file containing the data.</p> <p>Variable: <variable name> Specifies the variable name.</p>
	<p><TEMPLATE></p>	<p>Template (how the data are displayed)</p> <p>A file or a variable containing the templates can be passed as a parameter.</p> <p>File: "<TEMPLATES><EXTERNAL SRC=\""/Template.xml\"/> </TEMPLATES>", in which "Template.xml" is the file containing the templates.</p> <p>Variable: <variable name> Specifies the variable name.</p>

Example	<pre>ProcessXMLData("<EXTERNAL SRC=\" /MWSL/variables.xml \"/>", "<TEMPLATES><EXTERNAL SRC=\" /MWSL/variablesTemplate.xml \"/></TEMPLATES>");</pre>
MWSL example	<pre><MWSL> var Head = "<Provider Name =\"MyVarProvider\">"; var Data = "<Variable Name=\"ZUFUEHRUNG\" Type=\"String\" InitialValue=\"good\" Behavior=\"Manual\" Description=\"Part infeed.\"/>"; var Foot = "</Provider>"; var XMLData = Head + Data + Foot; var TemplateHead = "<TEMPLATES>"; var TemplateFoot = "</TEMPLATES>"; var TemplateFile = "<EXTERNAL SRC=\"\" + GetVar(\"Template\", \"URL\") + \"\"/>"; ProcessXMLData(XMLData, TemplateHead + TemplateFile + TemplateFoot); </MWSL></pre> <p>The XMLData variable contain the data nodes with the associated attributes.</p> <p>The example shows how variables can be defined for a template. The variable TemplateFile for the actual template consists of a reference to a file in this case.</p> <p>Note that the quotation marks in the template are protected from the XML parser by a preceding \.</p>

4.1.18 ReadFile

Syntax	<pre>string ReadFile(<file name>)</pre> <p>This function is similar to the function InsertFile, except that the content of the file is not written, but only returned as a return value.</p>	
Parameters	<file name>	Name of text file, including path.
Example	<pre><MWSL> var tmpFile = ReadFile("/files/include.mwsl"); write(tmpFile); </MWSL></pre>	
Output	The content of file include.mwsl is written to variable tmpFile and then written into the output.	
Valid as of	V4.4	

4.1.19 ReplaceString

Syntax	ReplaceString(<variable name>,<search pattern>,<replacement string>) Replacing strings.	
Parameters	<variable name>	Variable in which the characters shall be replaced.
	<search pattern>	Search pattern for replacing the characters.
	<replacement string>	string that is inserted.
Example	<pre><MWSL> var tmpString = "Ein String"; var tmpOutString; tmpOutString = ReplaceString(tmpString,"n","N"); write("Result: " + tmpOutString); </MWSL></pre>	
Output	Result: EiN StriNg	
Valid as of	V4.4	

4.1.20 SetVar

Syntax	SetVar(<Variable Name>, <Value>) This function is used to place process variables.	
Parameters	<variable name>	Variable name
	<value>	The new variable value.
Example	<pre>SetVar("var/userData.user1", "NewUserData");</pre> Sets the process variable <code>var/userData.user1</code> to the value <code>NewUserData</code> .	

4.1.21 ShareRealm

Syntax	ShareRealm(<Group>) Indicates whether the current user is a member of the group that is passed as a parameter. The return value can be <code>true</code> or <code>false</code> .	
Parameters	<Group>	The following groups are currently allowed as parameters: <ul style="list-style-type: none"> • NO_REALM No group association • ANY_REALM Any group association • [group name] Member of group [group name] Groups depend on configuration.

Example	<pre>write(ShareRealm ("ANY_REALM"));</pre> <p>If the current user is in any defined group, 1 is output. Otherwise, 0 is output.</p>
MWSL example	<pre><MWSL> if (ShareRealm("ANY_REALM")) { write ("<tr valign=\"baseline\">\r\n"); write ("<td><H2>Hello " + GetVar("Username", "HTTP") + ", you're successfully logged in.</H2></td>\r\n"); write ("</tr>\r\n"); } </MWSL></pre> <p>If the user is a member of any group, the instructions in the curly brackets are carried out.</p>

4.1.22 write

Syntax	write(<Text>)	
	The write function writes text to the output of an HTML page.	
Parameters	<Text>	Text, return values of functions, or variable contents can be passed.
Example	<pre> <MWSL> write("Hello World"); // Output: Hello World write("Hello" + " " + "World"); // Output: Hello World write(GetVar("Parameter", "URL")); // Output of the content of variable Parameter in the URL. write(5+6); // Output: 11 var zahl1 = 5; var zahl2 = 7; var string = "Hello"; write(string + ": " + zahl1 + zahl2); // Output: Hello: 57 write(zahl1 + zahl2); // Output: 12 write("Content of Parameter: " + GetVar("Parameter", "URL")); // Ausgabe: Content of Parameter: Hello // if Parameter contains the string "Hello". </MWSL> </pre>	

4.1.23 WriteToTab

Syntax	WriteToTab(<parTabPos>,<parFillChar>)	
	Write from <parFill> to position <parTabPos>.	
Parameters	<parTabPos>	Position up to which writing is to be performed (modulo).
	<parFillChar>	The 1st character that is written.

Example	<pre><MWSL> WriteToTab(10, "1"); write("xxx"); WriteToTab(10, "2"); </MWSL></pre>
Output	111111111xxx222222
Valid as of	V4.4

4.1.24 WriteVar

Syntax	<pre>WriteVar(<Variable Name>, <Variable Source>, <Format String>);</pre> <p>This command writes the content of a variable to the output.</p> <p>The functionality of the function <code>WriteVar</code> is similar to that of the function <code>GetVar</code>. <code>WriteVar</code> is equivalent to the call: <code>WriteVar(...) === write(GetVar (...))</code></p> <p>The sole difference is that <code>WriteVar</code> outputs the content of the specified variable, while <code>GetVar()</code> returns the content as a return value.</p>	
Parameters	<variable name>	Variable name
	<variable source>	<p>Name of variable source</p> <p>Valid values:</p> <ul style="list-style-type: none"> • URL • HTTP • PROCESS Read variables from the providers. • COOKIE Read variables from the HTTP header of the cookie • XML • SENDPAGE • DEFAULT The default setting is <code>PROCESS</code>. <p>If not source is stated, <code>DEFAULT</code> is selected, that is, the variable provider.</p> <p>The name of the variable providers, such as <code>SIMOTION</code>, <code>MINIWEB</code>, etc., are not designations for variable sources.</p> <p>The suitable provider is searched for in the web server based on the variable name.</p>
	<Format String>	<p>The handling of the format string depends on the variable source.</p> <p>Thus, this property is not possible for the variable sources <code>COOKIE</code> and <code>URL</code>.</p> <p>The call syntax is equivalent to that of the <code>GetVar()</code> (Page 112) function.</p>

<p>Example</p>	<pre> <MWSL> write(GetVar("Parameter", "URL")); // The content of the variable Parameter is output here. //GetVar returns the value of the variable, // which is then written to the output with write. // (See also GetVar() and write().) // The same output can also be achieved with the following // command. WriteVar("Parameter", "URL"); //WriteVar writes directly to the output and supplies // no return value WriteVar("var/userData.user1"); // Outputs the content of var/userData.user1, which is a // process variable. WriteVar("Accept-Language", "HTTP", "?-") // Outputs the content of the HTTP variable "Accept-Language" // up to the "-" character. WriteVar("var/userData.user1", "PROCESS", "[2,3]"); // Outputs characters 2 - 5 of the process variable var/ userData.user1. WriteVar("Accept-Language", "HTTP", "[3,0]") // Outputs the content of the HTTP variable "Accept-Language" // starting from the third character. </MWSL> </pre>
<p>Example</p>	<pre> <MWSL> // Identical calls WriteVar("var/modeOfOperation"); WriteVar("var/modeOfOperation", "PROCESS"); WriteVar("var/modeOfOperation", "DEFAULT"); </MWSL> </pre>

See also

Global variables (Page 52)

4.1.25 WriteXMLData

Syntax	<p>WriteXMLData (<DATA>, <TEMPLATE>)</p> <p>WriteXMLData outputs the data in contrast to ProcessXMLData. Instead of <code>write(ProcessXMLData(...));</code> , you can also write <code>WriteXMLData(...) ; .</code> <code>WriteXMLData () ;</code> is assigned the same parameters as <code>ProcessXMLData () ; .</code></p>	
Parameters	<DATA>	<p>Data for the dynamic HTML file</p> <p>A file or a variable containing the data can be passed as a parameter.</p> <p>File: "<code><EXTERNAL SRC=\" /Datendatei.xml \"/></code>", in which "data file.xml" is the file containing the data.</p> <p>Variable: <code><variable name></code> Specifies the variable name.</p>
	<TEMPLATE>	<p>Template (how the data are displayed)</p> <p>A file or a variable containing the templates can be passed as a parameter.</p> <p>File: "<code><TEMPLATES><EXTERNAL SRC=\" /Template.xml \"/></TEMPLATES></code>", , in which "Template.xml" is the file containing the templates.</p> <p>Variable: <code><variable name></code></p>
Example	<pre>WriteXMLData("<EXTERNAL SRC=\" /MWSL/variables.xml \"/>", "<TEMPLATES><EXTERNAL SRC=\" /MWSL/variablesTemplate.xml \"/></TEMPLATES>");</pre> <p>Additional information about the template mechanism: Mode of operation of the template mechanism (Page 66)</p>	

4.1.26 NodeIndex

Variable	NodeIndex	<p>NodeIndex is a process variable that is available when a template is parsed.</p> <p>This variable outputs the number of nodes that have already been run through.</p> <p>The access is exactly the same as for other variables of the PROCESS variable source.</p> <p>Additional information about the template mechanism: Mode of operation of the template mechanism (Page 66)</p>
Example	<pre><?xml version="1.0" ?> <TEMPLATES> <TEMPLATE NAME="Variable"> <POSITION NAME="LINE"> <![CDATA[<MWSL> WriteVar("NodeIndex ") </MWSL>]]> </POSITION> </TEMPLATE> </TEMPLATES></pre>	

4.1.27 NodeLevel

Variable	NodeLevel	<p>NodeLevel is a process variable that is available when a template is parsed.</p> <p>This variable outputs the hierarchy level of the current node.</p> <p>The access is exactly the same as for other variables of the PROCESS variable source.</p> <p>Additional information about the template mechanism: Mode of operation of the template mechanism (Page 66)</p>
Example	<pre><?xml version="1.0" ?> <TEMPLATES> <TEMPLATE NAME="Variable"> <POSITION NAME="LINE"> <![CDATA[<MWSL> WriteVar("NodeLevel") </MWSL>]]> </POSITION> </TEMPLATE> </TEMPLATES></pre>	

Index

B

Browse, 91

C

CancelTrace, 102
Client application, 90
Communication package, 84

D

DEFAULTDOCUMENT, 13
delete, 59

G

GetProperties, 92
GetStatus, 92, 103

H

Home page, 13

J

JavaScript
 ApplBrowser, 42
 ApplBrowseTree, 45
 ApplDataTable, 38
 Device access, 24
 Library appl.js, 37
 OPCBrowseRequest, 30
 OPCGetPropertiesRequest, 26
 OPCReadRequest, 24
 OPCSubscriptionAutoRefresh, 35
 OPCSubscriptionRequest, 32
 OPCWriteRequest, 29

M

MWSL, 59
 .cms extension, 16
 Access to the process variables, 47
 break, 64

Comments, 64
 continue, 64
 COOKIE variables, 57
 COOKIES, 56
 do, 63
 Error message, 18
 Error messages, 48
 For, 63
 Format string, 58
 Format string GetVar, WriteVar, 52
 function, 64
 Function overview, 65
 Global variables, 52
 HTTP header, 57
 If, 61
 Load pages into the controller, 15
 MBS and MCS files, 18
 Mode of operation, 47
 new, 59
 Operators, 58
 Processing variable values, 55
 return, 64
 Script variables, 50
 Structure, 48
 switch, 62
 Template mechanism, 66
 Translation, 15
 URL parameters, 55
 UTF-8, 15
 while, 63

MWSL examples

 SetVar(), 74
 TestMenu, 80
 TestTemplate, 74

MWSL functions

 AddHTTPHeader, 109
 createGUID, 109
 DecodeString, 110
 EncodeString, 111
 ExistFile, 111
 ExistVariable, 112
 GetLanguage, 112
 GetVar, 112
 InsertFile, 114
 IsAuthAlgo, 115
 isFinite, 115
 isNaN, 115
 IsSSL, 116
 parseFloat, 116

- parseInt, 116
- ProcessXMLData, 118
- ReadFile, 119
- ReplaceString, 120
- SetVar, 120
- ShareRealm, 120
- the, 110
- write, 122
- WriteToTab, 122
- WriteVar, 123
- WriteXMLData, 125

- MWSL, 47
- OPC XML server, 24

W

- Write, 93

O

- OPC
 - XML DA, 84
- OPC XML DA
 - Access protection, 89
- OPC XML server interface, 91
- OPC XML-DA R1.0 Specification, 84

R

- Read, 92
- References, 3
- Ring buffer, 97

S

- Server Side Includes, 82
- SOAP, 84
- StartTrace, 101
- StopTrace, 102
- Subscribe, 92
- SubscriptionCancel, 92
- SubscriptionPolledRefresh, 92

T

- Template mechanism
 - NodeIndex, 126
 - NodeLevel, 127
- TraceDataCycleEnum, 99
- TraceStateEnum, 99

U

- User-defined pages, 14
 - Home page, 13
 - JavaScript, 24