

SIEMENS

Ingenuity for life

Industry Online Support

Home

SINAMICS S120 web server - Creating user-defined web pages

SINAMICS S120 / V2.1

<https://support.industry.siemens.com/cs/ww/en/view/68691599>

Siemens
Industry
Online
Support



Legal information

Use of application examples

Application examples illustrate the solution of automation tasks through an interaction of several components in the form of text, graphics and/or software modules. The application examples are a free service by Siemens AG and/or a subsidiary of Siemens AG ("Siemens"). They are non-binding and make no claim to completeness or functionality regarding configuration and equipment. The application examples merely offer help with typical tasks; they do not constitute customer-specific solutions. You yourself are responsible for the proper and safe operation of the products in accordance with applicable regulations and must also check the function of the respective application example and customize it for your system.

Siemens grants you the non-exclusive, non-sublicensable and non-transferable right to have the application examples used by technically trained personnel. Any change to the application examples is your responsibility. Sharing the application examples with third parties or copying the application examples or excerpts thereof is permitted only in combination with your own products. The application examples are not required to undergo the customary tests and quality inspections of a chargeable product; they may have functional and performance defects as well as errors. It is your responsibility to use them in such a manner that any malfunctions that may occur do not result in property damage or injury to persons.

Disclaimer of liability

Siemens shall not assume any liability, for any legal reason whatsoever, including, without limitation, liability for the usability, availability, completeness and freedom from defects of the application examples as well as for related information, configuration and performance data and any damage caused thereby. This shall not apply in cases of mandatory liability, for example under the German Product Liability Act, or in cases of intent, gross negligence, or culpable loss of life, bodily injury or damage to health, non-compliance with a guarantee, fraudulent non-disclosure of a defect, or culpable breach of material contractual obligations. Claims for damages arising from a breach of material contractual obligations shall however be limited to the foreseeable damage typical of the type of agreement, unless liability arises from intent or gross negligence or is based on loss of life, bodily injury or damage to health. The foregoing provisions do not imply any change in the burden of proof to your detriment. You shall indemnify Siemens against existing or future claims of third parties in this connection except where Siemens is mandatorily liable.

By using the application examples you acknowledge that Siemens cannot be held liable for any damage beyond the liability provisions described.

Other information

Siemens reserves the right to make changes to the application examples at any time without notice. In case of discrepancies between the suggestions in the application examples and other Siemens publications such as catalogs, the content of the other documentation shall have precedence.

The Siemens terms of use (<https://support.industry.siemens.com>) shall also apply.

Security information

Siemens provides products and solutions with Industrial Security functions that support the secure operation of plants, systems, machines and networks.

In order to protect plants, systems, machines and networks against cyber threats, it is necessary to implement – and continuously maintain – a holistic, state-of-the-art industrial security concept. Siemens' products and solutions constitute one element of such a concept.

Customers are responsible for preventing unauthorized access to their plants, systems, machines and networks. Such systems, machines and components should only be connected to an enterprise network or the Internet if and to the extent such a connection is necessary and only when appropriate security measures (e.g. firewalls and/or network segmentation) are in place.

For additional information on industrial security measures that may be implemented, please visit <https://www.siemens.com/industrialsecurity>.

Siemens' products and solutions undergo continuous development to make them more secure. Siemens strongly recommends that product updates are applied as soon as they are available and that the latest product versions are used. Use of product versions that are no longer supported, and failure to apply the latest updates may increase customer's exposure to cyber threats.

To stay informed about product updates, subscribe to the Siemens Industrial Security RSS Feed at: <https://www.siemens.com/industrialsecurity>.

Table of contents

	Legal information	2
1	Introduction	5
1.1	Overview	5
1.2	Mode of operation	6
1.3	Components used	8
2	Basic information	9
2.1	Hypertext Markup Language (HTML)	9
2.1.1	Structure of an HTML file	9
2.1.2	Basic elements	10
2.1.3	IFrames	13
2.1.4	Canvas elements	14
2.2	MiniWeb Server Language (MWSL)	15
2.2.1	Principle of operation	15
2.2.2	Structure of an MWSL file	15
2.2.3	Variable types	16
2.2.4	Script variables	16
2.2.5	Global variables	18
2.2.6	User access	20
2.2.7	Operators	20
2.2.8	Overview of MWSL functions	21
2.2.9	Overview of SINAMICS process variables	22
2.2.10	MWSL functions	22
2.3	JavaScript	33
2.3.1	Integrating JavaScript into HTML	33
2.3.2	The Document Object Model (DOM)	34
2.3.3	getElementById	35
2.3.4	Event handler	36
2.3.5	Functions	36
2.3.6	Libraries	36
2.3.7	Variable types	37
2.4	Cascading Style Sheets (CSS)	39
2.4.1	Integrating CSS in HTML	39
2.4.2	Defining formats for classes	39
2.4.3	Defining individual formats	40
3	Engineering	42
3.1	Project planning and configuration	42
3.1.1	Concept	42
3.1.2	Files used	44
3.1.3	Structure and content	45
3.2	Creating a parameter source	48
3.3	Creating the content of the web page	54
3.3.1	Displaying parameter values	54
3.3.2	Updating parameter values	58
3.3.3	Using the callback function	60
3.3.4	Initializing and updating canvas elements	61
3.4	Loading the content of the web page	64
3.5	Sample pages	65
3.5.1	diagAxis.html (main page 1)	65
3.5.2	diagStatusAxis.html (main page 2)	67
3.5.3	axisRedLoader.html (load page)	69
3.6	Uploading files to the web server	71
3.7	Uploading new pages	75
3.8	Loading and commissioning the sample pages	76

Table of contents

3.8.1	...via a CF card reader	76
3.8.2	...via the webserver.....	78
3.9	Operation.....	82
3.10	Changing the name of the axes	85
4	Appendix	86
4.1	Service and Support.....	86
4.2	Application support.....	87
4.3	Links and Literature	87
4.4	Change documentation	87

1 Introduction

1.1 Overview

Introduction

The web server provides information on a SINAMICS device via its web pages. The access is realized using a standard web browser (e.g. Internet Explorer or Mozilla Firefox). The basis configuration of the web server can be realized via STARTER or the web server itself, e.g. by loading a configured project.

Users have the option to complement the standard web pages by their own web pages. The own created web pages ("user-defined web pages") can be subsequently uploaded to the SINAMICS S120 web server. Drive parameters can be read – and therefore the widest range of scenarios visualized – using these web pages, for example.

Description of the automation task

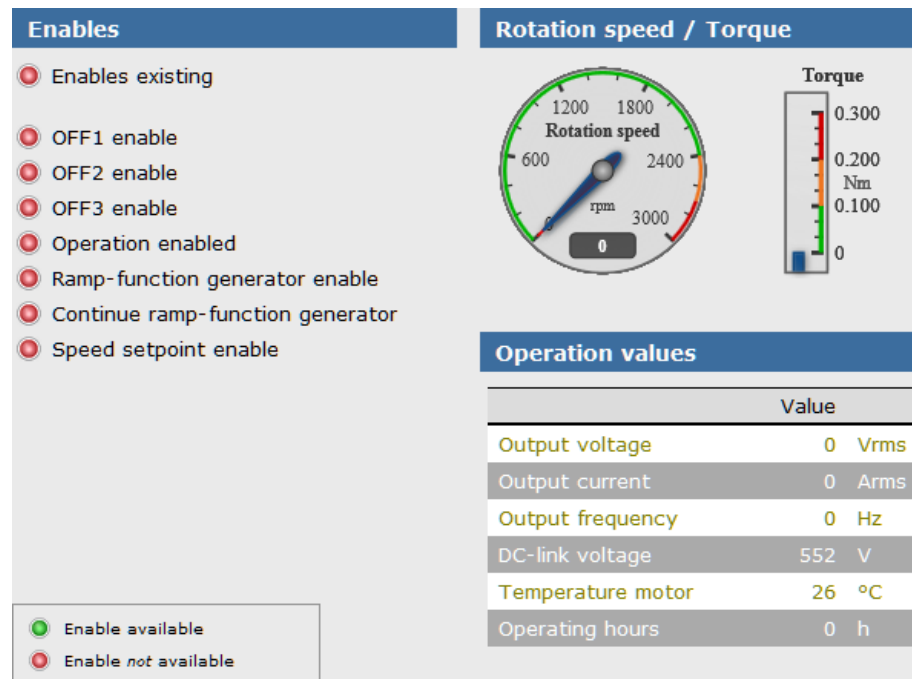
The following states of a drive are to be visualized using a user-defined web page:

- Enable signals (e.g. OFF1, ramp-function generator enable, etc.) using color-coded status displays
- Speed and torque using graphics (e.g. tachometer display)
- Drive parameters (e.g. output voltage, motor temperature, etc.) in the form of a table
- Control- and status words using color-coded status displays

The displayed data is updated at specific intervals; this means that changes to the drive state can be visualized.

The web page can be designed to address the user's own specific requirements and ideas.

Figure 1-1

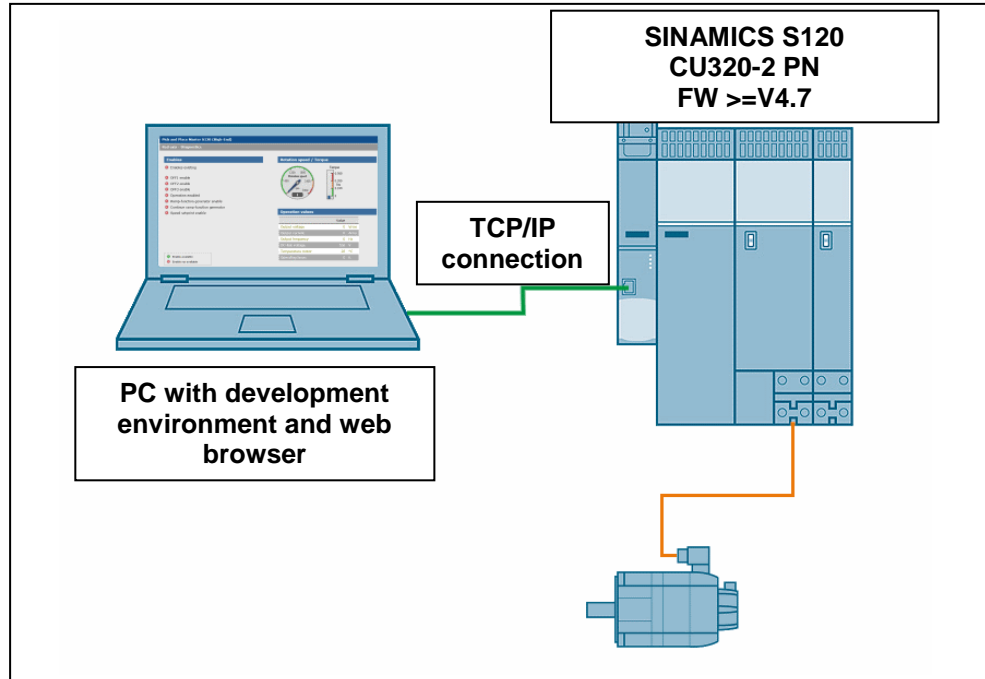


1.2 Mode of operation

Display

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

Figure 1-2



Based on a sample web page, the procedure to create user-defined web pages for the SINAMICS S120 web server is explained in more detail in this application description.

The data to be displayed on the web page are read from the drive using the script language **MiniWeb Server Language (MWSL)** (see Chapter 2.2: [MiniWeb Server Language \(MWSL\)](#)).

Using **JavaScript**, the properties of the web page and its content can be accessed; for example, this allows data to be updated at specific intervals, i.e. read out of the drive again.

Cascading Style Sheets (CSS) are used for the design. The web page formatting is centrally saved in these sheets.

Advantages

The application described here offers you the following advantages:

- Drive parameters and data can be accessed via MWSL and a standard web browser – an engineering system is not required!
- The displayed data can be updated and accessed using JavaScript
- The web page is designed using HTML and CSS – so that users can freely implement their ideas and requirements

Delimitation

This application does not contain a description of

- the functions of the SINAMICS S120 web server

Basic knowledge of this topic is assumed.

Note

You can find basic information on SINAMICS S120 in the function manual of the drive functions at the following link:

<https://support.industry.siemens.com/cs/ww/en/view/109740020>

Required knowledge

Basic know-how about creating web pages using HTML, JavaScript and CSS is assumed.

Supplementary conditions

1. Access to drive parameters

Currently only the **MWSL** script language of SINAMICS S120 is supported to access drive parameters.

2. Total memory size of user data

The total amount of data stored on the drive via the web server must not exceed **100 MB**. The total memory size of the saved data influences the backup times, and after a change on the CF card, also influences the time when rebooting, but only once. The more data there is, the longer it takes to back up.

1.3 Components used

This application example has been created with the following hardware and software components:

Table 1-1

Component	Number	Article number	Note
SINAMICS training case incl. CU320-2 PN	1	6ZB2480-0CM00 / 6ZB2480-0CN00	FW V4.8 HF 4
STARTER	1	6SL3072-0AA00-0AG0	V4.5
Mozilla Firefox	1	---	Download
Internet Explorer (as alternative to Mozilla Firefox)		---	Download
Microsoft Visual Studio Express 2012 for the web		---	Download (test version)

This application example consists of the following components:

Table 1-2

Component	Note
68691599_S120_Userdefined_Webpages_V2_0.zip	ZIP archive with finished example pages
68691599_S120_Userdefined_Webpages_V2_0_en.pdf	This document

2 Basic information

2.1 Hypertext Markup Language (HTML)

The Hypertext Markup Language (HTML) is a text-based markup language to structure content, such as text, graphics and hyperlinks in documents. HTML documents represent the basis of the World Wide Web, and displayed using a web browser. In addition to the content of a web page, displayed by the browser, HTML also includes additional data in the form of meta information. This summarizes the language used in the text, information about the author or the content of the text.

2.1.1 Structure of an HTML file

Every HTML document comprises three parts:

- The document type declaration
- The HTML head (<head>)
- The HTML body (<body>)

The document type declaration includes data on the markup language used and its version.

The head data of the page are noted in the HTML head. In addition to optional meta data (for example, information about the author), the page title is specified, as it will be subsequently displayed in the title line of the web browser.

The content of the web page to be displayed is in the HTML body itself, i.e. text passages with headers, references, tables and forms.

The basic framework of an HTML file generally looks like this:

```
<html>
  <head>
    <title>
      <!--title of the website-->
    </title>
  </head>
  <body>
    <!--content of the website-->
  </body>
</html>
```

All HTML elements are marked using tags. These are located in angle brackets (< and >) and comprise an opening and closing part; the closing part only differs from the opening part by a preceding slash. <html> and </html>).

Two tags <html> and </html> mark the HTML code and limit it. This area is again subdivided into a head and body area. The page title is noted between the <title> and </title> tags. The HTML body contains the actual page content.

2.1.2 Basic elements

Several basic elements are used to structure an HTML page, which will be described in more detail in the following section.

Areas (div / span)

A general block element is opened with `<div>`, which can encompass several other block elements.

Everything that is located between this tag and the closing `</div>` tag is interpreted as part of this area.

Several elements, such as text, graphics, tables etc. can be included in a common area. This general element always starts with a new line of the continuous text. It is intended that this is formatted using CSS (Cascading Style Sheets).

Analogous to the `<div>` element, there is also what is known as a `` element.

`` opens a general, inline area (within one line), this area is closed with ``. It is also intended that this is formatted using CSS.

This means that by using `` elements, it is possible to color individual lines or words of a text, for example.

Tables (table)

The table is one of the most important elements for HTML.

Using a table, a complete page can be subdivided, or also just a couple of points can be displayed in a structured fashion.

The `<table>` and `</table>` tags are used to create a table in HTML. A table comprises one or several table rows (`<tr>` or `</tr>`). In turn, a line comprises one or several cells (`<td>` or `</td>`, table data). The individual cells of a table include the content to be displayed and can themselves comprise tables.

Example:

```
<table>
  <tr>
    <td>
      <!--first row, first data cell-->
    </td>
    <td>
      <!--first row, second data cell-->
    </td>
  </tr>
</table>
```

This HTML code creates a single-row table with two columns.

The content of each cell describes its position in the table (e.g. "first row, first column"). The table can be continued as required, both regarding the number of rows as well as also the number of columns per row.

Text fields and buttons (input)

The `Input` fields are another important element.

These can be used to create input fields or buttons. The input fields are mostly used to enter text, which, for example by pressing an adjacent button, can be further processed using JavaScript.

In principle, the structure of both of these elements is identical; the `type` attribute represents the decisive difference. Here, to mark a text field, the `type` attribute is assigned the value `text`, for buttons, the value `button`.

In addition, buttons also require an `onclick` event handler (see Chapter 2.3.4: [Event handler](#)), which defines what should happen when clicking the button.

Example:

```
<input type="text" name="text" value="content" size="15"/>
<input type="button" id="button" value="Send" onclick="get()"/>
```

This HTML code creates a single-row input field, which is pre-assigned the word "content". This is followed by a button with the label "Send".

While the `value` attribute for text fields results in a pre-assignment, for a button, `value` is written to that button.

If this attribute is omitted, then the text field remains empty and the button is not labeled. The `size` attribute defines the display width of the field.

Using the name specified in the `name` attribute, elements can be referenced, for example to be able to access them using JavaScript.

This internal reference name is only used to access HTML elements using scripts, however without having any effect on the layout of a web page. Alternatively (or additionally), an ID can be assigned to an element (attribute `id`), which can be used to reference the element (see Chapter 2.3.3: [getElementById](#)).

In the `onclick` event handler of the button, it is specified that when clicking the button, a JavaScript function named `get()` should be called.

Contrary to other HTML elements, `Input` elements have no closing `</input>` tag – it is simply closed using `/>`.

Selection lists (select)

Users can be provided with lists with fixed selection options in the form of selection lists. One or several entries can then be selected from this list.

The selection lists are marked using the `<select>` and `</select>` HTML tags. If event handlers are noted within these `select` tags, a response can be made to the selection using JavaScript.

The actual list entries are written using `<option>` tags in the selection list. The `size` attribute can be used to specify how many of the entries should be displayed at once.

If the list has more entries than marked for display in the `size` attribute, then a scrollbar is automatically inserted, which can be used to navigate through all of the entries.

If only one entry is to be displayed (`size="1"`), a drop-down menu is created from the selection list.

Example:

```
<select id="selectList" size="1" onchange="switch()">
  <option value="0">Off</option>
  <option value="1">On</option>
</select>
```

This HTML code creates a selection list (drop-down menu) with the "Off" and "On" entries.

In the `onchange` event handler of the selection list, it is specified that when changing the selection, a JavaScript function named `switch()` should be called.

In order to be able to access the elements of the selection list using JavaScript, the selection list must be assigned an ID (attribute `id`) and the individual entries, an internal value (attribute `value`) which can be used to reference the entry selected using `getElementById` in JavaScript.

Lists (ul / ol)

In HTML, the `` tag opens an enumeration list (ul = unsorted list), the `` tag closes this list.

For an enumeration list, all of the list entries have an enumeration character (bullet). A list entry such as this is opened with the `` tag and is closed with ``.

Example

```
<ul>
  <li>...</li>
  <li>...</li>
</ul>
```

This HTML code generates an unsorted list, which has two list entries. The list entries can be continued as required.

Contrary to the `` tag, using the `` tag, a numbered list can be created (ol = numbered list). The list is closed using the `` tag.

For a numbered list, all of the list entries are automatically numbered. A list entry such as this is opened with the `` tag, and is closed with ``.

Note

You can find further information on the topic of "lists" under the following link:

http://www.w3schools.com/html/html_lists.asp

2.1.3 IFrames

IFrames (inline frame = embedded frame) is a layout resource that allows developers to embed third-party sources – especially other HTML pages – in their own web page.

Different than normal frames, in this case, the web page is not subdivided. Instead, an area is defined in the page, which is reserved to display a source (such as graphics or other web pages).

IFrames are marked using the `<iframe>` and `</iframe>` HTML tags. An explanatory text can be specified between these IFrame tags, which is displayed if the browser does not support IFrames (i.e. if the embedded source cannot be displayed).

The source itself – or more precisely its URL – is assigned the `src` attribute in the opening `<iframe>` tag.

The dimensions of the embedded frame are not aligned according to the size of the source, but are defined, e.g. in the attributes `width` and `height` of the associated stylesheet (CSS).

If the dimensions of the referenced source are larger than the area provided to the IFrame, then scroll bars are automatically inserted.

Using the `name` or `id`, attribute, which is also noted in the opening tag, it is possible to access the object properties of the IFrame using JavaScript, for instance to modify the dimensions or the source of the IFrame.

Example

```
<iframe src="picture1.jpg" id="iframe">  
  Your browser does not support IFrames.  
</iframe>
```

In this example, if the web browser supports IFrames, a graphic is displayed, otherwise "Your browser does not support IFrames" is displayed.

In order to be able to change additional attributes of the IFrame using CSS, the `id` attribute is noted in the opening `<iframe>` tag.

2.1.4 Canvas elements

In the HTML language, a canvas element is an area with a defined height and width which can be drawn into using JavaScript.

A canvas element is marked using the `<canvas>` and `</canvas>` HTML tags. An explanatory text can be specified between these canvas tags, which informs the user if the browser does not support canvas functionality.

In addition to being able to draw lines and squares, canvas also allows the following to be drawn:

- Circular arcs
- Bézier curves (quadratic and cubic)
- Color graduations
- Graphics (formats: PNG, GIF, JPEG), which can be scaled, positioned and cut
- Transparency (with several graduations)
- Text

Objects and object groups can be shifted, rotated and scaled. Animation is possible using JavaScript time functions.

Example

```
<canvas id="canvas" width="160" height="160">  
  Canvas is not supported by your browser.  
</canvas>
```

In the example, a canvas element is created with a width and height of 160 pixels. If the web browser does not support canvas elements, the text is displayed, located between the canvas tags.

In order to be able to access the object properties of the canvas element using JavaScript, the `id` attribute is noted in the opening `<canvas>` tag. This can be used, for example, to enter the content of the canvas element.

Note

Canvas elements are supported by the following web browsers:

- Internet Explorer from **Version 9.0**
- Mozilla Firefox from **Version 3.6**
- Opera from **Version 11.0**
- Google Chrome from **Version 14.0**

2.2 MiniWeb Server Language (MWSL)

The MiniWeb Server Language (MWSL) is a script language that is interpreted on the web server of the drive. It is rather similar to the JavaScript language but represents only a small part of the language scope.

2.2.1 Principle of operation

The MWSL enables a client (e.g. a PC) to be operated with a simple browser without scripting, as the web server generates the pages to be displayed dynamically.

MWSL enables variables to be accessed and processed. Among other things, it allows access to process variables (e.g. drive parameters) that are present on the basis web server system. These can be appropriately evaluated using MWSL.

When accessing variables, the client requests a URL on the web server. An MWSL file is on this; a temporary HTML file is generated from this on the web server using the MWSL service.

This is subsequently sent to the client where it is displayed.

2.2.2 Structure of an MWSL file

An MWSL file is a file with any format that also contains MWSL tags.

Example:

```
<html>
  <head>
    [...]
  </head>
  <body>
    <table>
      <tr>
        [...]
        <td>
          <MWSL>
            <!--
            //MWSL code
            -->
          </MWSL>
        </td>
        [...]
      </tr>
    </table>
  </body>
</html>
```

If the MWSL functionality is needed, the following tags are added:

- the `<MWSL>` tag opens the area that contains the MWSL code
- the `</MWSL>` closes this area

The HTML comment characters that follow the `<MWSL>` tag are not mandatory. However they are recommended because they protect the MWSL code from the HTML interpreter, thus preventing possible incorrect outputs.

2.2.3 Variable types

MWLS always makes a distinction between script variables and global variables:

- Script variables are defined within the area that contains the MWSL code
- Global variables are provided by variable sources

Note

Global variables are sources of information from the web service environment. Variables are exclusively accessed using access functions. Global variables are grouped in variable sources according to their origin.

2.2.4 Script variables

Script variables are variables that are only valid in the page in which they have been declared.

The variables are applicable beyond MWSL tags, i.e. they can be generated in one MWSL area, and first used in the next MWSL area.

A distinction is not made between data types, for instance, there is no explicit data type for "Integer" or "Char".

A variable is created as follows:

```
var <variable name> = <value>;
```

The data type is determined internally by the variable assignment.

Example 1:

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

In the example above, two variables are created, `string1` and `string2`. The two strings (with spaces) are connected with one another using the plus character. The result is output using the `write` command (output: Hello World).

Example 2:

```
<MWSL>
<!--
  var num1 = 5;
  var num2 = 7;
  var result;
  result = num1 + num2;
  write(result);
-->
</MWSL>
```

Two variables are created in the example above, `num1` and `num2`. The two numbers are added, and the result is saved in the `result` variable. The result is output using the `write` command (output: 12).

Keyword 'var'

A variable declaration is opened using the `var` keyword.

```
var  varName1 = InitialValue1,
     varName2 = InitialValue2,
     ... ;
```

Several variables are declared and (optionally) pre-assigned initial values. Several declarations can be specified, separated with a comma. The declaration is terminated using a semicolon.

Visibility and validity areas

For MWSL, the visibility and validity of variables must be observed.

Example 1:

```
<MWSL>
<!--
  var myVar = 10;
  {
    myVar = 20;
    write("Inner: " + myVar + ",");
  }
  write("Outer: " + myVar);
-->
</MWSL>
```

In this example, in the instruction block (in brackets), the `myVar` variable of the outer level (outside the brackets) is accessed; this is because no variable with the `myVar` name was declared at the instruction block level. As a result, the `myVar = 20;` instruction changes the value of the variables of the outer level (output: Inner: 20, outer: 20).

Example 2:

```
<MWSL>
<!--
  var myVar = 10;
  {
    var myVar = 20;
    write("Inner: " + myVar + ",");
  }
  write("Outer: " + myVar);
-->
</MWSL>
```

Contrary to example 1, within the instruction block, a `myVar` variable was also declared. The `write` command in the instruction block now accesses the variable of the inner level, the `write` command outside the instruction block, accesses the variable of the outer level.
(Output: Inner: 20, outer: 10)

Note

As a result of the higher risk of making mistakes, double declarations involving variables of the same name (for example, in example 2) should be avoided.

2.2.5 Global variables

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

- PROCESS variables enable access to normal variables of the web server (e.g. drive parameters). This is the standard access.
- URL variables provide access to variables contained in a URL.
- HTTP variables return the content of variables in the HTTP header.

PROCESS variables

PROCESS variables can be accessed using the following command:

```
GetVar("1.Params.2", "PROCESS");
```

The first number "1" defines the drive object number (Control unit).

".Params." is mandatory.

The second number "2" corresponds to the parameter number.

The variable source PROCESS must be written in uppercase letters.

For instance, if the `Color` variable does not exist, then a "zero" is returned.

PROCESS is the standard variable source, which means that PROCESS can also be omitted:

```
GetVar("1.Params.2");
```

Note

An overview of the functions provided in MWSL is listed in Chapter [2.2.8: Overview of MWSL functions](#)

URL variables

Using the functions `WriteVar`, `GetVar` and MWSL offers the possibility of processing URL parameter values.

Example of a URL with appended parameters

```
http://localhost/MWSL/StringOperationTest.mwsl?  
Parameter1=Hello&Parameter2=World!&StartValue=2&EndValue=5
```

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

For example, the URL variable `Parameter1` can now be output using the following command:

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

URL must be written in uppercase letters.

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

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

Note

Certain characters require a coding in order to be transferred correctly.

An overview of the most frequently used characters and their associated coding is provided in the "SIMOTION IT Programming and Web Services" programming manual (Chapter 3.1.8.9) at the following link:

<https://support.industry.siemens.com/cs/ww/en/view/109744613>

HTTP variables

A wide range of general information can be saved in the `<head>` tag of an HTML page.

The header can be read and written to using the MWSL functions `GetVar` and `WriteVar`.

Example

```
<html>  
  <head>  
    [...]   
    <META http-equiv="Accept-Language" content="de">  
    [...]   
  </head>  
  <body>  
    [...]   
  </body>  
</html>
```

In the above example, in the <META> tag, the HTTP variable `Accept-Language` is defined using the `http-equiv="Accept-Language"` command. Using the `content` attribute, it is initialized with the value `de`.

Note You can find further information on META data at the following link:

https://www.w3schools.com/tags/tag_meta.asp

These variables are accessed in a similar way as for URL variables. 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 uppercase letters.

2.2.6 User access

Both the user: "Administrator" and the user: "SINAMICS" have access to the user defined pages, thus it is possible to distinguish between the users via MWSL.

The logged in user is returned with the following command:

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

(SINAMICS or Administrator)

Alternatively the command [Table 2-25 ShareRealm](#) can be used.

2.2.7 Operators

Table 2-1 Comparison operators

Operator	Description
< or <=	This operator returns TRUE if the left variable is less (or less or equal) to the right variable.
> or >=	This operator returns TRUE if the left variable is greater (or greater or equal) to the right variable.
==	This operator returns TRUE if the left variable is equal to the right variable.

Table 2-2 Logical operators

Operator	Description
!	This operator returns TRUE if the subsequent parameter is FALSE (logical NOT).
&&	This operator returns TRUE if a TRUE value is present on the left side <u>and</u> right side (logical AND).
	This operator returns TRUE if a TRUE value is present on the left side <u>or</u> right side (logical OR).

Table 2-3 Mathematics operators

Operator	Description
+ or -	This operator adds the left and right variable or subtracts the value of the right variable from the value of the left variable.
* or /	This operator multiplies the left variable with the right variable or divides the left variable by the right variable.

Operator	Description
++ or --	This operator increments (+1) or decrements (-1) the prefixed variable.
%	This operator returns the remainder of a division (modulo).

2.2.8 Overview of MWSL functions

Table 2-4 Overview of all MWSL functions

Function	Description
AddHTTPHeader Table 2-6 AddHTTPHeader	Insert <Http-Header> in a page.
CreateGUID Table 2-7 CreateGUID	Generates a unique alphanumeric ID in the system.
DecodeString Table 2-8 DecodeString	Converts a string encoded with EncodeString back to its original.
die Table 2-9 die	Abort program execution.
EncodeString Table 2-10 EncodeString	Replaces special characters by their URL-coded hex value (%hh).
ExistFile Table 2-11 ExistFile	Checks whether a file with the name <file name> exists. The function returns the file length as returned value.
ExistVariable Table 2-12 ExistVariable	Query of the existence of a variable.
GetLanguage Table 2-13 GetLanguage	Returns the currently set language.
GetVar Table 2-14 GetVar	Return the value of a variable of the corresponding variable source.
InsertFile Table 2-15 InsertFile	Import of a <File>. A path can be specified.
isFinite Table 2-16 isFinite	Returns false if the passed value is NaN or infinite.
isNaN Table 2-17 isNaN	Checks whether the passed value is an invalid Double.
IsSSL Table 2-18 IsSSL	Returns true if the client is connected to the server via an SSL connection
parseFloat Table 2-19 parseFloat	Conversion of a string to a double value.
parseInt Table 2-20 parseInt	Conversion of a string to an integer value.
ProcessXMLData Table 2-21 ProcessXMLData	Generation of dynamic HTML files with special XML files.
ReadFile Table 2-22 ReadFile	Returns the content of the file as the return value.
ReplaceString Table 2-23 ReplaceString	Replacement of strings matching the search pattern.
SetVar Table 2-24 SetVar	Sets values of parameters.
ShareRealm Table 2-25 ShareRealm	Indicates whether the current user is a member of the group that is passed as a parameter. The return value can

Function	Description
	be true or false.
write Table 2-26 write	Writes <text> strings to the HTML page. <Text> can also be the return value of functions.
WriteVar Table 2-27 WriteVar	Output of a variable value. The syntax is identical to the GetVar() function.
WriteXMLData Table 2-28 WriteXMLData	Outputs the data directly in contrast to ProcessXMLData().

2.2.9 Overview of SINAMICS process variables

Table 2-5 SINAMICS variables

Variable name	Description
SINAMICS.CU	Returns the drive system: "SINAMICS S120"
SINAMICS.FirmwareVersion	Returns the firmware version e.g. "V05.10.23.00"
SINAMICS.IsFailSafe	Returns whether the CF card has a failsafe update. Possible values: "Yes" or "No"
SINAMICS.Status	Returns whether the device must be commissioned for the first time. Possible values: 0 = Yes 2 = No
SINAMICS.SysTime	Returns the current system time in [ms]

2.2.10 MWSL functions

AddHTTPHeader

Table 2-6 AddHTTPHeader

Syntax	AddHTTPHeader (<http-header>)	
	<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. 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>	
Parameter	<http-header>	<p>Character string that ends with \r\n.</p> <p>If multiple HTTP headers are to be entered (only possible with Set-cookie), the individual headers must be separated by \r\n.</p>
Example	<pre><MWSL> var strCookie; strCookie = "Set-cookie: siemens_automation_language=de"; AddHTTPHeader(strCookie); </MWSL> <html> <head> <title> MWSL Function AddHTTPHeader </title> </head> <body> <h2> Testpage </h2></pre>	

2 Basic information

	<pre></body> </html></pre>
--	----------------------------------------

CreateGUID

Table 2-7 CreateGUID

Syntax	<pre>CreateGUID()</pre> <p>Generates a unique alphanumeric ID in the system</p>
Parameter	
Example	<pre><MWSL> write(CreateGUID()); </MWSL></pre>
Output	5022420B-02A7-0000-B362-3B7F4E87148D

DecodeString

Table 2-8 DecodeString

Syntax	<pre>DecodeString(<string>)</pre> <p>Converts a string encoded with EncodeString back to its original.</p>	
Parameter	<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	<pre>Original: Straße Flüsse Gelände Vögel Encoded: Stra&#xdf;e&#x20;Fl&#xfc;sse&#x20;Gel&#xe4;nde&#x20;V&#xf6;gel Decoded: Straße Flüsse Gelände Vögel</pre>	

die

Table 2-9 die

Syntax	<pre>die(<Param0>,<Param1>,...)</pre> <p>Break program execution</p>	
Parameter	<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 ---	

EncodeString

Table 2-10 EncodeString

Syntax	EncodeString(<string> Replaces special characters by their URL-coded hex value (%hh).	
Parameter	<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	

ExistFile

Table 2-11 ExistFile

Syntax	ExistFile(<file name> Checks whether a file with the name <file name> exists. The function returns the file length as the returned value. Files that are associated with the MWSL compiler (*.mwsl, *.js, *.css, ...) the ExistFile - call should always be made to the compiled file (*.cms).	
Parameter	<file name>	Name of the sought file. The file path refers to the root directory of the user: OEM/SINAMICS/HMI.
Example	<pre><MWSL> var tmpLength = ExistFile("/USERFILES/test.mwsl.cms"); write("File length:"+ tmpLength); </MWSL></pre>	
Output	File length: 38	

ExistVariable

Table 2-12 ExistVariable

Syntax	ExistVariable(<variable name>, <variable source> This function queries the presence of a variable. It returns true or false.	
Parameter	<variable name>	Name of the variable.
	<variable source>	Name of the variable source. Possible values: <ul style="list-style-type: none"> • URL • HTTP • COOKIE
Example	<pre><MWSL> ExistVariable("Variable1", "URL") </MWSL></pre>	
Output	Returns true if the URL variable "Variable1" exists, otherwise false.	

GetLanguage

Table 2-13 GetLanguage

Syntax	GetLanguage() Returns the currently set language.
Parameter	
Example	<MWSL> write("The currently set language is '" + GetLanguage() + "'"); </MWSL>
Output	The currently set language is 'en'

GetVar

Table 2-14 GetVar

Syntax	GetVar(<variable name>,<variable source>,<format string>) This function returns the value of a variable from a variable source.	
Parameter	<variable name>	Name of the variable
	<variable source>	Name of the variable source Possible values: <ul style="list-style-type: none"> • URL • HTTP • PROCESS • COOKIE • DEFAULT The default setting is PROCESS If no source is stated DEFAULT is selected, that is, the variable provider.
	<format string>	The handling of the format string depends on the variable source. Thus, this property is not possible for the variable sources COOKIE and URL.
Example	<p>GetVar("Parameter","URL"); Returns the content of the URL – variable: Parameter.</p> <p>GetVar("Username","HTTP"); Returns the content of the HTTP – variable: Username.</p> <p>GetVar("Accept-Language","HTTP","?-"); Returns the content of the HTTP – variable: Accept-Language. The format string "?-" indicates that all characters up to the first occurrence of the "-" character are returned.</p> <p>GetVar("SINAMICS.FirmwareVersion") Returns the firmware version. See also: Table 2-5 SINAMICS variables</p> <p>GetVar("1.Params.18","PROCESS") or alternative, because PROCESS is the default variable source: GetVar("1.Params.18") Returns the value of the parameter 18 of the drive object with the number 1.</p> <p>GetVar("2.Params.63","PROCESS","%3.2f"); Returns the value of the parameter: 63 of the drive object with the number 2. The format 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</p>	

2 Basic information

	point will be displayed. The variable source ("PROCESS") is mandatory if the format string parameter is used.
--	------------------------------------------------------------------------------------------------------------------

InsertFile

Table 2-15 InsertFile

Syntax	InsertFile(<file name>)	
	<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 associated with the MWSL compiler (*.mws1, *.msl, *.xsl, *.js, *.xmlf, *.css) the MWSL scripts it contains will be run.</p> <p>URL parameters can be passed with usual syntax (<file name>?<parameter>=<value>).</p>	
Parameter	<file name>	Name of text file, including path
Example	<pre><HTML> <BODY> [...] <MWSL> if(ExistFile("/USERFILES/TMPL/Output.mws1.cms") > 0){ InsertFile("/USERFILES/TMPL/Output.mws1?myparam=123"); } </MWSL> [...] </BODY> </HTML></pre>	
Output	In the html page the content of the file Output.mws1 is inserted and displayed in HTML format.	

isFinite

Table 2-16 isFinite

Syntax	isFinite(<value>)	
	Returns false if the passed value is NaN or infinite.	
Parameter	<value>	Value to check
Example	<pre><MWSL> write("Test of the number 123456 - isFinite: "); write(isFinite(123456) + "
"); write("Test of NaN - isFinite: "); write(isFinite(parseInt(2147483647))); </MWSL></pre>	
Output	Test of the number 123456 - isFinite: 1 Test of NaN - isFinite: 0	

2 Basic information

isNaN

Table 2-17 isNaN

Syntax	isNaN(<value>)	
	Checks whether the passed value is an invalid double.	
Parameter	<value>	Value to check
Example	<pre><MWSL> write("Test of 123456 - isNaN: "); write(isNaN(123456) + "
"); write("Test of NaN - isNaN: "); write(isNaN(parseInt(2147483647))); </MWSL></pre>	
Output	<pre>Test of 123456 - isNaN: 0 Test of NaN - isNaN: 1</pre>	

IsSSL

Table 2-18 IsSSL

Syntax	IsSSL()	
	Returns true if the client is connected to the server via an SSL connection	
Parameter		
Example	<pre><MWSL> write("Is the client connected via a SSL connection: "); write(IsSSL()); </MWSL></pre>	
Output	<pre>Is the client connected via a SSL connection: 1</pre>	

parseFloat

Table 2-19 parseFloat

Syntax	parseFloat(<string>)	
	Conversion of a string to a double value	
Parameter	<string>	String to be converted
Example	<pre><MWSL> var a = parseFloat("10") + "
"; var b = parseFloat("10.00") + "
"; var c = parseFloat("10.33") + "
"; var d = parseFloat("34 45 66") + "
"; var e = parseFloat(" 60 ") + "
"; var f = parseFloat("40 years"); write(a + b + c + d + e + f); </MWSL></pre>	
Output	<pre>10 10 10.33 34 60 40</pre>	

parseInt

Table 2-20 parseInt

Syntax	parseInt(<value>,<base>) Conversion of a string to an integer value	
Parameter	<value>	String to be converted If a value starts with 0x, it will be interpreted as hexadecimal. Values starting with 0 will be interpreted as octal. All other values are interpreted in decimal format. Maximum value: 2147483646 (0x7FFFFFFE) Minimum value: -2147483647 (-0x7FFFFFFF) If values exceed the upper limit NaN will be returned. If the value shall be interpreted as a negative number a "-" has to be 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 = "101"; var tmpVar1 = "100"; var tmpSum = tmpVar0 + tmpVar1; write(tmpSum + "
"); var tmpVarInt0 = parseInt(tmpVar0,"2"); var 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 + "
"); tmpVar0 = "ABC"; tmpVarInt0 = parseInt(tmpVar0,"16"); write(tmpVarInt0 + "
"); tmpVarInt0 = parseInt(tmpVar0); write(tmpVarInt0 + "
"); tmpVar0 = "0x7FFFFFFF"; write(parseInt(tmpVar0)); </MWSL> </pre>	
Output	<pre> 201 9 AB 21 2748 NaN </pre>	

ProcessXMLData

Table 2-21 ProcessXMLData

<p>Syntax</p>	<p><code>ProcessXMLData (<DATA>, <TEMPLATE>)</code></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 is 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 add 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 you can find in the SIMOTION IT Programming and Web Services Documentation: https://support.industry.siemens.com/cs/ww/en/view/109757319</p>	
<p>Parameter</p>	<p><DATA></p>	<p>Data for the dynamic HTML file A file or a variable containing the data can be passed as a parameter. File: "<EXTERNAL SRC=\"/datafile.xml \"/>", in which <code>datafile.xml</code> is the file containing the data. Variable: <variable name> Specifies the variable name.</p>
	<p><TEMPLATE></p>	<p>Template (data format) A file or a variable containing the templates can be passed as a parameter. File: "<TEMPLATES><EXTERNAL SRC=\"/Template.xml\"/> </TEMPLATES>", in which "<code>Template.xml</code>" is the file that contains the templates. Variable: <variable name> Specifies the variable name.</p>
<p>Example</p>	<p><code>ProcessXMLData("<EXTERNAL SRC=\"/USERFILES/variables.xml \"/>", "<TEMPLATES><EXTERNAL SRC=\"/USERFILES/variablesTemplate.xml\"/></TEMPLATES>");</code></p>	

© Siemens AG 2018 All rights reserved

ReadFile

Table 2-22 ReadFile

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

ReplaceString

Table 2-23 ReplaceString

Syntax	ReplaceString(<variable name>,<search pattern>,<replacement string>) Replacing strings.	
Parameter	<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 = "SINAMICS S120"; var tmpOutString; tmpOutString = ReplaceString(tmpString,"I","i"); write("Result: " + tmpOutString); </MWSL></pre>	
Output	Result: SiNAMiCS S120	

SetVar

Table 2-24 SetVar

Syntax	SetVar(<variable name>,<value>) This functions sets process parameters.	
Parameter	<variable name>	Name of the variable For syntax see: GetVar with "PROCESS" source
	<value>	New parameter value
Example	<pre>SetVar("1.Params.3",3); Sets the parameter: "3" of the drive object with the number: "1" (Control Unit) to the value: "3". SetVar("CU_S.Params.977",1); Sets the parameter: "977" of the drive object with the name: "CU_S" (Control Unit) to the value: "1". SetVar("2.Params.10",2); Sets the parameter: "10" of the drive object with the number: "2" to the value: "2". SetVar("Drive_2.Params.10",0); Sets the parameter: "10" of the drive object with the name: "Drive_2" to the value: "2".</pre>	

ShareRealm

Table 2-25 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 true or false	
Parameter	<group>	Following parameters are valid: <ul style="list-style-type: none"> • NO_REALM No group association • ANY_REALM Any group association • Sinamics Group SINAMICS • Administrator Group Administrator
Example	<pre><MWSL> if (ShareRealm("Administrator")){ write("Successfully logged in as Administrator"); } </MWSL></pre>	
Output	If the user is logged in as Administrator, the instruction in brackets is executed	

write

Table 2-26 write

Syntax	write(<text>) The write function writes text to the output of a HTML page.	
Parameter	<text>	Text, return values of functions, or variable contents can be passed
Example	<pre><MWSL> write("Hello World!"); Output: Hello World write(GetVar("Parameter", "URL")); Outputs the content of the URL variable "Parameter" var string="123"; write(string); Outputs the variable "string": 123 </MWSL></pre>	

WriteVar

Table 2-27 WriteVar

Syntax	<p>WriteVar(<variable name>,<variable source>,<format string>)</p> <p>The function WriteVar is similar to GetVar but writes the content of a variable directly to the output. WriteVar is equivalent to the call: write(GetVar(...))</p>	
Parameter	<variable name>	Name of the Variable
	<variable source>	See Table 2-14 GetVar
	<format string>	See Table 2-14 GetVar
Example	<pre><MWSL> WriteVar("Parameter","URL"); Outputs the content of the URL variable "Parameter" WriteVar("1.Params.2") Outputs the value of the parameter 2 of the drive object with the number: 1 (Control Unit) </MWSL></pre>	

WriteXMLData

Table 2-28 WriteXMLData

Syntax	<p>WriteXMLData(<DATA>,<TEMPLATE>)</p> <p>WriteXMLData outputs the data in contrast to ProcessXMLData directly. write(ProcessXMLData(...)); is equivalent to WriteXMLData(...);</p>	
Parameter	<DATA>	See Table 2-21 ProcessXMLData
	<TEMPLATE>	See Table 2-21 ProcessXMLData
Example	<pre>WriteXMLData("<EXTERNAL SRC=\""/USERFILES/variables.xml \"/>", "<TEMPLATES><EXTERNAL SRC=\""/USERFILES/variablesTemplate.xml\"/></TEMPLATES>");</pre>	

2.3 JavaScript

JavaScript is a script language that is predominantly used for dynamic web pages.

JavaScript makes it possible to evaluate user actions and change, reload or generate the content of a web page, therefore expanding the options of HTML. As a result of the functionality provided by JavaScript, a dynamic page can be generated from basic, static HTML pages, which then responds more like an application rather than just displaying text.

All of the actions executed by JavaScript are purely restricted to accessing data, which the web page contains. This prevents that JavaScript applications access data on the user's hard disk and possibly manipulate this data. This technique is also referred to as "sandbox".

2.3.1 Integrating JavaScript into HTML

In order to be able to integrate JavaScript into an HTML page, for the code, a script area must be defined.

This area is limited by the `<script>` tags. In order that the interpreter knows the script type involved, in the opening `<script>` tag, the `type` attribute must be given the information "text/JavaScript" for JavaScript. Everything, that is located in this area, is interpreted as JavaScript.

The script area can either be located in the head area of the HTML file (between `<head>` and `</head>`), or at any location in the body area of the web page (between `<body>` and `</body>`).

It is possible to define several script areas in one HTML page. From a script area, it is also possible to access functions from other script areas, as long as these areas are located on the same HTML page.

Example

```
<html>
  <head>
    <script type="text/javascript">
      function double(item) {
        return (item * 2);
      }
    </script>
  </head>
  <body>
    <script type="text/javascript">
      document.write(double(3));
    </script>
  </body>
</html>
```

This HTML code returns a web page with the content "6".

The JavaScript instruction `document.write()` in the body area of the page, calls a `double()` function (see Chapter 2.3.5: [Functions](#)) with the transfer parameter "3", which was defined in another script area of the page. This function doubles the transferred parameters, and returns the result. The supplied value is written to the document using the `document.write()` instruction.

2.3.2 The Document Object Model (DOM)

JavaScript is used in HTML pages to access HTML elements, and to manipulate these in some form or another.

These HTML objects are accessed using the `Document Object Model (DOM)`.

In DOM, all objects of an HTML document are classified according to a hierarchic tree-like structure and are seen as nodes.

Each of these nodes provides certain access methods to secure access to its elements and attributes.

In JavaScript, the "window" object represents the uppermost node.

From this object, the browser window can be accessed as well as all of the objects contained in it.

The "document" object – the HTML file itself – is located below this "window".

Via this object, all of the elements contained in the web page can be accessed, as long as these have been assigned the `id` or `name` attribute.

Example

```
<input type="text" name="input" value="">
<input type="button" value="write"
onClick="window.document.input.value = 'Hello'">
```

In the example above, a text field with the name "input" is created, whose content is not pre-assigned. Further, a button is created with the "write" label.

When clicking this button, using JavaScript the `<input>` element is accessed, and its content ("value") is modified (see Chapter 2.3.4: [Event handler](#)). The word "Hello" can then be seen in the text field.

The tree-like structure is then scanned until the selected element is reached – starting at the "window" object, through "document", "input" up to "value" object. The addressed object refers to the name, which was assigned to the element in the `name` attribute.

When accessing elements in an HTML page, addressing can either use absolute or relative path data.

This means that each element can be directly addressed from the root of the DOM tree (absolute path data) or relative with respect to itself.

If a part of the DOM tree is accessed from elements, which are located on the same "branch", then the access to the target object can also only be realized from the nodes, which serve both elements as the lowest common node.

Absolute addressing of elements in a web page is possible at any time and from any element. In so doing, the path from the root of the DOM train – from the "window" or "document" object – is referenced along the nodes up to the selected target object.

2.3.3 getElementById

Using `getElementById` it is possible to access HTML elements, if these were first assigned the `id` attribute.

For example, this allows that their size, alignment or even content can be changed. In this case, it must be noted that the assigned `id` name must be unique in the particular HTML page, i.e. any `id` must only occur once per page.

Example

```
<input type="text" value="" id="text" size="15" />
<table>
  <tr>
    <td>The value is:</td>
    <td id="valueColumn"></td>
  </tr>
</table>
<input type="button" value="Copy" onclick="copy()" />
<script type="text/javascript">
  function copy() {
    document.getElementById('valueColumn').innerHTML =
      document.getElementById('text');
  }
</script>
```

In the example above, an input field is allocated the `id`-attribute, which is assigned the "text" value. This is followed by a single-row table, in which the first field is assigned the text "The value is:". The `id="valueColumn"` is assigned to the second field that is still empty. Finally, a button is created, which when clicked, calls the JavaScript function `copy()`. In this function, the content of the input field is copied into the empty table cell using `getElementById`.

Note Please observe the different modes of access to the content of the particular element!

For `<input>` elements, their display values can be accessed using the `value` (`document.getElementById(...).value`) attribute, while for table cells, the `innerHTML` attribute must be addressed (`document.getElementById(...).innerHTML`).

Note You can find further information on this topic at the following links:

https://www.w3schools.com/jsref/met_document_getelementbyid.asp

2.3.4 Event handler

Event handlers handle events.

Events can include mouse clicks, entries into text fields, clicking buttons or similar. Using event handlers, a JavaScript technique can be called, which correspondingly responds to an event.

As a consequence, event handlers form the interface between HTML and JavaScript.

These can be identified at their prefix "on". They are noted as attributes in HTML elements, which should respond to a specific event.

A reaction is assigned to the event handler. These can either comprise individual JavaScript instructions, or a function defined in JavaScript – if several JavaScript instructions are to be executed one after the other.

For HTML programming, the following event handlers are most frequently used:

- `onload` When loading a file
- `onclick` When clicking (e.g. a button)
- `onchange` When a change has been made (e.g. selecting an entry of a selection list or changing the value of a text field)

Note

You can find a detailed list of all of the available event handlers, including examples, at the following link:

https://www.w3schools.com/tags/ref_eventattributes.asp

2.3.5 Functions

Using JavaScript, it is possible to define `functions`.

JavaScript instructions can be noted in these functions, whose execution is started when calling the function.

A function only starts with the `function` keyword and a function name, which the user can freely allocate.

This function name must be unique within an HTML document.

The function is called using a previously defined name.

Function calls can also be located in event handlers, for example to respond to user inputs; they can also be noted in other functions, in order to act as help function. It is always possible to access functions in other script areas, as long as these areas are located on the same HTML page. Just the same as in other programming languages, it is always possible to transfer parameters to JavaScript functions – or to have values returned from these.

All JavaScript instructions in an HTML file, which are not located in a function, are immediately executed by the JavaScript interpreter of the web browser as soon as the file is loaded.

Note

You can find further information on this topic at the following link:

https://www.w3schools.com/js/js_functions.asp

2.3.6 Libraries

Collections of JavaScript functions can be integrated in HTML pages using JavaScript `libraries`. The functions defined in the libraries can then be called in script areas of the web page.

The advantage of this technique is that frequently used functions can be encapsulated in a file, which then only have to be integrated once in the appropriate HTML document. As a consequence, when these functions are used a multiple number of times in different HTML pages, they do not have to be redefined each time.

Integrating JavaScript libraries in HTML is realized in almost precisely the same way as when integrating JavaScript. A script area must also be defined in this case.

This area is limited using the `<script>` tags. In order that the interpreter knows the script type involved, in the opening `<script>` tag, the `type` attribute must be given the information "text/JavaScript" for JavaScript. Everything, that is located in this area, is interpreted as JavaScript.

The difference to standard script areas is the fact that no functions can be defined in this area, but using the `src` attribute, the interpreter is told the path under which the library can be found.

The path should be specified either relative or absolute to the HTML document.

Example:

```
<head>
  <script src="scripts/Library.js" type="text/javascript">
  </script>
</head>
```

In the example above, in an HTML document, the JavaScript library "Library.js" is integrated in the opening `<script>` tag using the `src` attribute.

The path is specified, relative to the document.

It is now possible to access the functions that are saved in the "Library.js" library in every other script area of this HTML page.

2.3.7 Variable types

Different than in many other programming languages, variables in JavaScript are not rigidly linked to a particular `variable type`.

Although a distinction is made between several basic types, for example numbers, character strings or truth values, this assignment does not have to be maintained across the board.

For example, a variable that was initialized with a numerical value, in the course of a function, can be assigned a character string without previous conversion being required.

The value must be transferred in quotation marks in order to assign a variable a character string. These quotation marks must be omitted if numerical values are assigned. The keywords "false" or "true" are available for truth values.

Example

```
<script type="text/javascript">
  var item1 = 0;
  var item2 = "string";
  var item3 = false;

  item1 = item2; //item1: string
  item2 = item3; //item2: false
  item3 = 0;    //item3: 0
</script>
```

In this script area, three variables are defined and each immediately initialized with a number, a character string and a truth value. After this, the contents of the variables are exchanged between one another, without it being necessary to explicitly convert the variables.

The fact that the type of a variable can be interpreted differently, can, under certain circumstances, result in unpredictable results. This is because, for example, a variable read-in via a text field can be interpreted as character string as well as number depending on the particular processing function.

If a number is doubled using the mathematical function "*" (multiply) (i.e. its value is multiplied by two), then this is interpreted as numerical value, and twice the value of the number is output as result.

If a number is doubled using the mathematical function "+" (add) (i.e. its value is added to itself), then the number is also interpreted as numerical value, and twice the value of the number is output as result (e.g. $15 + 15 = 30$).

If a character string is used instead of a number, then it cannot be doubled using multiplication as it does not involve a numerical value. In this case, "NaN" would appear as result, which means "Not a Number".

For a character string, if adding is used to double it, then the character string is shown twice one after the other as result (e.g. "15" + "15" = "1515").

2.4 Cascading Style Sheets (CSS)

Cascading Style Sheets (CSS) represent an indirect supplement for HTML. These involve a language to define format properties of individual HTML elements.

An important CSS function is the possibility of defining central formats. This means that central definitions can be noted in an external file regarding the appearance of an element, and this stylesheet can then be integrated into many HTML pages in parallel. All elements of the corresponding HTML files will then be allocated format properties, that were defined once at a central location. Using this procedure, design and functionality can be separated from one another.

2.4.1 Integrating CSS in HTML

In many cases, standard formats are used for several HTML files of a project. These formats can be defined in a separate text file, and this file can be integrated into any required HTML file.

If the format definition is modified in a separate file, then the modifications are implemented as standard in all files in which the separate CSS file has been integrated.

Using the `<link>` tag, a CSS file, which contains CSS format definitions, can be referenced in the file header of an HTML file.

`rel="stylesheet"` and `type="text/css"` must be located within the `<link>` tags. The path of the required file is specified using the `href` attribute.

The referenced file must be a pure text file, which should have the extension `.css`.

Note

Other options for integrating CSS in HTML files are described at the following link:

https://www.w3schools.com/css/css_howto.asp

2.4.2 Defining formats for classes

Formats for classes can be defined in CSS files, which can be accessed in HTML elements via the `class` attribute.

There are two ways of noting HTML element classes:

- for one specific HTML element type, or
- for no specific HTML element type.

Format definitions for a class always start with a point, followed by a name for the particular class.

The names after the point can be freely assigned, however they may not include

- spaces and German umlaut characters,
- not start with a digit or a hyphen,
- and should not contain an underscore and not be too long.

Example:

```
h1 {
  font-family : Arial;
  font-size : 2em;
  font-weight : normal;
}
h1.back {
  background-color : #FFFF00
}
*.back {
  background-color : #00FFFF
}
```

In the example above, using `h1.back` a class called "back" is addressed, which is only applicable for HTML elements, type `h1` (title, 1st order). Therefore, this can also be used on elements such as `<h1 class="back">`.

Used with `*`, a format definition can be noted for a class, which can be applied to all elements. Here, the star is applicable as a universal selector, however it can also be completely omitted.

2.4.3 Defining individual formats

Just as formats can be defined for classes that are addressed in an HTML file using the `class` attribute, formats can also be defined that are addressed using the `id` attribute.

As the value assignment at such an attribute should involve a unique name throughout the complete document, it therefore involves a central format definition for the one element with this `id`.

However, in CSS the `id` names are not only considered, for example in JavaScript as unique identifiers, but also as unique element type identifiers.

Example:

```
#redArea {
  position : absolute;
  top : 130px;
  left : 30px;
  width : 320px;
  padding : 10px;
  margin : 0px;
  border : 4px solid #EE0000;
}
```


2 Basic information

An individual format is defined in the example above.
Such an individual format starts with the hash sign #, followed by a name allocated by the user.
An HTML element, which uses this name as value assignment at the `id` attribute, is then allocated the appropriate formatting.

Note

You can find further information on "defining formats" at the following link:

https://www.w3schools.com/css/css_syntax.asp

3 Engineering

3.1 Project planning and configuration

The procedure to generate a user-defined web page for the SINAMICS S120 Webserver is shown in the following.

As example, a web page is created, which is structured as a pure diagnostics page. This displays the actual state of an axis of a SINAMICS S120 drive.

In addition to the sequence when creating the page, its structure is explained in more detail. An explanation is also given as to how such a user-defined web page must be converted in order to display this at the drive.

The web page is created for the virtual company "Pick and Place Master" and its machine "Pick and Place Master S120".

3.1.1 Concept

The concept used here to create a user-defined web page for the SINAMICS S120 Webserver means that the following files must be created for each new page:

- **Parameter source**

The parameter source includes all of the drive parameters, which are to be displayed on the corresponding web page.

The parameter source has a fixed structure, and under no circumstances may it be changed.

All of the parameters are read out of the drive using MWSL functions. Every parameter must be allocated a unique `ID`. Via this `ID`, access can be realized from the actual web page so that the parameter value can be displayed there.

Depending in which form the parameter value is to be displayed on the web page, the assigned `IDs` must be allocated corresponding "groups" in the parameter source.

In this case, a distinction is made between the following groups:

- **plainTextVariables**

For `IDs` that are assigned the `plainTextVariables` group, the parameter values are read out of the drive and written to the web page 1:1. This means that the value is not formatted according to a certain number of decimal places.

- **numericVariables**

For `IDs` that are assigned the `numericVariables` group, the parameter values from the drive are limited to a corresponding number of decimal places before being transferred to the web page.

In addition, for each of these parameter values, when creating the web page, a `warningLimit` and a `criticalLimit` can be optionally specified. If the value exceeds one of these limits, then this is indicated e.g. using a yellow or red background.

- **indicatorVariables**

For `IDs` that are assigned the `indicatorVariables` group, the parameters from the drive are displayed on the web page as color-coded status displays. These status displays change their color, depending on the value of the corresponding parameter.

Therefore, the group of `indicatorVariables` is used to display individual bits of a parameter or generally, boolean values.

- **gaugeVariables**

For `IDs` that are assigned the `gaugeVariables` group, the parameter values from the drive are displayed on the web page as round instrument

(`Canvas` element). This type of display is suitable, for example, to display the actual drive speed.

Labeling as well as the start and end of the round instrument scale can be specified by the user when creating the web page.

- **barVariables**

For `IDs` that are assigned the `barVariables` group, the parameter values from the drive are displayed on the web page as bars (`Canvas` element).

Labeling as well as the start and end of the bar scale can be specified by the user when creating the web page.

- **Main page**

The complete content of the web page is defined on the main page.

Here, the user can create, e.g. tables, `Canvas` elements can be inserted and also status displays created.

By assigning the `id` attribute, a parameter, previously defined in the parameter source, can be transferred to the individual HTML elements. To do this, the `id` attribute of the HTML element must correspond to the `ID` of the corresponding parameter from the parameter source. This means that the value of the corresponding parameter is displayed at precisely this location on the web page.

Further, using JavaScript functions defined by the user, it is possible to make the web page dynamic. Therefore, depending on the selection, it is possible to hide or display certain areas of the web page.

Further, the JavaScript function to cyclically update the parameters is called on the main page; this means that all parameters of the parameter source are cyclically updated at an interval that can be defined by the user.

- **Load page**

The load page only includes static content, for example the page title and an `IFrame`, via whose attribute `src` the path of a page can be specified, that is to be loaded into this area (in this particular case, the main page).

This page must always be created as a result of the inherent system properties. This is because, when directly displaying the main page in the SINAMICS S120 Webserver, scripting is not supported. If the load page is not used, then, for example it is not possible to make the web page dynamic using JavaScript. Further, it is also not possible to cyclically update the parameters of the parameter source and display these on the main page.

- **JavaScript file**

All of the JavaScript functions defined by the user are encapsulated in a JavaScript file, and this is integrated in the head area on the main page in an opening `<script>` tag. This procedure ensures that the main page and the JavaScript functions used remain transparent.

- **Stylesheet (optional)**

A CSS file can be optionally created, which contains the complete formatting of the web page. Depending on the defined format type (format for a class or individual format), the individual HTML elements can be assigned their format in this way using the `class` or `id` attribute. This is precisely the advantage if several HTML elements should have the same formatting, as the format properties only have to be defined once at a central location.

3.1.2 Files used

The following files were created for the web page sample. They will be described in more detail in the following:

- **configuration**
 - axis1.doName
 - axis1.title
- **css**
 - diagAxis.css
 - diagStatus.css
 - pickAndPlaceMaster.css
 - pickAndPlaceMasterHeader.css
- **images**
 - indicatorCritical.png
 - indicatorOff.png
 - indicatorOn.png
 - indicatorNeutral.png
- **scripts**
 - diagAxis.js
 - diagStatusAxis.js
 - libByMichael.js
- **axisRedLoader.mwsl**
- **diagAxis.mwsl**
- **diagStatusAxis.mwsl**
- **variablesDiagAxis.mwsl**
- **variablesDiagStatusAxis.mwsl**

Note

The files used for this example are included in the ZIP archive `68691599_S120_Userdefined_Webpages_V2_0.zip`, as MWSL files which must be uploaded to the drive web server.

Please note that the sample page supplied is based on the drive object with the name which is specified in the file "axis1.doName". Default is "SERVO_02".

3.1.3 Structure and content

Figure 3-1 axisRedLoader.mwsl

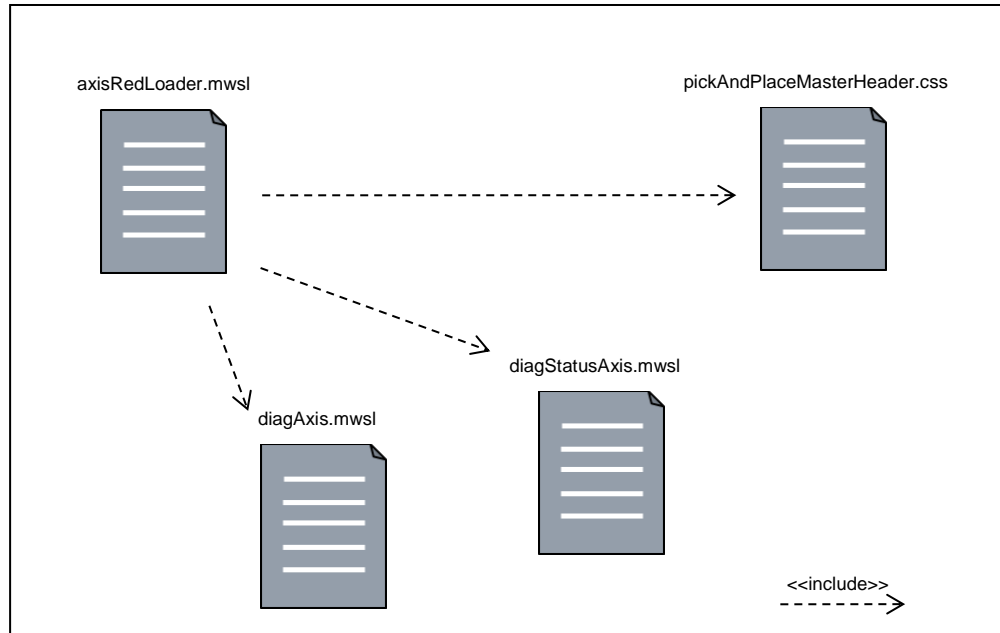


Table 3-1

File	Description	Type
axisRedLoader.mwsl	This page contains an <code>IFrame</code> , which is filled, depending on what the user selects, with the content of the <code>diagAxis.mwsl</code> or <code>diagStatusAxis.mwsl</code> page. Therefore, this page is only used to load another page.	Load page
diagAxis.mwsl	This page contains a display of the actual axis enables. The enables are visualized using status displays with different colors. Further, using two <code>Canvas</code> elements, the actual speed and torque of the axis is displayed. In addition, important parameters of the axis and their actual value are displayed in a tabular form.	Main page
diagStatusAxis.mwsl	This page includes an overview of the actual control and status words for the sequence control, faults and alarms as well as the speed controller. They are also visualized using status displays with different colors.	Main page
pickAndPlaceMaster-Header.css	The stylesheet includes the layout or the formatting for the texts displayed in the <code>axisRedLoader.mwsl</code> file.	Stylesheet

Figure 3-2 diagAxis.mwsl

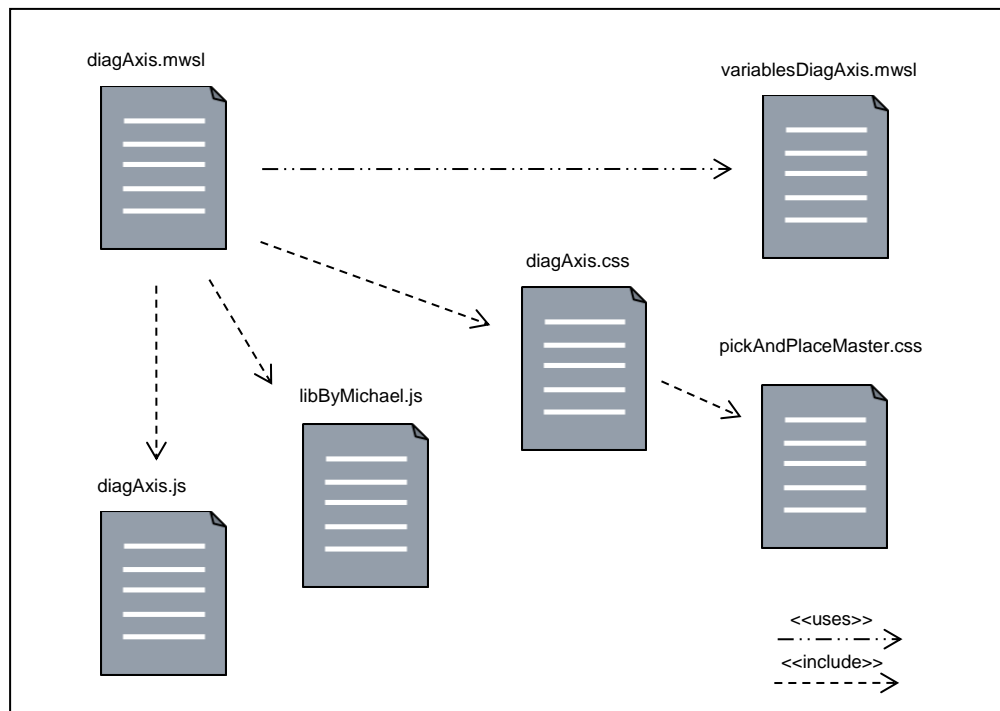


Table 3-2

File	Description	Type
diagAxis.mwsl	See Table 3-1	Main page
variablesDiag-Axis.mwsl	This source contains all of the parameters of the axis that are to be cyclically updated, and which are used in the HTML page <code>diagAxis.mwsl</code>	Parameter source
libByMichael.js	The JavaScript library includes a series of functions, which also allow parameters to the cyclically updated. <code>Canvas</code> elements are also provided. After the parameters have been updated, the parameter values are automatically written to the corresponding location in the HTML page <code>diagAxis.mwsl</code> using the library. This library can be universally used!	JavaScript library
diagAxis.js	The JavaScript library includes all of the functions defined by the user, which are used to dynamically layout the HTML page <code>diagAxis.mwsl</code> . This library is specific to a certain page!	JavaScript library
diagAxis.css	The stylesheet includes the specific formatting for the HTML page <code>diagAxis.mwsl</code>	Stylesheet
pickAndPlace-Master.css	The stylesheet includes additional formatting types, which are used across various pages; this means that these formatting types can be used for additional user-specific web pages.	Stylesheet

Figure 3-3 diagStatusAxis.mwsl

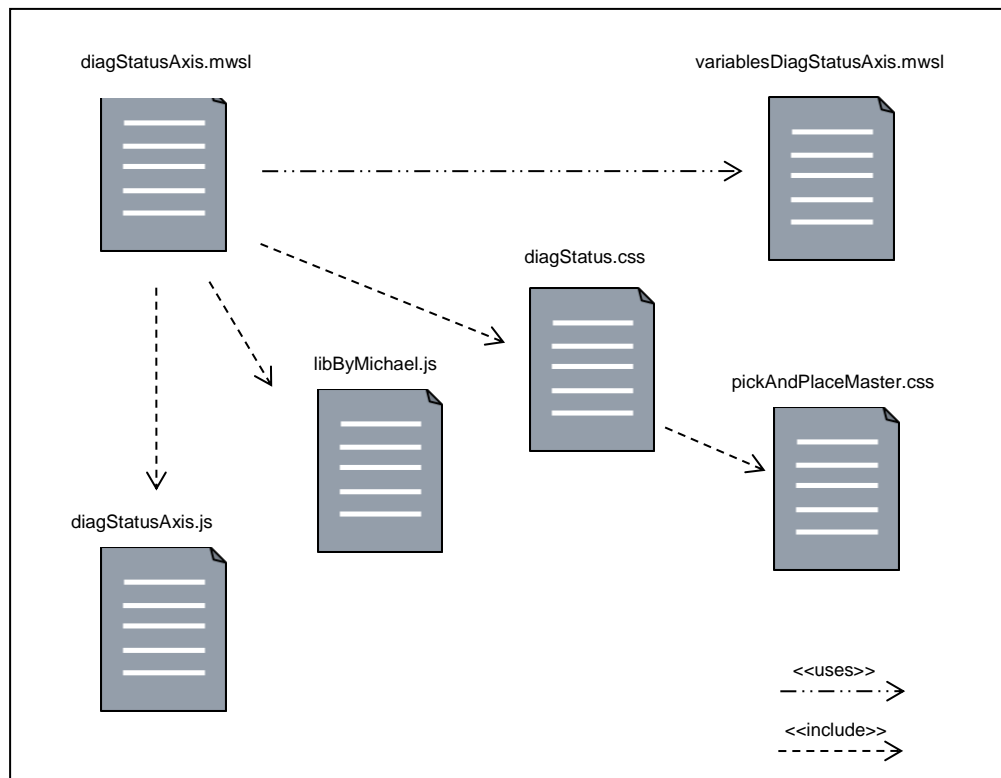


Table 3-3

File	Description	Type
diagStatusAxis.mwsl	See Table 3-1	Main page
variablesDiagStatus-Axis.mwsl	This source contains all of the parameters of the axis that are to be cyclically updated, and which are used in the HTML page <code>diagStatusAxis.mwsl</code>	Parameter source
libByMichael.js	See Table 3-2	JavaScript library
diagStatusAxis.js	The JavaScript library includes all of the functions defined by the user, which are used to dynamically layout the HTML page <code>diagStatusAxis.mwsl</code> This library is specific to a certain page!	JavaScript library
diagStatus.css	The stylesheet includes the specific formatting for the HTML page <code>diagStatusAxis.mwsl</code>	Stylesheet
pickAndPlaceMaster.css	See Table 3-2	Stylesheet

3.2 Creating a parameter source

Every parameter of the drive, which is to be subsequently displayed in the web page, must first be added to the parameter source.

The parameter source has the following, fixed structure; it is not permissible that this is changed, with the exception of the content of individual groups:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<root>
  <versions>
    <document>1</document>
    <minInterpreter>1</minInterpreter>
  </versions>
  <variables>
    <user>
      <MWSL><!--
        var PREFIX_DO_PARAMS = "",
        doName = GetVar("doName", "URL");
        PREFIX_DO_PARAMS = doName + ".Params.";
      --></MWSL>
    </user>
    <plainTextVariables>
      <!-- if no plainTextVariables are defined,
        this section has to be deleted-->
    </plainTextVariables>
    <numericVariables>
      <!-- if no numericVariables are defined,
        this section has to be deleted-->
    </numericVariables>
    <indicatorVariables>
      <!-- if no indicatorVariables are defined,
        this section has to be deleted-->
    </indicatorVariables>
    <gaugeVariables>
      <!-- if no gaugeVariables are defined,
        this section has to be deleted-->
    </gaugeVariables>
    <barVariables>
      <!-- if no barVariables are defined,
        this section has to be deleted-->
    </barVariables>
  </variables>
</root>
```

Between the <variables> or </variables> tags, groups are created, in which the IDs for the drive parameters are defined.

Here, it must be noted that the groups that are not required, must be completely deleted if no IDs were defined in these for drive parameters.

plainTextVariables or numericVariables

In order to add a new parameter of the group `plainTextVariables` or `numericVariables`, proceed as follows:

Table 3-4

No.	Action
1.	<p>Assign a new ID, which is unique for the page, for the parameter that should be read out of the drive. Insert this ID into the <code>plainTextVariables</code> or <code>numericVariables</code> group of the parameter source.</p> <pre data-bbox="469 658 1214 808"> <plainTextVariables> <valActualSpeed> <!-- MWSL code for reading parameter value--> </valActualSpeed> </plainTextVariables> </pre>
2.	<p>To read the parameter from the drive, the corresponding MWSL command must be called. Insert the MWSL code between the previously defined tags of the parameter ID.</p> <pre data-bbox="469 931 1353 1305"> <plainTextVariables> <valActualSpeed> <MWSL><!-- var PREFIX_DO_PARAMS = "", doName = GetVar("doName", "URL"); PREFIX_DO_PARAMS = doName + ".Params."; var ActualSpeed; ActualSpeed=GetVar(PREFIX_DO_PARAMS+"22","PROCESS"); write(ActualSpeed); --></MWSL> </valActualSpeed> </plainTextVariables> </pre> <p>The <code><MWSL></code> or <code></MWSL></code> tags contain the MWSL code to be executed. In this particular example, a temporary variable with the name <code>tmpActualSpeed</code> was created, which is written to using the <code>GetVar()</code> command.</p> <p>In the brackets of the command, initially the path must be specified, under which the corresponding drive parameter can be found. <code>SERVO_02</code> is the name of the drive object, this can vary. The second part of path <code>".Params."</code> is for all SINAMICS S120 drives for all drive objects identical.</p> <p>In this case, the number 22 specifies that parameter number 22 should be read out of the drive object (speed actual value in rpm, smoothed).</p> <p>Finally, the parameter value must be written to the parameter source. This is done using the MWSL command <code>write()</code>.</p>

3. Alternatively, the parameter value can be directly written to the parameter source, without having to create a temporary variable:

```
<MWSL><!--  
  //first alternative  
  write(GetVar(PREFIX_DO_PARAMS+"22", "PROCESS"));  
  
  //second alternative  
  WriteVar(PREFIX_DO_PARAMS+"22", "PROCESS");  
--></MWSL>
```

These two MWSL commands are equal options to the first solution. The parameter value is now located at precisely this location in the parameter source, and can therefore be displayed in the web page using the previously assigned parameter ID.

indicatorVariables

In order to add a new parameter of the group `indicatorVariables`, proceed as follows:

Table 3-5

No.	Action
1.	<p>Assign a new ID, which is unique for the page, for the parameter that should be read out of the drive. Insert this ID into the group <code>indicatorVariables</code> of the parameter source.</p> <pre data-bbox="470 638 1244 795"> <indicatorVariables> <indicatorOff1Enable> <!-- MWSL code for reading parameter value --> </indicatorOff1Enable> </indicatorVariables> </pre>
2.	<p>To read the parameter from the drive, the corresponding MWSL command must be called. Insert the MWSL code between the previously defined tags of the parameter ID.</p> <pre data-bbox="470 907 1372 1310"> <indicatorVariables> <indicatorOff1Enable> <MWSL><!-- var PREFIX_DO_PARAMS = "", doName = GetVar("doName", "URL"); PREFIX_DO_PARAMS = doName + ".Params."; if (_accessLevel > ACCESS_LEVEL_NONE) { write((GetVar(PREFIX_DO_PARAMS + "898") & 0x0001)); } --></MWSL> </indicatorOff1Enable> </indicatorVariables> </pre> <p>The <code><MWSL></code> or <code></MWSL></code> tags contain the MWSL code to be executed. In this example, a variable <code>PREFIX_DO_PARAMS</code> was created, which contains the generally valid part of the path for all drive parameters of the drive object <code>SERVO_02</code>. This type of variable declaration is especially practical, if several parameters are to be read out of one drive object, which have the same path.</p> <p>In the brackets of the <code>GetVar()</code> command, the previously defined path must be specified, under which the corresponding drive parameter can be found.</p> <p>In this case, the number <code>898</code> specifies that parameter number <code>898</code> should be read out of the drive object (CO/BO: control word sequence control). Using hexadecimal code <code>0x0001</code> this parameter is masked using an AND operator, i.e. here only the state of the 1st bit (bit 0) is evaluated.</p> <p>Finally, the parameter value must be written to the parameter source. This is realized using the MWSL command <code>write()</code>.</p> <p>The parameter value is now located at precisely this location in the parameter source, and can therefore be displayed in the web page using the previously assigned parameter ID.</p>

gaugeVariables

In order to add a new parameter of the group `gaugeVariables`, proceed as follows:

Table 3-6

No.	Action
1.	<p>Assign a new ID, which is unique for the page, for the parameter that should be read out of the drive. Insert this ID into the group <code>gaugeVariables</code> of the parameter source.</p> <pre data-bbox="470 571 1228 728"> <gaugeVariables> <gaugeSpeed> <!-- MWSL code for reading parameter value --> </gaugeSpeed> </gaugeVariables> </pre>
2.	<p>To read the parameter from the drive, the corresponding MWSL command must be called. Insert the MWSL code between the previously defined tags of the parameter ID.</p> <pre data-bbox="470 840 1125 1276"> <gaugeVariables> <gaugeSpeed> <value> <MWSL><!-- WriteVar(PREFIX_DO_PARAMS + "22"); --></MWSL> </value> <setValue> <MWSL><!-- WriteVar(PREFIX_DO_PARAMS + "20"); --></MWSL> </setValue> </gaugeSpeed> </gaugeVariables> </pre> <p>For parameter IDs, which belong to the parameter group <code>gaugeVariables</code> it should be noted, that a distinction can again be made between an actual value and setpoint.</p> <p>The <code><value></code> and <code></value></code> tags therefore contain the MWSL code for the actual value of the parameter ID; the <code><setValue></code> and <code></setValue></code> tags, the MWSL code for the setpoint of the parameter ID (optional).</p> <p>It is not permissible to change the name of the tags!</p> <p>The defined parameter ID <code>gaugeSpeed</code> therefore has two parameter values of the drive object, which will be subsequently displayed in the web page using two pointers in the corresponding round instruments.</p> <p>In this example, the two parameter values are read out of the drive object using the function <code>WriteVar()</code> and are simultaneously written to the parameter source.</p> <p>In the brackets of the command, the path is specified under which the corresponding drive parameter can be found (see: PREFIX_DO_PARAMS).</p> <p>In this case, the number 22 specifies that parameter number 22 should be read out of the drive object (speed actual value rpm, smoothed).</p> <p>In addition, the value of parameter number 20 is read out of the drive object (speed setpoint, smoothed).</p>

barVariables

Proceed as follows to add a new parameter of the `barVariables` group:

Table 3-7

No.	Action
1.	<p>Assign a new ID, which is unique for the page, for the parameter that should be read out of the drive. Insert this ID into the group <code>barVariables</code> of the parameter source.</p> <pre data-bbox="469 636 1230 792"> <barVariables> <barTorque> <!-- MWSL code for reading parameter value --> </barTorque> </barVariables> </pre>
2.	<p>To read the parameter from the drive, the corresponding MWSL command must be called. Insert the MWSL code between the previously defined tags of the parameter ID.</p> <pre data-bbox="469 904 1126 1178"> <barVariables> <barTorque> <value> <MWSL><!-- WriteVar(PREFIX_DO_PARAMS + "31"); --></MWSL> </value> </barTorque> </barVariables> </pre> <p>For parameter IDs, which belong to the parameter group <code>barVariables</code>, it should be observed that the MWSL code must be noted here, to read out the parameters between the <code><value></code> and <code></value></code> tags.</p> <p>It is not permissible to change the name of the tag!</p> <p>In this example, the parameter value is read out of the drive object using the function <code>WriteVar()</code> and is simultaneously written to the parameter source.</p> <p>In the brackets of the command, the path is specified under which the corresponding drive parameter can be found (see: PREFIX_DO_PARAMS). In this case, the number 31 specifies that parameter number 31 should be read out of the drive object (torque actual value, smoothed).</p>

Note

Please take additional examples for parameter IDs of the groups shown above from the source, `variablesDiagAxis.mwsl` (WEBSITES folder) in the ZIP archive `68691599_S120_Userdefined_Webpages_V2_0.zip`.

3.3 Creating the content of the web page

The complete content of the subsequent web page is saved in the main page. In order that the values of the parameters, which were previously defined in the parameter source, are also displayed, linking via the assigned parameter IDs must be realized here.

Note The individual steps are explained using the following example of the `diagAxis.mwsl` page.

3.3.1 Displaying parameter values

plainTextVariables

Table 3-8

No.	Action				
1.	<p>The link between the parameter source and the main page is realized using the parameter ID defined in the parameter source.</p> <table border="1" data-bbox="316 884 1248 1216"> <thead> <tr> <th data-bbox="316 884 922 918">Main page</th> <th data-bbox="922 884 1248 918">Parameter source</th> </tr> </thead> <tbody> <tr> <td data-bbox="316 918 922 1216"> <pre data-bbox="331 929 866 1205"><div> <table> <tr> <td>Actual Speed</td> <td id="valActualSpeed">---</td> <td>1/min</td> </tr> </table> </div></pre> </td> <td data-bbox="922 918 1248 1216"> <pre data-bbox="938 958 1232 1182"><plainTextVariables> <valActualSpeed> <MWSL><!-- ... --></MWSL> </valActualSpeed> </plainTextVariables></pre> </td> </tr> </tbody> </table>	Main page	Parameter source	<pre data-bbox="331 929 866 1205"><div> <table> <tr> <td>Actual Speed</td> <td id="valActualSpeed">---</td> <td>1/min</td> </tr> </table> </div></pre>	<pre data-bbox="938 958 1232 1182"><plainTextVariables> <valActualSpeed> <MWSL><!-- ... --></MWSL> </valActualSpeed> </plainTextVariables></pre>
Main page	Parameter source				
<pre data-bbox="331 929 866 1205"><div> <table> <tr> <td>Actual Speed</td> <td id="valActualSpeed">---</td> <td>1/min</td> </tr> </table> </div></pre>	<pre data-bbox="938 958 1232 1182"><plainTextVariables> <valActualSpeed> <MWSL><!-- ... --></MWSL> </valActualSpeed> </plainTextVariables></pre>				
2.	<p>In order to visualize the drive parameter in the main page, the <code>id</code> attribute of an HTML element must be assigned the corresponding parameter ID (here: <code>valActualSpeed</code>). The HTML element can be, for example, a table cell (see above), in which the parameter value is then displayed.</p>				

numericVariables

Table 3-9

No.	Action	
1.	The link between the parameter source and the main page is realized using the parameter ID defined in the parameter source.	
	<p style="text-align: center;">Main page</p> <pre data-bbox="331 533 975 931"> <div> <table> <tr> <td>Output Voltage</td> <td id="valOutputVoltage"> --- 300 320 </td> <td>Vrms</td> </tr> </table> </div> </pre>	<p style="text-align: center;">Parameter source</p> <pre data-bbox="1031 622 1294 842"> <numericVariables> <valOutputVoltage> <MWSL><!-- ... --></MWSL> </valOutputVoltage> </numericVariables> </pre>
2.	<p>In order to visualize the drive parameter in the main page, the <code>id</code> attribute of an HTML element must be assigned the corresponding parameter ID (here: <code>valOutputVoltage</code>).</p> <p>The HTML element can be, for example, a table cell (see above), in which the parameter value is then displayed.</p> <p>For parameter IDs that belong to the <code>numericVariables</code> group, it should be noted, that to display the values in the main page, an additional <code></code> element with the <code>class="value"</code> attribute is required.</p> <p>For example, if a column of the table is to have a color background as soon as the value exceeds a specific limit, then this can be optionally implemented using additional <code></code> elements with the <code>class="warningLimit"</code> as well as <code>class="criticalLimit"</code> attributes. The limit values are noted between the appropriate tags.</p> <p>If the parameter value exceeds the limit that was defined for <code>warningLimit</code>, then the corresponding table column has a yellow background and for <code>criticalLimit</code> red.</p>	

indicatorVariables

Table 3-10

No.	Action	
1.	The link between the parameter source and the main page is realized using the parameter ID defined in the parameter source.	
	<p style="text-align: center;">Main page</p> <pre data-bbox="331 459 893 705"><div> <div> <p>OFF1 Enable</p> </div> </div></pre>	<p style="text-align: center;">Parameter source</p> <pre data-bbox="957 474 1295 689"><indicatorVariables> <indicatorOff1Enable> <MWSL><!-- ... --></MWSL> </indicatorOff1Enable> </indicatorVariables></pre>
2.	<p>In order to visualize the drive parameter in the main page, the <code>id</code> attribute of an HTML element must be assigned the corresponding parameter ID (here: <code>indicatorOff1Enable</code>).</p> <p>In this case, the HTML element must be an image (see above), i.e. in this particular case, a status display whose color changes depending on the parameter value.</p> <p>The <code>src</code> attribute is used to define which image is displayed if the main page is subsequently called in the drive. The text, which should be displayed if the appropriate image is not available, is transferred to the <code>alt</code> attribute.</p> <p>The text, which should be subsequently located next to the status display, is noted between the <code><p></code> and <code></p></code> tags.</p>	
3.	<p>Normally, individual bits of a parameter can be evaluated using status displays, so that the value of the corresponding bit is visualized using different colors.</p> <p>If the bit has a value of "0", then a <u>red</u> status displays is shown.</p> <p>If the bit has a value of "1", then a <u>green</u> status display is shown.</p>	
4.	<p>With appropriate configuration a <u>blue</u> status display can be shown in the application.</p> <p>To do this, in the relevant parameter source, the actual value of the corresponding bit is read out ("0" or "1"), and depending on this the value "3" is written to the HTML page.</p> <p><u>Example:</u></p> <pre data-bbox="319 1350 1053 1720"><controlWordAxisCommandOpenBrake> <MWSL><!-- if (_accessLevel > ACCESS_LEVEL_NONE) { if (((controlWordAxis & 0x0080) >> 7) == 0){ write(INDICATOR_NEUTRAL); } else{ write(INDICATOR_ON); } } --></MWSL> </controlWordAxisCommandOpenBrake></pre> <p>In the example above, bit 7 of the status word (ZSW1) of the axis is checked for its value. If it is "0", the value "3" is written to the HTML page, thus creating a blue status display.</p> <p>If bit 7 has a value of "1", then its value is written to the HTML page, and therefore a green status display is shown.</p> <p>In this particular example, the <code>INDICATOR_NEUTRAL</code> is pre-assigned with the value "3", the <code>INDICATOR_ON</code> variable with "1".</p>	

gaugeVariables

Table 3-11

No.	Action				
1.	<p>The link between the parameter source and the main page is realized using the parameter ID defined in the parameter source.</p> <table border="1" data-bbox="316 488 1214 972"> <thead> <tr> <th data-bbox="316 488 791 526">Main page</th> <th data-bbox="791 488 1214 526">Parameter source</th> </tr> </thead> <tbody> <tr> <td data-bbox="316 535 791 972"> <pre data-bbox="331 629 730 875"><div> <div> <canvas id="gaugeSpeed" width="160" height="160"> </canvas> </div> </div></pre> </td> <td data-bbox="791 535 1214 972"> <pre data-bbox="807 535 1046 972"><gaugeVariables> <gaugeSpeed> <value> <MWSL><!-- ... --></MWSL> </value> <setValue> <MWSL><!-- ... --></MWSL> </setValue> </gaugeSpeed> </gaugeVariables></pre> </td> </tr> </tbody> </table>	Main page	Parameter source	<pre data-bbox="331 629 730 875"><div> <div> <canvas id="gaugeSpeed" width="160" height="160"> </canvas> </div> </div></pre>	<pre data-bbox="807 535 1046 972"><gaugeVariables> <gaugeSpeed> <value> <MWSL><!-- ... --></MWSL> </value> <setValue> <MWSL><!-- ... --></MWSL> </setValue> </gaugeSpeed> </gaugeVariables></pre>
Main page	Parameter source				
<pre data-bbox="331 629 730 875"><div> <div> <canvas id="gaugeSpeed" width="160" height="160"> </canvas> </div> </div></pre>	<pre data-bbox="807 535 1046 972"><gaugeVariables> <gaugeSpeed> <value> <MWSL><!-- ... --></MWSL> </value> <setValue> <MWSL><!-- ... --></MWSL> </setValue> </gaugeSpeed> </gaugeVariables></pre>				
2.	<p>In order to visualize the drive parameter in the main page, the <code>id</code> attribute of an HTML element must be assigned the corresponding parameter ID (here: <code>gaugeSpeed</code>).</p> <p>In this case, the HTML element must be a <code>Canvas</code> element (see above), where the required size in the web page is specified using the <code>width</code> and <code>height</code> attributes.</p>				

barVariables

Table 3-12

No.	Action				
1.	<p>The link between the parameter source and the main page is realized using the parameter ID defined in the parameter source.</p> <table border="1" data-bbox="316 1449 1214 1780"> <thead> <tr> <th data-bbox="316 1449 791 1487">Main page</th> <th data-bbox="791 1449 1214 1487">Parameter source</th> </tr> </thead> <tbody> <tr> <td data-bbox="316 1496 791 1780"> <pre data-bbox="331 1518 730 1765"><div> <div> <canvas id="barTorque" width="160" height="160"> </canvas> </div> </div></pre> </td> <td data-bbox="791 1496 1214 1780"> <pre data-bbox="807 1496 1046 1780"><barVariables> <barTorque> <value> <MWSL><!-- ... --></MWSL> </value> </barTorque> </barVariables></pre> </td> </tr> </tbody> </table>	Main page	Parameter source	<pre data-bbox="331 1518 730 1765"><div> <div> <canvas id="barTorque" width="160" height="160"> </canvas> </div> </div></pre>	<pre data-bbox="807 1496 1046 1780"><barVariables> <barTorque> <value> <MWSL><!-- ... --></MWSL> </value> </barTorque> </barVariables></pre>
Main page	Parameter source				
<pre data-bbox="331 1518 730 1765"><div> <div> <canvas id="barTorque" width="160" height="160"> </canvas> </div> </div></pre>	<pre data-bbox="807 1496 1046 1780"><barVariables> <barTorque> <value> <MWSL><!-- ... --></MWSL> </value> </barTorque> </barVariables></pre>				
2.	<p>In order to visualize the drive parameter in the main page, the <code>id</code> attribute of an HTML element must be assigned the corresponding parameter ID (here: <code>barTorque</code>).</p> <p>In this case, the HTML element must be a <code>Canvas</code> element (see above), where the required size in the web page is specified using the <code>width</code> and <code>height</code> attributes.</p>				

3.3.2 Updating parameter values

There are two options for updating parameter values:

- **Single update**
For a single update, after calling the web page, the required parameter values are read out of the drive once using a JavaScript function and written into the web page.
This type of update only makes sense for parameters whose values do not change during operation (e.g. configuration data of interfaces).

Table 3-13

No.	Action
1.	<p>To update parameter values once (i.e. the parameter source), two JavaScript functions are available, which are defined in the JavaScript library <code>libByMichael.js</code>.</p> <pre data-bbox="319 705 1276 952">function updateValues() { /* first function for updating values */ updateDocument.updateValues('variablesDiagAxis.mwsl'); /* second function for updating values */ updateDocument.updateValuesEx('variablesDiagAxis.mwsl', null, null); }</pre> <p>Both function calls are shown in the example above.</p> <p>The <code>updateValues()</code> function can be executed, e.g. after calling the particular web page, which means that the parameter values are updated once. To achieve this, the function call must be noted in the <code>onload</code> event handler of the <code><body></code> tag of the web page.</p> <pre data-bbox="319 1131 758 1232"><body onload="updateValues()"> ... </body></pre>
2.	<p>The name of the parameter source is transferred as parameter to the <code>updateDocument.updateValues()</code> or <code>updateDocument.updateValuesEx()</code> function (in this case: <code>variablesDiagAxisRed.mwsl</code>).</p> <p>The difference between the two functions is that the function <code>updateDocument.updateValuesEx()</code> can transfer two additional parameters. The second transfer parameter is used to update <code>Canvas</code> elements, the third transfer parameter is used to call what is known as a callback function, which checks as to whether the parameter values have been successfully updated.</p> <p>If these transfer parameters are pre-assigned the value <code>zero</code> – as is the case in the example above – then the particular functionality is not executed.</p>

Note

For additional information on updating canvas elements, please observe the information in Chapter 3.3.4: [Initializing and updating canvas elements](#).

- **Cyclic update**

For a cyclic update, after calling the web page, the required parameter values are cyclically read out of the drive using a JavaScript function, and written to the web page. Users can specify the update interval at the JavaScript function. As a consequence, this type of update makes sense for parameters whose values change in operation (e.g. control/status words of the drive).

Table 3-14

No.	Action
1.	<p>To cyclically update parameter values (i.e. the parameter source), the already known function <code>updateDocument.updateValues()</code> or <code>updateDocument.updateValuesEx()</code> can be noted in the function <code>setInterval()</code> provided by JavaScript.</p> <pre data-bbox="363 712 1369 936">function updateValues() { /* first function for updating values */ setInterval(function () { updateDocument.updateValues(...)}, 3000); /* second function for updating values */ setInterval(function () { updateDocument.updateValuesEx(...)}, 3000); }</pre> <p>Both function calls are shown in the example above.</p> <p>The <code>updateValues()</code> function can be executed, e.g. after calling the particular web page, which means that the parameter values are cyclically updated. To achieve this, the function call must be noted in the <code>onload</code> event handler of the <code><body></code> tag of the web page.</p> <pre data-bbox="363 1182 805 1272"><body onload="updateValues()"> ... </body></pre>
2.	<p>The difference to the single update of parameter values is the fact that a time in milliseconds can be specified at the <code>setInterval()</code> function; after this time expires, the <code>updateDocument.updateValues()</code> or <code>updateDocument.updateValuesEx()</code> function can be called again. As a consequence, the parameter values are read out of the drive again and written to the web page.</p>

Note

Depending on the number of parameters to be read out of the drive, as well as the actual system conditions, the time should be selected, which is specified using the `setInterval()` function.

The time should not be less than **1000ms**, as otherwise, errors can occur when updating the parameter values.

Note

You can find additional information on the `setInterval()` JavaScript function at the following link:

https://www.w3schools.com/jsref/met_win_setinterval.asp

3.3.3 Using the callback function

The callback function can be used to identify whether the parameter values were successfully updated. If this is the case, then an additional JavaScript function can be called (e.g. to format these parameter values).

Table 3-15

No.	Action
1.	<p>The name of the callback function can be freely selected, and must be transferred as third parameter to the <code>updateDocument.updateValuesEx()</code> function (in this case: <code>checkUpdate</code>).</p> <pre data-bbox="363 651 1305 779">function updateValues() { updateDocument.updateValuesEx('variablesDiagAxis.mws1', null, checkUpdate); }</pre>
2.	<p>The callback function <code>checkupdate</code> is automatically called as soon as the <code>updateDocument.updateValuesEx()</code> function was executed.</p> <pre data-bbox="363 902 1118 1312">function checkUpdate(finishStateofRequest) { var STATES_REQUEST_CALLBACK = { REQUEST_NOT_SUCCESSFUL: 0, REQUEST_SUCCESSFUL: 1, REQUEST_ABORTED: 2 }; if (finishStateofRequest === STATES_REQUEST_CALLBACK.REQUEST_SUCCESSFUL) { formatValues(); updateValues(); } }</pre> <p>In doing so, the JavaScript library <code>libByMichael.js</code> is internally evaluated, as to whether the parameter values were successfully updated – or not. The result is then located in <code>finishingStateOfRequest</code> transfer parameter. If this parameter has a value of 1 (<code>REQUEST_SUCCESSFUL</code>), then another JavaScript function, defined by the user, can be called (in this case: <code>formatValues()</code>).</p> <p>This procedure ensures that valid parameter values are available, if these are then to be subsequently processed.</p>

3.3.4 Initializing and updating canvas elements

In order that `Canvas` elements can always be displayed, these must be initialized using JavaScript after calling the appropriate web page.

Table 3-16

No.	Action
1.	<p>The JavaScript function to initialize <code>Canvas</code> elements must be called after loading the web page.</p> <pre data-bbox="360 546 762 645"><body onload="setupPage()"> ... </body></pre> <p>This can be done using the <code>onload</code> event handler, which is noted in the <code><body></code> tag, and is called via the appropriate function after loading the web page (in this case: <code>initCanvas()</code>).</p> <pre data-bbox="360 824 1040 981">if (moduleCanvasHelpers.isCanvasSupported()) { gauges = initGauges(); bars = initBars(); canvasControls = gauges.concat(bars); }</pre>
2.	<p>It is recommended that the initialization of the particular types of <code>Canvas</code> elements (i.e. round instruments (<code>Gauge</code>) as well as bars (<code>Bar</code>)) are again encapsulated in separate functions, and these can then be called in the <code>initCanvas()</code> function. Especially if several elements of the same type are used in an HTML page, these can be simply and quickly added in the corresponding function.</p> <pre data-bbox="360 1205 1276 1451">function initGauges() { var gaugeSpeed = new canvasControls.Gauge('gaugeSpeed', 0, 3000, 'Rotation speed', 'rpm'); gaugeSpeed.addDefaultColoredSections(0, 2400, 2700, 3000); gaugeSpeed.refresh(null, null); return [gaugeSpeed]; }</pre>
3.	<p>The variable <code>gaugeSpeed</code> is initialized using the <code>canvasControls.Gauge()</code> function as new <code>Canvas</code> element, type <code>Gauge</code>.</p> <p>The parameters in brackets have the following functions:</p> <ul style="list-style-type: none"> • gaugeSpeed parameter ID, which was defined in the parameter source, and assigned a <code>Canvas</code> element in the HTML page. • 0 and 3000 Minimum and maximum value of the scale of the <code>Canvas</code> element • Rotation speed Title of the <code>Canvas</code> element • rpm Unit in which the scale of the <code>Canvas</code> element is shown
4.	<p>Using the command <code>gaugeSpeed.addDefaultColoredSections(...)</code>, the scale of the <code>Canvas</code> element can be subdivided into colored sectors:</p>

No.	Action
	<ul style="list-style-type: none"> - Between the first and second value, which are noted in the brackets, the scale of the <code>Canvas</code> element is displayed in <u>green</u>. - Between the second and third value, the scale of the <code>Canvas</code> element is displayed in <u>yellow</u>. - Between the third and fourth value, the scale of the <code>Canvas</code> element is displayed in <u>red</u>.
5.	The command <code>gaugeSpeed.refresh(null, null)</code> only means that at the start (i.e. after calling the website), a pointer is not displayed at the <code>Canvas</code> element.
6.	<p>The function <code>initGauges()</code> returns, using the command <code>return[gaugeSpeed]</code> an array, which as index contains the variable <code>gaugeSpeed</code>, which previously was initialized as new <code>Canvas</code> element, type <code>Gauge</code>.</p> <p>If several <code>Canvas</code> elements of this type are to be initialized, then the code above can be copied, and again inserted in the function <code>initGauges()</code>. In this case, only the variable name of the new element, e.g. in <code>gaugeSpeed2</code>, has to be renamed, and the corresponding parameter ID inserted.</p> <p>The command <code>return[...]</code> then contains, as additional index, the variable <code>gaugeSpeed2</code>:</p> <pre>return[gaugeSpeed, gaugeSpeed2];</pre>

Note

```
function initBars() {
    var barTorque = new canvasControls.Bar('barTorque', 0,
    0.3, 'Torque', 'Nm');
    barTorque.addDefaultColoredSections(0, 0.1, 0.2, 0.3);
    barTorque.refresh(null);

    return [barTorque];
}
```

The function `initBars()` shows an example on how to initialize canvas elements, type `Bars`.

The only difference to the function `initGauges()` is that for the command `canvasControls`, the expression `.Bar` is used instead of `.Gauge`. Otherwise, the procedure is analogous to the steps described above.

No.	Action
7.	<p>The canvas elements can now be updated using the <code>initCanvas()</code> function.</p> <pre data-bbox="359 338 1332 499">/* Change to cyclic updating mode with a period of 3000ms */ setInterval(function () { updateDocument.updateValuesEx('variablesDiagAxis.mwsl?doName=' + doName, canvasControls, null); }, 3000);</pre> <p>In this case, the previously defined functions <code>initGauges()</code> and <code>initBars()</code> are called, and the particular return values (i.e. arrays) are transferred to variables <code>gauges</code> and <code>bars</code>.</p> <pre data-bbox="359 645 686 707">gauges = initGauges(); bars = initBars();</pre>
8.	<p>Using the JavaScript function <code>concat()</code>, both arrays are then combined to create one array, and saved in variable <code>canvasControls</code>. The variable <code>canvasControls</code> then contains all array elements that should be updated.</p> <pre data-bbox="359 860 906 887">canvasControls = gauges.concat(bars);</pre>
9.	<p>In turn, the elements are updated using the function <code>updateDocument.updateValuesEx(...)</code>, which – noted in the JavaScript function <code>setInterval()</code> – is cyclically called.</p> <p>The variable <code>canvasControls</code> is transferred to the function as second parameter, which means that the individual <code>Canvas</code> elements of the HTML page are supplied with new values.</p>

Note

If only type (`Gauge` or `Bar`) canvas elements are contained in an HTML page, then the variable, which is written to with the return array of the particular initialization function, can also be directly transferred to the `updateDocument.updateValuesEx()` function. The route via the JavaScript function `concat()` is then not necessary.

3.4 Loading the content of the web page

A second HTML page must be created in order that the web page can be subsequently displayed in the drive web server. This is necessary, as scripting would otherwise not be supported as a result of the inherent system. The page contains an `IFrame`, which is used to load the actual web page.

Note The individual steps are explained using the following example of the `diagAxis.mwsl` page.

Table 3-17

No.	Action
1.	<pre data-bbox="359 705 1364 896"><iframe src="USERFILES/WEBSITES/diagAxis.mwsl?doName= <MWSL><!--write(doNameAxis);--></MWSL>" id="contentOneAxis" class="fullContentWindowIframe"> &lt;p&gt;Unfortunately your browser is not able to display embedded IFrames&lt;/p&gt; </iframe></pre> <p data-bbox="359 907 1364 996">The path, from the perspective of the load page, under which the main page is saved, must be transferred to the attribute <code>src</code> in the opening <code><iframe></code> tag (in this case: <code>USERFILES/WEBSITES/diagAxisRed.mwsl</code>).</p> <p data-bbox="359 1008 1364 1097">The size of the display area is saved in the format <code>fullContentWindowIframe</code> in the stylesheet <code>pickAndPlaceMasterHeader.css</code> using the two attributes <code>width</code> and <code>height</code>, which is accessed using the <code>class</code> attribute.</p> <p data-bbox="359 1108 1364 1164">If the browser is not able to display <code>IFrames</code>, then a text can be noted between the <code><iframe></code> and <code></iframe></code> tags, which is then displayed to inform the user.</p>

Note You can also find additional information on `IFrames` under the following link:
https://www.w3schools.com/html/html_iframe.asp

3.5 Sample pages

3.5.1 diagAxis.html (main page 1)

Note

The structure and content of the sample page `diagAxisRed.html` is explained in more detail in the following section.

Please note that the content of the page-specific sources – the stylesheet `diagAxis.css` and the JavaScript library `diagAxis.js` – are not discussed in any detail here.

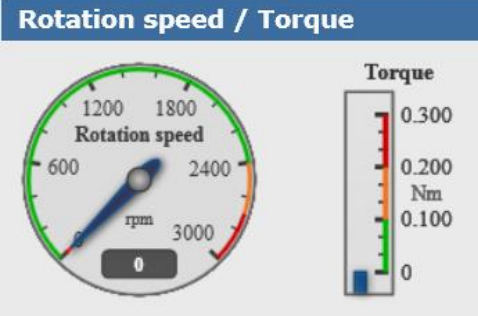
You can find basic information on the topics of "JavaScript" and "CSS" in the chapters 0: [JavaScript](#) as well as 2.4: [Cascading Style Sheets \(CSS\)](#).

Table 3-18

HTML code	Web page
<pre> <div class="pageColumn" id="indicatorsColumn"> <h1>Enables</h1> <div class="indicatorItem"> <p>Enables existing</p> </div>
 <div class="indicatorItem"> <p>OFF1 enable</p> </div> <div class="indicatorItem"> <p>OFF2 enable</p> </div> <div class="indicatorItem"> <p>OFF3 enable</p> </div> <div class="indicatorItem"> <p>Operation enabled</p> </div> <div class="indicatorItem"> <p>Ramp-function generator enable</p> </div> <div class="indicatorItem"> <p>Continue ramp-function generator</p> </div> <div class="indicatorItem"> <p>Speed setpoint enable</p> </div> </div> </pre>	 <p>The screenshot shows a web page titled "Enables" with a blue header. Below the header is a list of status indicators, each with a circular icon and a text label:</p> <ul style="list-style-type: none"> Enables existing (Red icon with white center) OFF1 enable (Red icon with white center) OFF2 enable (Green icon with white center) OFF3 enable (Green icon with white center) Operation enabled (Green icon with white center) Ramp-function generator enable (Green icon with white center) Continue ramp-function generator (Green icon with white center) Speed setpoint enable (Green icon with white center)

Using the above HTML code an area (`<div>`) is created in the `<body>` area of the page in which individual status displays can be inserted using additional `<div>` elements. Explanatory text is added to each status display using its `<p>` element. The status displays are formatted using the `class="indicatorItem"` attribute. The `indicatorItem` format is saved in the `pickAndPlaceMaster.css` stylesheet.

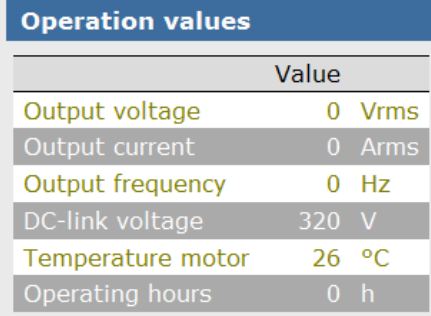
Table 3-19

HTML-Code	Web page
<pre> <div class="pageColumnRight" id="valuesColumn"> <h1>Rotation speed / Torque</h1> <div id="canvasControls"> <canvas class="canvasControl" id="gaugeSpeed" width="140" height="140">Canvas (HTML5) is not </canvas> <canvas class="canvasControl" id="barTorque" width="80" height="140">supported by your browser. </canvas> </div> ... </div> </pre>	

An additional area of the web page contains the two Canvas elements.

The height (height) and width (width) of each element is directly specified here; additional formatting information is saved in the `diagAxis.css` stylesheet; this can be accessed using the `class="canvasControl"` attribute.

Table 3-20

HTML-Code	Web page
<pre> <div class="pageColumnRight" id="valuesColumn"> ... <h1>Operation values</h1> <table class="valuesTable"> <colgroup> <col class="valuesTableColumn1" /> <col class="valuesTableColumnValues" /> <col class="valuesTableColumnUnits" /> </colgroup> <thead> <tr> <th></th> <th>Value</th> <th></th> </tr> </thead> <tbody> <tr class="valuesTableRowEven"> <td>Output voltage</td> <td id="valOutputVoltage" class="valuesTableValue"> --- 300 320 </td> <td>Vrms</td> </tr> ... </tbody> </table> </div> </pre>	

The web page also includes a table (<table>), which lists several important drive parameters.

Attributes `class="warningLimit"` and `class="criticalLimit"` are assigned to the parameters. If the parameter value falls below one of these limits, then the corresponding table cell has a colored background.

Note To improve the readability, in some instances, the complete HTML code is not shown, and line breaks inserted. Points mark the missing locations.

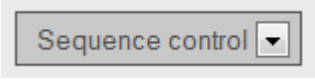
3.5.2 diagStatusAxis.html (main page 2)

Note The structure and content of the sample page `diagStatusAxis.html` is explained in more detail in the following section.

Please note that the content of the page-specific sources – the stylesheet `diagStatus.css` and the JavaScript library `diagStatusAxis.js` – are not discussed in any detail here.

You can find basic information on the topics of "JavaScript" and "CSS" in the chapters 0: [JavaScript](#) as well as 2.4: [Cascading Style Sheets \(CSS\)](#).

Table 3-21

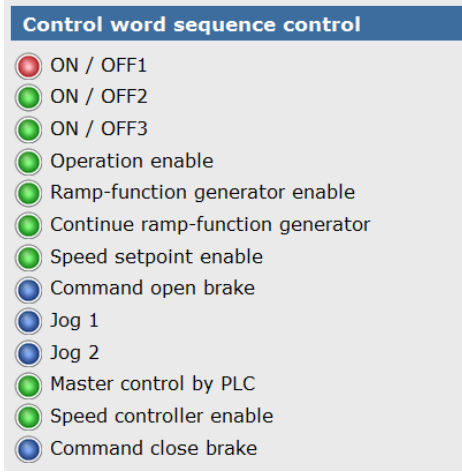
HTML-Code	Web page
<pre data-bbox="240 831 1027 1164"> <div id = 'selectContents'> <select class="customSelectBox" id="selectContentsToShow" size="1" onchange="diagStatusAxisHandler.setPageContents()"> <option value=0>Sequence control</option> <option value=1>Faults / Alarms 1</option> <option value=2>Faults / Alarms 2</option> <option value=3>Speed controller</option> </select> </div> </pre>	

Using the above HTML code, an area (`<div>`) is created in the `<body>` area of the page in which a selection list is inserted using the `<select>` tag.

The list includes a total of four selection options (`option`). However, using the attribute `size="1"`, only one option is directly displayed, which means that the list becomes a drop-down menu.

When changing the value, the `setPageContents()` JavaScript function is called each time. Using the `value` attributes, the individual options are assigned internal values; these are used to query the user's selection and dynamically adapt the content of the web page.

Table 3-22

HTML-Code	Webseite
<pre> <div class="pageColumn" id="controlWordColumn"> <h1>Control word sequence control</h1> <div class="indicatorItem"> <p>ON / OFF1</p> </div> <div class="indicatorItem"> <p>ON / OFF2</p> </div> <div class="indicatorItem"> <p>ON / OFF3</p> </div> ... </div> </pre>	

The other HTML code of the page comprises areas, which contain status displays for the following control and status words:

- Sequence control
- Faults/alarms 1
- Faults/alarms 2
- Speed controller

Depending on what the user has selected from the selection list, the relevant areas are displayed or hidden (control and status words).

3.5.3 axisRedLoader.html (load page)


Note

The structure and content of the sample page `axisRedLoader.html` is explained in more detail in the following section.

Please note that the content of the page-specific source – the stylesheet `pickAndPlaceMasterHeader.css` – is not discussed in any detail here.

You can find basic information on the topic of "CSS" in Chapter 2.4: [Cascading Style Sheet \(CSS\)](#).

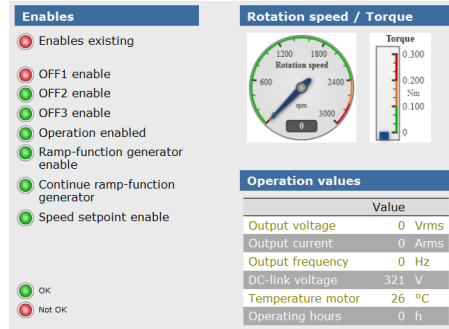
Table 3-23

HTML-Code
<pre> <div class="pageHeadings"> <h1>Pick and Place Master S120 (High-End)</h1> <div id="subHeadingBox"> <h2 id="headingOneAxis"><MWSL><!--write(titleAxis);--></MWSL> - Diagnostics</h2> <div id="headerNavigationTabs"> <input type="button" id="menuControlWords" class="navigationItemUnSelected" value="Status" onclick="document.getElementById('contentOneAxis').src = 'USERFILES/WEBSITES/diagStatusAxis.mwsl?doName=<MWSL><!--write(doNameAxis);--></MWSL>'; document.getElementById('headingOneAxis').innerHTML = '<MWSL><!--write(titleAxis);--></MWSL> - Status'; document.getElementById('menuDiagnostics').className = 'navigationItemUnSelected'; document.getElementById('menuControlWords').className = 'navigationItemSelected'; document.getElementById('menuControlPanel').className = 'navigationItemUnSelected';" /> <input type="button" id="menuDiagnostics" class="navigationItemSelected" value="Diagnostics" onclick="document.getElementById('contentOneAxis').src = 'USERFILES/WEBSITES/diagAxis.mwsl?doName=<MWSL><!--write(doNameAxis);--></MWSL>'; document.getElementById('headingOneAxis').innerHTML = '<MWSL><!--write(titleAxis);--></MWSL> - Diagnostics'; document.getElementById('menuDiagnostics').className = 'navigationItemSelected'; document.getElementById('menuControlWords').className = 'navigationItemUnSelected'; document.getElementById('menuControlPanel').className = 'navigationItemUnSelected';" /> </div> </div> </div> </pre>
Webseite


Using the above HTML code, an area (`<div>`) is created in the `<body>` area of the page which contains the header (i.e. the titles) of the various pages.

Using the two buttons (Diagnostics and Status) in the (`<div id="headerNavigationTabs">`) you can determine which content of the web page is to be displayed, i.e. the content of page `diagAxis.html` or `diagStatusAxis.html`.

Table 3-24

HTML-Code	Web page														
<pre> <iframe id="contentOneAxis" class="fullContentWindowIframe" src="USERFILES/WEBSITES/diagAxis.mwsl?doName= <MWSL><!--write(doNameAxis);--></MWSL>" &lt;p&gt;Unfortunately your browser is not able to display embedded IFrames&lt;/p&gt; </iframe> </pre>	 <p>Enables</p> <ul style="list-style-type: none"> <input type="radio"/> Enables existing <input type="radio"/> OFF1 enable <input checked="" type="radio"/> OFF2 enable <input checked="" type="radio"/> OFF3 enable <input checked="" type="radio"/> Operation enabled <input checked="" type="radio"/> Ramp-function generator enable <input checked="" type="radio"/> Continue ramp-function generator <input checked="" type="radio"/> Speed setpoint enable <p><input checked="" type="radio"/> OK <input type="radio"/> Not OK</p> <p>Rotation speed / Torque</p> <p>Rotation speed: 0 rpm Torque: 0 Nm</p> <p>Operation values</p> <table border="1"> <thead> <tr> <th></th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>Output voltage</td> <td>0 Vrms</td> </tr> <tr> <td>Output current</td> <td>0 Arms</td> </tr> <tr> <td>Output frequency</td> <td>0 Hz</td> </tr> <tr> <td>DC-link voltage</td> <td>321 V</td> </tr> <tr> <td>Temperature motor</td> <td>26 °C</td> </tr> <tr> <td>Operating hours</td> <td>0 h</td> </tr> </tbody> </table>		Value	Output voltage	0 Vrms	Output current	0 Arms	Output frequency	0 Hz	DC-link voltage	321 V	Temperature motor	26 °C	Operating hours	0 h
	Value														
Output voltage	0 Vrms														
Output current	0 Arms														
Output frequency	0 Hz														
DC-link voltage	321 V														
Temperature motor	26 °C														
Operating hours	0 h														

The IFrame contains the actual content of the web page.

Depending on what the user has selected (Diagnostics or Status), the path of the required page is assigned to the attribute src of the IFrame, and therefore its content is displayed.

The size of the display area is saved in the format fullContentWindowIframe in the stylesheet pickAndPlaceMasterHeader.css, and is accessed using the class attribute.

3.6 Uploading files to the web server

Note

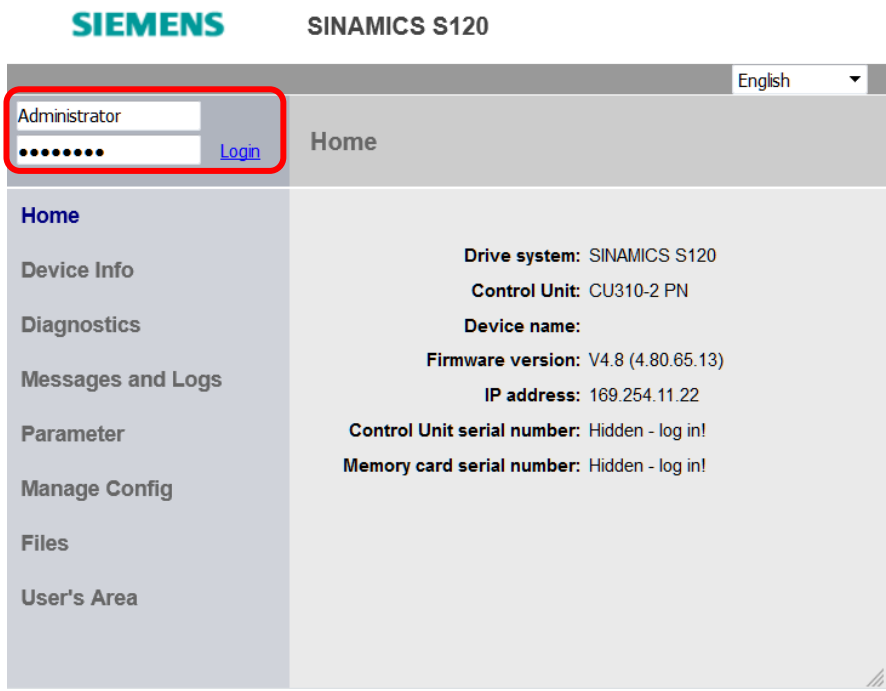
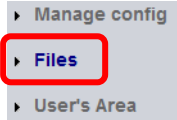
The following preconditions must be observed in order that web pages created can be uploaded to the drive web server:

- A functioning TCP/IP connection must exist between the PG/PC and the SINAMICS drive, via which the drive can be accessed.
- The basic commissioning of the drive must have been completed, i.e. all of the drive objects are available and ready for operation.
- During the basic commissioning, the user must have been set up as "**Administrator**" in STARTER. Only this user has the rights to upload user-defined pages into the web server.

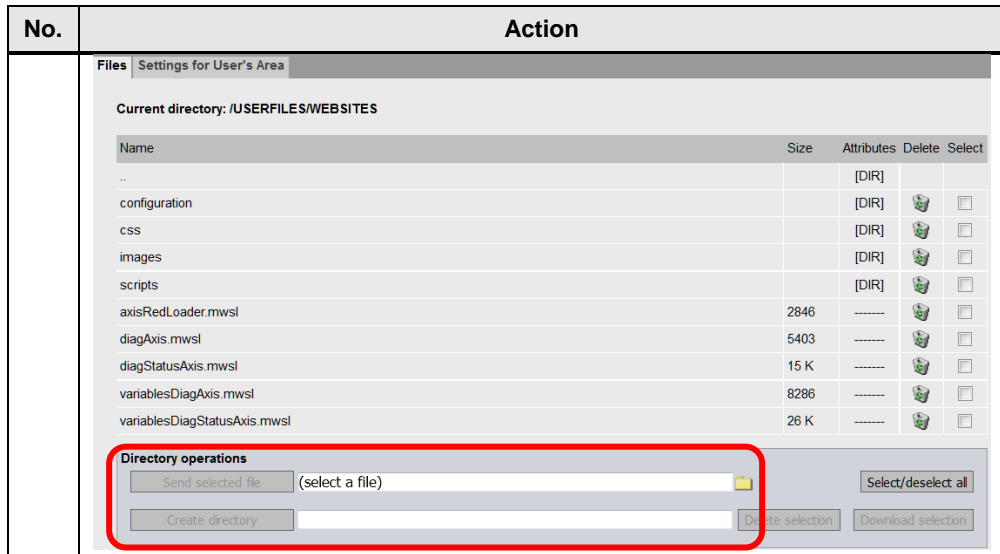
Also refer to the SINAMICS S120 Function Manual, Chapter 6.28.3: Configuring the web server.

<https://support.industry.siemens.com/cs/ww/en/view/109740020>

Table 3-25

No.	Action
1.	Call the web server of the SINAMICS drive by entering its IP address (e.g. the default IP address of the commissioning interface X127: 169.254.11.22) in the address line of your web browser, and confirm your input by pressing the enter key.
2.	<p>In the start page, enter the "Administrator" user name as well as the password you assigned to this user when commissioning the system and click on "Login".</p>  <p>The screenshot shows the SINAMICS S120 web interface. At the top, there is a header with the SIEMENS logo and 'SINAMICS S120'. Below the header, there is a navigation bar with 'Home' and a language dropdown set to 'English'. The main content area is divided into a left sidebar and a right main area. The sidebar contains links for Home, Device Info, Diagnostics, Messages and Logs, Parameter, Manage Config, Files, and User's Area. The main area displays system information: Drive system: SINAMICS S120, Control Unit: CU310-2 PN, Device name, Firmware version: V4.8 (4.80.65.13), IP address: 169.254.11.22, Control Unit serial number: Hidden - log in!, and Memory card serial number: Hidden - log in!. The login form is located at the top left of the main area, with the 'Administrator' username and a password field (masked with dots) highlighted by a red box. A 'Login' button is next to the password field.</p>
3.	<p>After you have successfully logged on, change to the "Files" menu.</p>  <p>The screenshot shows a sidebar menu with three items: 'Manage config', 'Files', and 'User's Area'. The 'Files' item is highlighted with a red box.</p>

No.	Action
4.	<p>New folders/directories can be created using the "Create Directory" button. To do this, enter the appropriate folder name in the text box, which is located to the right of the button.</p> <p>Create the folder "WEBSITES" first. In this folder all user-defined pages will be saved.</p> <p>To use the example pages create following subfolders in the folder "WEBSITES":</p> <ul style="list-style-type: none"> • css • images • scripts <p>A single file can be uploaded using the "Send selected file" button. To do this, select the appropriate file using the folder symbol, located to the right of the button.</p> <p>Upload all of the files that are required to display the sample pages. To do this, extract the zip archive supplied 68691599_s120_Userdefined_Webpages_V2_0.zip.</p> <p>Ensure that all of the files are saved in the following (folder) structure in the web server!</p> <ul style="list-style-type: none"> ▪ WEBSITES <ul style="list-style-type: none"> • css <ul style="list-style-type: none"> - diagAxis.css - diagStatus.css - pickAndPlaceMaster.css - pickAndPlaceMasterHeader.css • configuration <ul style="list-style-type: none"> - axis1.doName - axis1.title • images <ul style="list-style-type: none"> - indicatorCritical.png - indicatorOff.png - indicatorOn.png - indicatorNeutral.png • scripts <ul style="list-style-type: none"> - diagAxis.js - diagStatusAxis.js - libByMichael.js • axisRedLoader.mwsl • diagAxis.mwsl • diagStatusAxis.mwsl • variablesDiagAxis.mwsl • variablesDiagStatusAxis.mwsl



Note

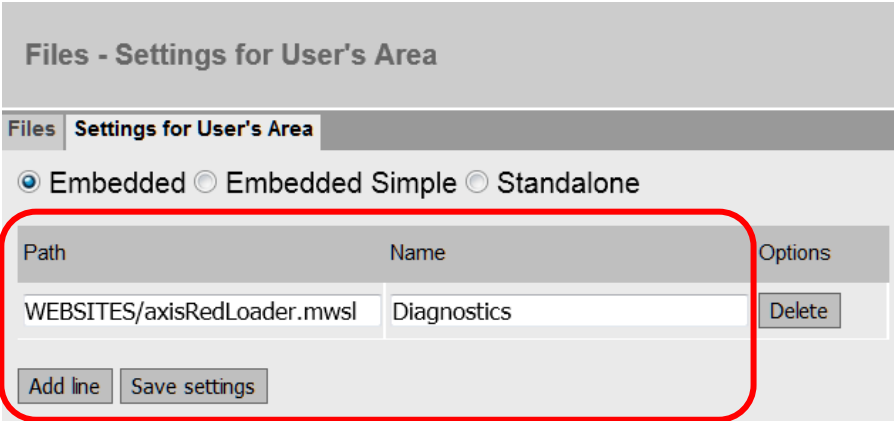

Alternatively, all of the files can be directly loaded to the CF card of the SINAMICS drive using a CF card reader.

To do this, extract the zip archive supplied
68691599_S120_Userdefined_Webpages_V2_0.zip.

Then copy the files of the MBS folder – including all of the subdirectories – into the target folder "OEM/SINAMICS/HMI/USERFILES" on the CF card.

© Siemens AG 2018. All rights reserved.

No.	Action
5.	<p>Then change to the "Files" menu and switch to the "Settings for User's Area" tab.</p>
6.	<p>Here you can select which web pages are to be displayed in the web server and how this is to be done:</p> <ul style="list-style-type: none"> Embedded This setting should be selected, if more than one web page is to be displayed in the web server. The web pages are listed in the "User's Area" menu and can be individually called as part of the web server. Embedded simple This setting can only be selected if only one web page is to be displayed

No.	Action
	<p>in the web server. The web page is then directly displayed as part of the web server, as soon as the user changes into the "User's Area" menu.</p> <ul style="list-style-type: none"> Standalone This setting can also only be selected if only one web page is to be displayed in the web server. The web page is then displayed as independent page (i.e. not as part of the web server), as soon as the user changes into the "User's Area" menu. <p>The sample pages are implemented as "embedded". Using the "Add row" button you can insert a new row here. Here, enter the path of the required web page (i.e. the file name of the load page) as well as a name via which the web page will be subsequently called and displayed in the "User's Area" menu. Then save your settings using the "Save settings" button.</p> 
7.	<p>Then change into the "User's Area" menu.</p>  <p>Depending on which display settings you have selected (embedded, embedded simple, standalone), the web pages are either directly displayed, or can be called using the previously assigned name.</p> <p>This is what the user area looks like if the pages are integrated as "embedded". For each row entry in the "Files > Settings" menu, a new tab is created in the "User's Area".</p>

3.7 Uploading new pages

Tabelle 3-26

No.	Action
1.	<p>Check all of the path data that was used in the individual sources. Paths must always be specified from the perspective of the source, just as they will be subsequently saved in the drive. Ensure that all source names in the path data, which refer to other pages that you create, have the .mwsI extension (with the exception of JavaScript files, CSS files and images)!</p> <p><u>Example:</u></p> <pre><iframe id="contentOneAxis" class="fullContentWindowIframe" src="USERFILES/WEBSITES/diagAxis.mwsI?doName= </MWSL><!--write(donameAxis)--></MWSL>"> <p>&lt;p&gt;Unfortunately your browser is not able to display embedded IFrames&lt;/p&gt; </iframe></pre>
2.	<p>Afterwards the files can be uploaded to the web server. Existing older files may be overwritten automatically.</p>

NOTE

Existing older files are only getting overwritten if the used control unit has at least following hardware version:

- CU310-2 DP / CU310-2 PN from HW-version E
- CU320-2 DP from HW-version G
- CU320-2 PN from HW-version D,

And the uploaded files have a newer creation date than the existing files and the CF card is 2GB in size.

No.	Action			
3.	<p>On first use the MWSL files will be compiled automatically. .mwsI → .mwsI.cms</p> <p><u>Example:</u></p> <table style="width: 100%; border: none;"> <tr> <td style="border: none;"> <ul style="list-style-type: none"> axisRedLoader.mwsI diagAxis.mwsI diagStatusAxis.mwsI variablesDiagAxis.mwsI variablesDiagStatusAxis.mwsI </td> <td style="border: none; text-align: center; vertical-align: middle; font-size: 2em;">→</td> <td style="border: none;"> <ul style="list-style-type: none"> axisRedLoader.mwsI.cms diagAxis.mwsI.cms diagStatusAxis.mwsI.cms variablesDiagAxis.mwsI.cms variablesDiagStatusAxis.mwsI.cms </td> </tr> </table>	<ul style="list-style-type: none"> axisRedLoader.mwsI diagAxis.mwsI diagStatusAxis.mwsI variablesDiagAxis.mwsI variablesDiagStatusAxis.mwsI 	→	<ul style="list-style-type: none"> axisRedLoader.mwsI.cms diagAxis.mwsI.cms diagStatusAxis.mwsI.cms variablesDiagAxis.mwsI.cms variablesDiagStatusAxis.mwsI.cms
<ul style="list-style-type: none"> axisRedLoader.mwsI diagAxis.mwsI diagStatusAxis.mwsI variablesDiagAxis.mwsI variablesDiagStatusAxis.mwsI 	→	<ul style="list-style-type: none"> axisRedLoader.mwsI.cms diagAxis.mwsI.cms diagStatusAxis.mwsI.cms variablesDiagAxis.mwsI.cms variablesDiagStatusAxis.mwsI.cms 		

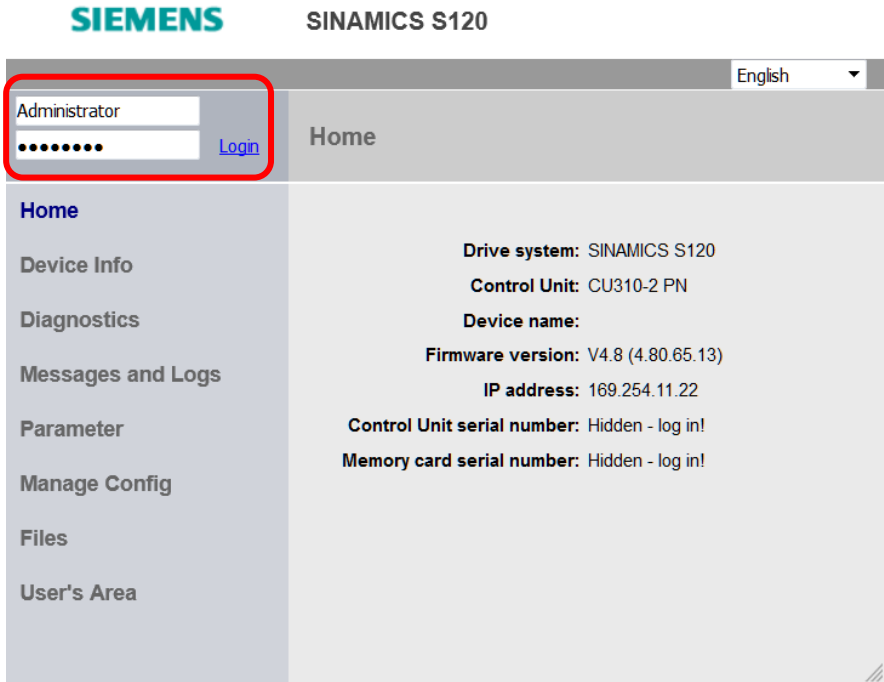
3.8 Loading and commissioning the sample pages

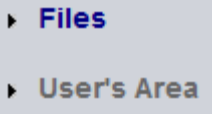
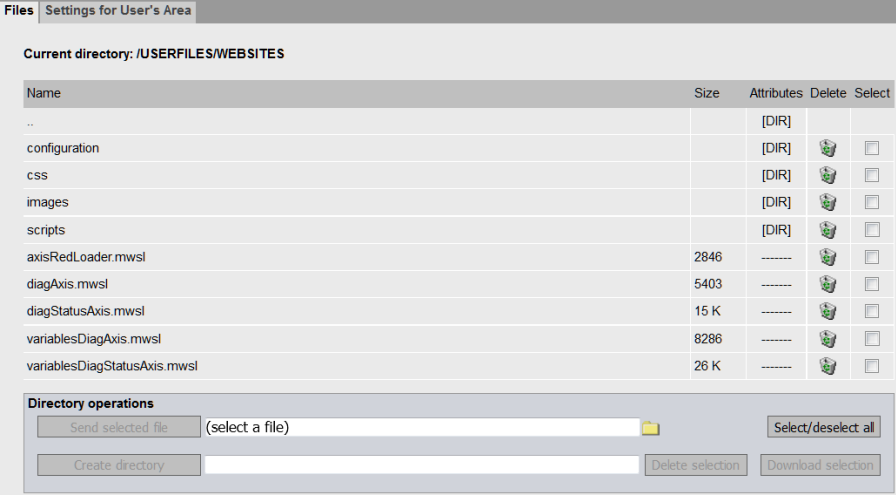
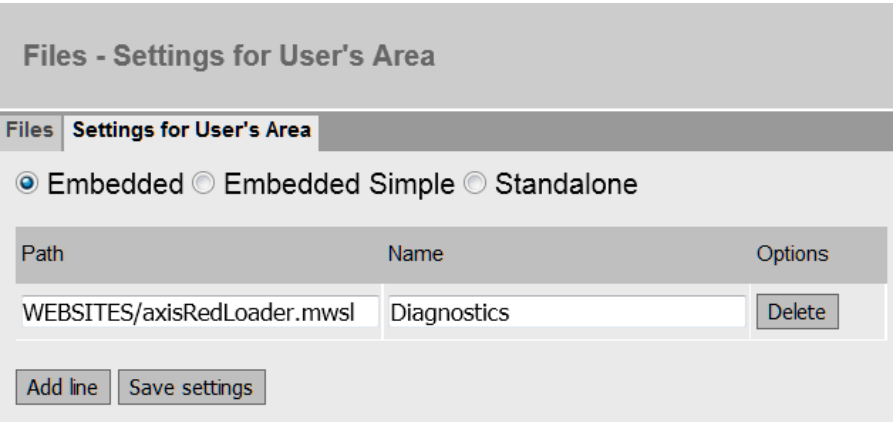
The startup of the completed sample pages of the zip archive 68691599_S120_Userdefined_Webpages_V2_0.zip, which was supplied with this application example, is explained in this chapter.

Chapter: Basic information as well as Chapter: Project planning and configuration do not have to be necessarily observed here, but they do help to understand how the web pages are created and to understand the structure of the configured sample pages.

3.8.1 ...via a CF card reader

Table 3-27

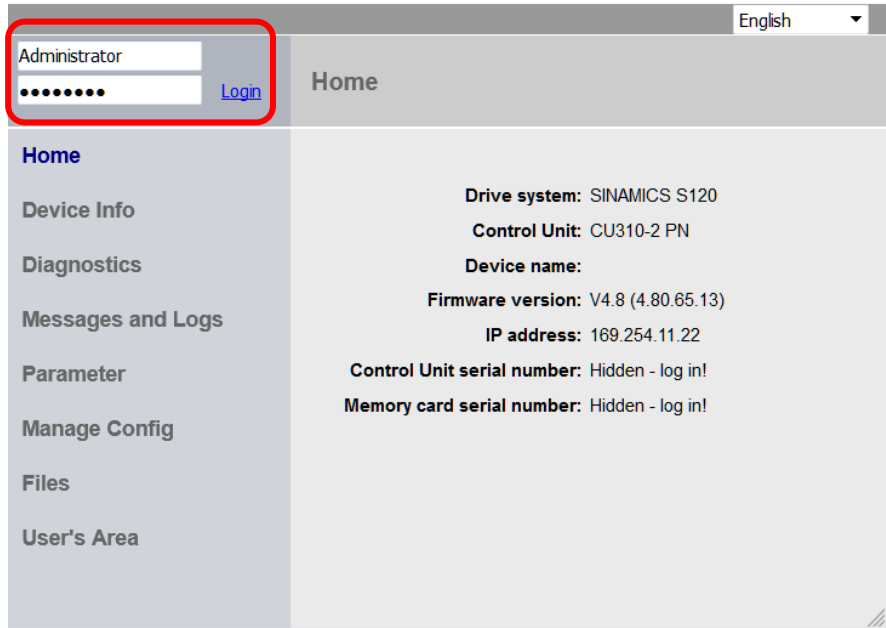
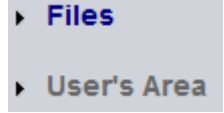
No.	Action
1.	Load all files from the zip-archive 68691599_S120_Userdefined_Webpages_V2_0.zip directly to the CF card of the SINAMICS drive using a CF card reader. Copy the folder "WEBSITES" including all files and subfolders to the target folder " OEM/SINAMICS/HMI/USERFILES " on the CF card.
2.	Turn on the SINAMICS drive with the inserted CF card.
3.	Call the web server of the SINAMICS drive by entering its IP address (e.g. the default IP address of the commissioning interface X127: 169.254.11.22) in the address line of your web browser, and confirm your input by pressing the Enter key.
4.	<p>In the start page, enter the "Administrator" user name as well as the password you assigned to this user when commissioning the system and click on "Login".</p>  <p>Note The "Administrator" user must have been enabled on carrying out the basic commissioning of the drive. The basic commissioning of the drive must have been completed.</p>

No.	Action
5.	After successful login change to the "Files" menu. 
6.	Your folder "WEBSITES" should then look like this: 
7.	Change to the "Files" menu under the "Settings for User's Area" tab. Either select the "Embedded" or "Embedded simple" setting, and insert a new line using the "Add line" button: Path: WEBSITES/axisRedLoader.mwsl Name: Diagnostics  Save your settings using the "Save settings" button.
8.	The application can now be used.

NOTE The correct operation of the loaded page can be checked as per **chapter 3.9: Operation**.

3.8.2 ...via the webserver

Table 3-28

No.	Action
1.	Call the web server of the SINAMICS drive by entering its IP address (e.g. the default IP address of the commissioning interface X127: 169.254.11.22) in the address line of your web browser, and confirm your input by pressing the Enter key.
2.	<p>In the start page, enter the "Administrator" user name as well as the password you assigned to this user when commissioning the system and click on "Login".</p>  <p>Note The "Administrator" user must have been enabled on carrying out the basic commissioning of the drive. The basic commissioning of the drive must have been completed.</p>
3.	<p>After successful login change to the "Files" menu.</p> 

4. New folders can be created by pressing the **"Create Directory"** button. Assign the appropriate folder names in the entry field, which is located to the right of the button.

Create the folder "WEBSITES" first. In this folder all userdefined webpages are saved.

Create the following subfolders in the folder "WEBSITES" to use the sample pages:

- configuration
- css
- images
- scripts

A single file can be uploaded using the **"Send selected file"** button. To do this, select the appropriate file using the folder symbol, located to the right of the button.

Files Settings for User's Area

Current directory: /USERFILES/WEBSITES

Name	Size	Attributes	Delete	Select
..		[DIR]		
configuration		[DIR]		<input type="checkbox"/>
css		[DIR]		<input type="checkbox"/>
images		[DIR]		<input type="checkbox"/>
scripts		[DIR]		<input type="checkbox"/>
axisRedLoader.mwsl	2846	-----		<input type="checkbox"/>
diagAxis.mwsl	5403	-----		<input type="checkbox"/>
diagStatusAxis.mwsl	15 K	-----		<input type="checkbox"/>
variablesDiagAxis.mwsl	8286	-----		<input type="checkbox"/>
variablesDiagStatusAxis.mwsl	26 K	-----		<input type="checkbox"/>

Directory operations

Send selected file (select a file) Select/deselect all

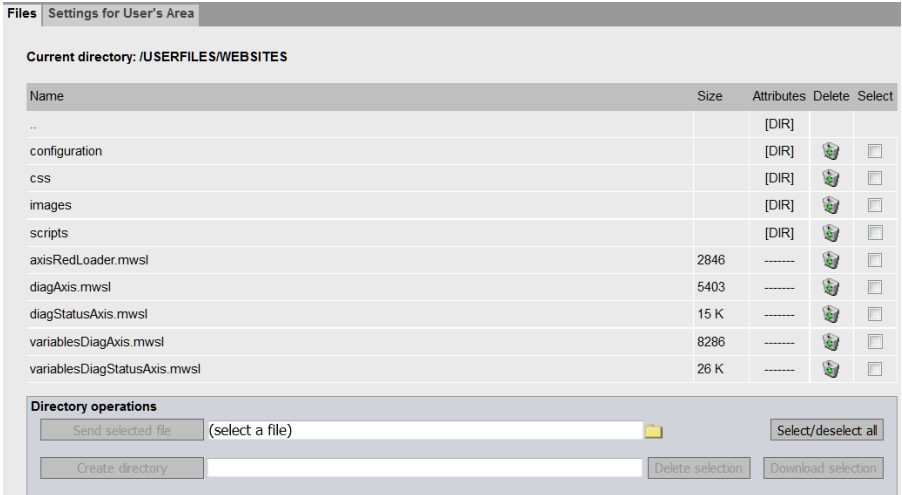
Create directory Delete selection Download selection

5. Upload all of the files that are required to display the required sample pages. To do this, extract the ZIP archive 68691599_S120_Userdefined_Webpages_V2_0.zip supplied. The files required for the sample pages are located in the folder "WEBSITES".

Ensure that all files of the sample pages are saved in the web server in the following (folder) structure in the web server of your drive:

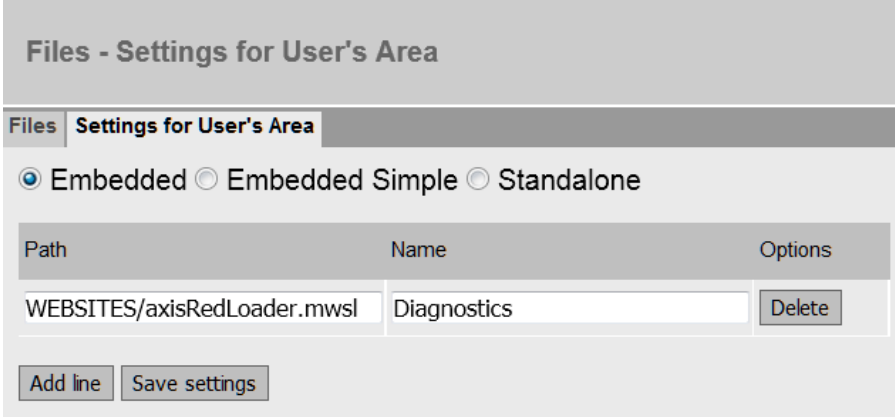
- **WEBSITES**
 - configuration
 - axis1.doName
 - axis1.title
 - css
 - diagAxis.css
 - diagStatus.css
 - pickAndPlaceMaster.css
 - pickAndPlaceMasterHeader.css
 - images
 - indicatorCritical.png
 - indicatorOff.png
 - indicatorOn.png
 - indicatorNeutral.png
 - indicatorWarning.png
 - scripts
 - diagAxis.js
 - diagStatusAxis.js
 - libByMichael.js
 - axisRedLoader.mwsl
 - diagAxis.mwsl
 - diagStatusAxis.mwsl
 - variablesDiagAxis.mwsl
 - variablesDiagStatusAxis.mwsl

6. Your folder "WEBSITES" should then look like this:



Name	Size	Attributes	Delete	Select
..		[DIR]		
configuration		[DIR]		<input type="checkbox"/>
css		[DIR]		<input type="checkbox"/>
images		[DIR]		<input type="checkbox"/>
scripts		[DIR]		<input type="checkbox"/>
axisRedLoader.mwsl	2846	-----		<input type="checkbox"/>
diagAxis.mwsl	5403	-----		<input type="checkbox"/>
diagStatusAxis.mwsl	15 K	-----		<input type="checkbox"/>
variablesDiagAxis.mwsl	8286	-----		<input type="checkbox"/>
variablesDiagStatusAxis.mwsl	26 K	-----		<input type="checkbox"/>

Directory operations: Send selected file (select a file) [Select/deselect all] Create directory [Delete selection] [Download selection]

7. Change to the **"Files"** menu under the **"Settings for User's Area"** tab. Either select the "Embedded" or "Embedded simple" setting, and insert a new line using the **"Add line"** button:
Path: WEBSITES/axisRedLoader.mwsl
Name: Diagnostics
- 
- Save your settings using the **"Save settings"** button.
8. The application can now be used.

NOTE

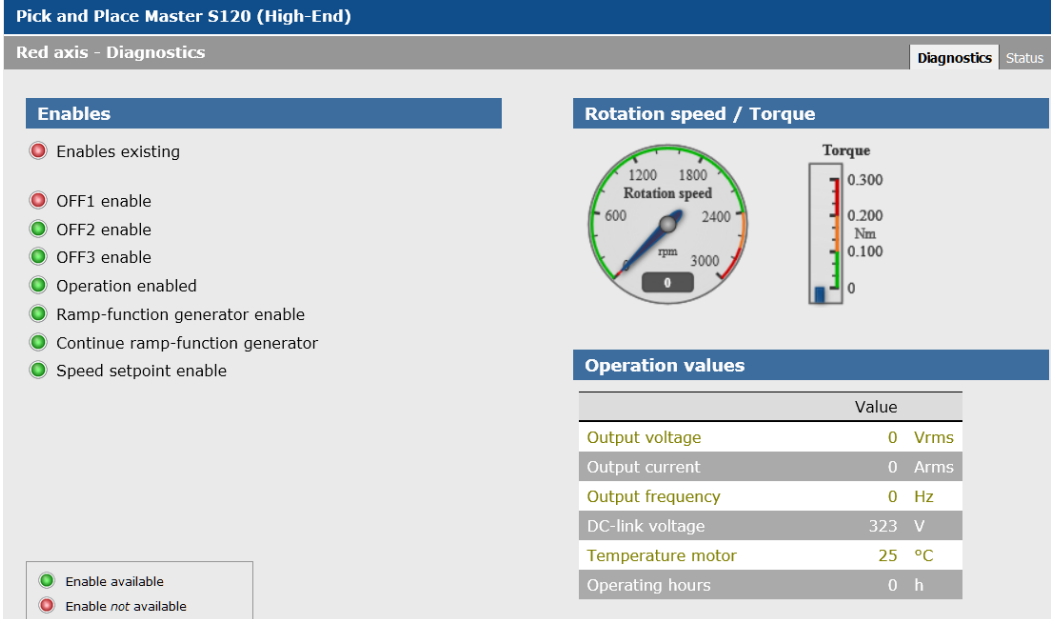
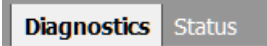
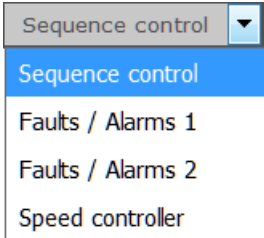
The correct operation of the loaded page can be checked as per **chapter 3.9: Operation**.

3.9 Operation

Table 3-29

Nr.	Aktion														
1.	<p>After you have successfully commissioned the application, go to the web server of the SINAMICS drive in the "User's Area" menu.</p> <ul style="list-style-type: none"> ▶ Files ▶ User's Area 														
2.	<p>The user-defined web page is divided into two parts:</p> <ul style="list-style-type: none"> • Diagnostics page with important parameters, as well as enables and speed/torque of a drive axis <div data-bbox="304 658 1291 1240" style="border: 1px solid gray; padding: 5px;"> <p>Pick and Place Master S120 (High-End)</p> <p>Red axis - Diagnostics Diagnostics Status</p> <div style="display: flex; justify-content: space-between;"> <div style="width: 45%;"> <p>Enables</p> <ul style="list-style-type: none"> <input type="radio"/> Enables existing <input type="radio"/> OFF1 enable <input type="radio"/> OFF2 enable <input type="radio"/> OFF3 enable <input type="radio"/> Operation enabled <input type="radio"/> Ramp-function generator enable <input type="radio"/> Continue ramp-function generator <input type="radio"/> Speed setpoint enable <div style="border: 1px solid gray; padding: 2px; margin-top: 10px;"> <ul style="list-style-type: none"> <input type="radio"/> Enable available <input type="radio"/> Enable not available </div> </div> <div style="width: 45%;"> <p>Rotation speed / Torque</p> <div style="display: flex; justify-content: space-around;"> <div style="text-align: center;">  <p>Rotation speed</p> </div> <div style="text-align: center;">  <p>Torque</p> </div> </div> <p>Operation values</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th></th> <th style="text-align: right;">Value</th> </tr> </thead> <tbody> <tr> <td>Output voltage</td> <td style="text-align: right;">0 Vrms</td> </tr> <tr> <td>Output current</td> <td style="text-align: right;">0 Arms</td> </tr> <tr> <td>Output frequency</td> <td style="text-align: right;">0 Hz</td> </tr> <tr> <td>DC-link voltage</td> <td style="text-align: right;">323 V</td> </tr> <tr> <td>Temperature motor</td> <td style="text-align: right;">25 °C</td> </tr> <tr> <td>Operating hours</td> <td style="text-align: right;">0 h</td> </tr> </tbody> </table> </div> </div> </div> <ul style="list-style-type: none"> • Status page for control and status words of a drive axis <div data-bbox="304 1285 1291 1924" style="border: 1px solid gray; padding: 5px;"> <p>Pick and Place Master S120 (High-End)</p> <p>Red axis - Status Diagnostics Status</p> <p>Sequence control ▼</p> <div style="display: flex; justify-content: space-between;"> <div style="width: 45%;"> <p>Control word sequence control</p> <ul style="list-style-type: none"> <input type="radio"/> ON / OFF1 <input type="radio"/> ON / OFF2 <input type="radio"/> ON / OFF3 <input type="radio"/> Operation enable <input type="radio"/> Ramp-function generator enable <input type="radio"/> Continue ramp-function generator <input type="radio"/> Speed setpoint enable <input type="radio"/> Command open brake <input type="radio"/> Jog 1 <input type="radio"/> Jog 2 <input type="radio"/> Master control by PLC <input type="radio"/> Speed controller enable <input type="radio"/> Command close brake <div style="border: 1px solid gray; padding: 2px; margin-top: 10px;"> <ul style="list-style-type: none"> <input type="radio"/> OK <input type="radio"/> Not OK <input type="radio"/> May be not important </div> </div> <div style="width: 45%;"> <p>Status word sequence control</p> <ul style="list-style-type: none"> <input type="radio"/> Ready for switch on <input type="radio"/> Ready <input type="radio"/> Operation enabled <input type="radio"/> Jog active <input type="radio"/> No coasting active <input type="radio"/> No quick stop active <input type="radio"/> Switching on inhibited active <input type="radio"/> Drive ready <input type="radio"/> Controller enable <input type="radio"/> Control request <input type="radio"/> Pulses enable <input type="radio"/> Open holding brake <input type="radio"/> Command close holding brake <input type="radio"/> Pulse enable from the brake control <input type="radio"/> Setpoint enable from the brake control </div> </div> </div>		Value	Output voltage	0 Vrms	Output current	0 Arms	Output frequency	0 Hz	DC-link voltage	323 V	Temperature motor	25 °C	Operating hours	0 h
	Value														
Output voltage	0 Vrms														
Output current	0 Arms														
Output frequency	0 Hz														
DC-link voltage	323 V														
Temperature motor	25 °C														
Operating hours	0 h														

© Siemens AG 2018. All rights reserved.

Nr.	Aktion														
3.	<p>The diagnostics page is always displayed, if the user-defined web page is called using the "Diagnostics" entry.</p> <p>Set enables are shown in green, while missing enables are shown in red.</p> <p>The actual speed as well as the torque of the axis are visualized using the two canvas elements in the form of pointers or bars.</p> <p>If a parameter of the parameter table exceeds a limit value defined in the HTML source, then its value has a colored background (warning: yellow, critical value: red).</p> <p>The corresponding data (i.e. parameter values) are then read out of the drive in a 3 second time grid.</p>  <table border="1" data-bbox="882 981 1270 1189"> <thead> <tr> <th></th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>Output voltage</td> <td>0 Vrms</td> </tr> <tr> <td>Output current</td> <td>0 Arms</td> </tr> <tr> <td>Output frequency</td> <td>0 Hz</td> </tr> <tr> <td>DC-link voltage</td> <td>323 V</td> </tr> <tr> <td>Temperature motor</td> <td>25 °C</td> </tr> <tr> <td>Operating hours</td> <td>0 h</td> </tr> </tbody> </table>		Value	Output voltage	0 Vrms	Output current	0 Arms	Output frequency	0 Hz	DC-link voltage	323 V	Temperature motor	25 °C	Operating hours	0 h
	Value														
Output voltage	0 Vrms														
Output current	0 Arms														
Output frequency	0 Hz														
DC-link voltage	323 V														
Temperature motor	25 °C														
Operating hours	0 h														
4.	<p>You can toggle between the diagnostics page and the status page using the "Diagnostics" or "Control- / Status words" entries.</p> 														
5.	<p>The status page displays various control and status words of a drive axis. You can select one of the following views from the drop-down menu:</p>  <p>Control and status word...</p> <ul style="list-style-type: none"> • ...of the sequence control (sequence control) • ...faults/alarms 1 (Faults / Alarms 1) • ...faults/alarms 2 (Faults / Alarms 2) • ...of the speed controller (speed controller) 														
6.	<p>The most important bits of the control and/or status word, which have the value TRUE, are shown in green on the status page. Important bits with the value FALSE, are correspondingly shown in red.</p>														



Nr.	Aktion		
	<p>Bits of the particular control and status word that are not so important are shown in blue, i.e. are shown in a neutral form (see also indicatorVariables). The corresponding data (i.e. parameter values) are then read out of the drive in a 3 second time grid.</p> <div style="border: 1px solid black; padding: 5px;"> <p>Pick and Place Master S120 (High-End)</p> <p>Red axis - Status Diagnostics Status</p> <p>Sequence control ▾</p> <table border="0" style="width: 100%;"> <tr> <td style="width: 50%; vertical-align: top;"> <p>Control word sequence control</p> <ul style="list-style-type: none"> ● ON / OFF1 ● ON / OFF2 ● ON / OFF3 ● Operation enable ● Ramp-function generator enable ● Continue ramp-function generator ● Speed setpoint enable ● Command open brake ● Jog 1 ● Jog 2 ● Master control by PLC ● Speed controller enable ● Command close brake </td> <td style="width: 50%; vertical-align: top;"> <p>Status word sequence control</p> <ul style="list-style-type: none"> ● Ready for switch on ● Ready ● Operation enabled ● Jog active ● No coasting active ● No quick stop active ● Switching on inhibited active ● Drive ready ● Controller enable ● Control request ● Pulses enable ● Open holding brake ● Command close holding brake ● Pulse enable from the brake control ● Setpoint enable from the brake control </td> </tr> </table> <div style="margin-top: 10px; border: 1px solid gray; padding: 2px;"> <ul style="list-style-type: none"> ● OK ● Not OK ● May be not important </div> </div>	<p>Control word sequence control</p> <ul style="list-style-type: none"> ● ON / OFF1 ● ON / OFF2 ● ON / OFF3 ● Operation enable ● Ramp-function generator enable ● Continue ramp-function generator ● Speed setpoint enable ● Command open brake ● Jog 1 ● Jog 2 ● Master control by PLC ● Speed controller enable ● Command close brake 	<p>Status word sequence control</p> <ul style="list-style-type: none"> ● Ready for switch on ● Ready ● Operation enabled ● Jog active ● No coasting active ● No quick stop active ● Switching on inhibited active ● Drive ready ● Controller enable ● Control request ● Pulses enable ● Open holding brake ● Command close holding brake ● Pulse enable from the brake control ● Setpoint enable from the brake control
<p>Control word sequence control</p> <ul style="list-style-type: none"> ● ON / OFF1 ● ON / OFF2 ● ON / OFF3 ● Operation enable ● Ramp-function generator enable ● Continue ramp-function generator ● Speed setpoint enable ● Command open brake ● Jog 1 ● Jog 2 ● Master control by PLC ● Speed controller enable ● Command close brake 	<p>Status word sequence control</p> <ul style="list-style-type: none"> ● Ready for switch on ● Ready ● Operation enabled ● Jog active ● No coasting active ● No quick stop active ● Switching on inhibited active ● Drive ready ● Controller enable ● Control request ● Pulses enable ● Open holding brake ● Command close holding brake ● Pulse enable from the brake control ● Setpoint enable from the brake control 		

3.10 Changing the name of the axes

You can change the name and the displayed title of the axis under "USERFILES/WEBSITES/configuration".

Figure 3-4

Current directory: /USERFILES/WEBSITES/configuration

Name	Size	Attributes	Delete	Select
..		[DIR]		
axis1.doName	8	-----		<input type="checkbox"/>
axis1.title	8	-----		<input type="checkbox"/>

Here you find 2 files for the axis:

In the file with the ending ".doName" you can assign the actual name of the drive.

In this example:

"SERVO_02"

In the file with the ending ".title" you can assign the name of the axis that is displayed on the web pages.

In this example:

"Red axis"

4 Appendix

4.1 Service and Support

Industry Online Support

Do you have any questions or need assistance?

Siemens Industry Online Support offers round the clock access to our entire service and support know-how and portfolio.

The Industry Online Support is the central address for information about our products, solutions and services.

Product information, manuals, downloads, FAQs, application examples and videos – all information is accessible with just a few mouse clicks:

<https://support.industry.siemens.com/>

Technical Support

The Technical Support of Siemens Industry provides you fast and competent support regarding all technical queries with numerous tailor-made offers – ranging from basic support to individual support contracts. Please send queries to Technical Support via Web form:

<https://www.siemens.com/industry/supportrequest>

SITRAIN – Training for Industry

We support you with our globally available training courses for industry with practical experience, innovative learning methods and a concept that's tailored to the customer's specific needs.

For more information on our offered trainings and courses, as well as their locations and dates, refer to our web page:

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

Service offer

Our range of services includes the following:

- Plant data services
- Spare parts services
- Repair services
- On-site and maintenance services
- Retrofitting and modernization services
- Service programs and contracts

You can find detailed information on our range of services in the service catalog web page:

<https://support.industry.siemens.com/cs/sc>

Industry Online Support app

You will receive optimum support wherever you are with the "Siemens Industry Online Support" app. The app is available for Apple iOS, Android and Windows Phone:

<https://support.industry.siemens.com/cs/ww/en/sc/2067>

4.2 Application support

Siemens AG
 Digital Factory Division
 Factory Automation
 Production Machines
 DF FA PMA APC
 Frauenaucher Str. 80
 91056 Erlangen, Germany
 mailto: profinet.team.motioncontrol.i-dt@siemens.com

4.3 Links and Literature

Table 4-1

No.	Topic
\1\	Siemens Industry Online Support https://support.industry.siemens.com
\2\	Link to this entry page of this application example https://support.industry.siemens.com/cs/ww/en/view/68691599
\3\	SINAMICS S120 Function Manual https://support.automation.siemens.com/WW/view/en/68042590
\4\	SIMOTION IT Programming Manual https://support.automation.siemens.com/WW/view/en/61148084
\5\	w3schools https://www.w3schools.com/
\6\	Other user-defined pages https://support.automation.siemens.com/WW/view/en/78388880
\7\	Protection with Industrial Security https://support.industry.siemens.com/cs/ww/en/view/50203404

4.4 Change documentation

Table 4-2

Version	Date	Modifications
V1.0	06/2013	First version
V2.0	08/2017	Revision (new format .mws)
V2.1	08/2018	Added all MWSL functions