SIEMENS
Ingenuity for life

Creating User-Defined Web Pages for S7-1200/S7-1500

SIMATIC STEP 7 (TIA Portal V16)
S7-1200 / S7-1500

https://support.industry.siemens.com/cs/ww/en/view/68011496

Siemens Industry Online Support

Legal information

Table of Contents

1 Preface

Aim of the application example

This application example shows you how you can easily and quickly create your own user-defined web pages, or “user-defined pages”, for the S7-1200/1500.

Main contents of the application example

The following main points will be discussed in this application example:

- Fundamentals for user-defined pages.
- Creating user-defined web pages.
- Configuring the web server with user-defined web pages.

Advantages

- **Integrated Web server**
  The standard web pages for easy viewing of service and diagnostics information are activated with one click. Additionally, individually created, user-defined web pages can be generated.
- **Location-independent**
  The web page can be called up world-wide via a standard internet browser.
- **Application example**
  Universal use of the application example for all control systems.

Uses

- No additional hardware and software required.
- Access to the web server is possible over long distances and mobile communication devices, e.g. tablet PC, smartphone, etc.
- Reduced working hours through simple enabling of the web server.
- Time saving in planning and implementing your automation solution through simple adjustment of the application example on hand.

Note The application example and the web server should not and cannot replace an HMI system.

2 Automation Task

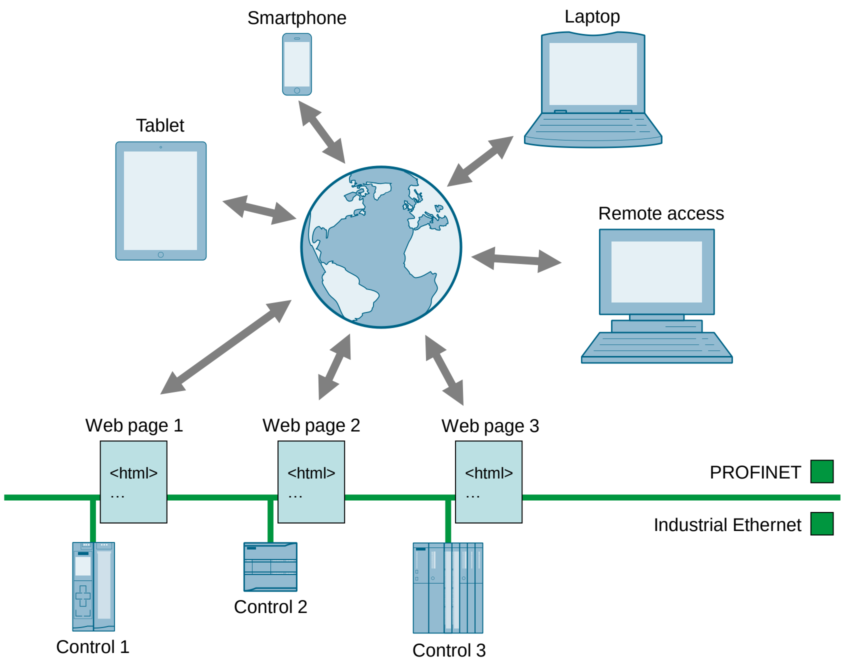
Overview of the automation task

Modern automation technology increasingly integrates internet technologies which together with integrated Ethernet-based communication enable, for example, direct access to the system via the intranet.

During the test and commissioning phase, the commissioning engineer wants to have flexible access to the CPU; individual data is to be visualized during operation for diagnostic purposes.

For access mechanisms via the internet or intranet, it is reasonable to use already existing standards, such as HTTP technology or standard web browsers, and common “languages” such as HTML (Hypertext Markup Language) or JavaScript.

Figure 2-1 Overview of the automation task



Requirements for the automation task

- Access the CPU with standard hardware and standard mechanisms via the Industrial Ethernet. Additional hardware and software are not required.
- Access the CPU that is individually related to the system and also visualized, if required. Each CPU has its own individual web page.
- Operating personnel without any automation knowledge are also provided simple access to the CPU.

3 Automation Solution

3.1 Overview of the Overall Solution

Diagram

SIMATIC CPUs with PROFINET interfaces provide the opportunity to access CPU variables with the help of web pages provided by the system.

Access the integrated web server of the CPU via a web browser.

The web server contains standard web pages, such as the Start Page, identification, diagnostic buffer, module status, messages, communication, topology, and file browser.

In addition to the standard web pages, you can also design and call the web pages individually for your application case.

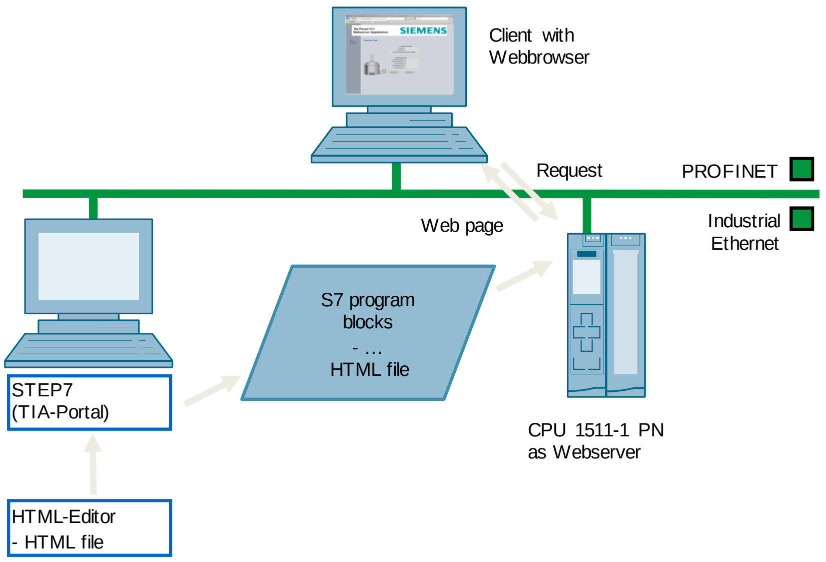
To create your user-defined web page, you can use tools such as Microsoft Expression Web, Notepad++, etc.

For designing your web page, you can use all options provided to you by HTML, CSS (Cascading Style Sheets), and JavaScript.

In addition, there are Automation Web Programming (AWP) commands for directed communication with the CPU.

The following Figure gives an overview of the implemented solution.

Figure 3-1 Overview of the overall solution



In this application example, the CPU simulates a tank which can be controlled through the web page.

The application example was implemented with three different solution possibilities (examples):

- Web page standard: HTML without JavaScript
- Web page optimized: HTML with JavaScript
- Web API (only S7-1500): HTML with Web API and JavaScript

3.2 Advantages and Possible Uses of Web Server Applications

By having access options through various web browsers, control data can be displayed, and controlled to a limited degree, by any computer or web-enabled devices without additional software installation.

Another advantage is being able to use the entire network infrastructure of a plant without any additional hardware components. This means that the corresponding controls can be accessed at any point in the plant where network access is available.

Evaluation, diagnostics, and controlling of the controllers can also be performed over large distances using mobile communication devices such as tablet PCs or smartphones. This also requires you, however, to take corresponding precautions to protect your system. Please observe our security notes in the Chapter "Warranty and Liability" in this context.

⚠ **WARNING**

No safety-relevant functions should be realized via the web server functionality due to the missing time deterministic of web applications!

3.3 Creating User-Defined Web Pages

The following steps describe the procedure for creating user-defined pages.

Figure 3-2 Procedure at a glance

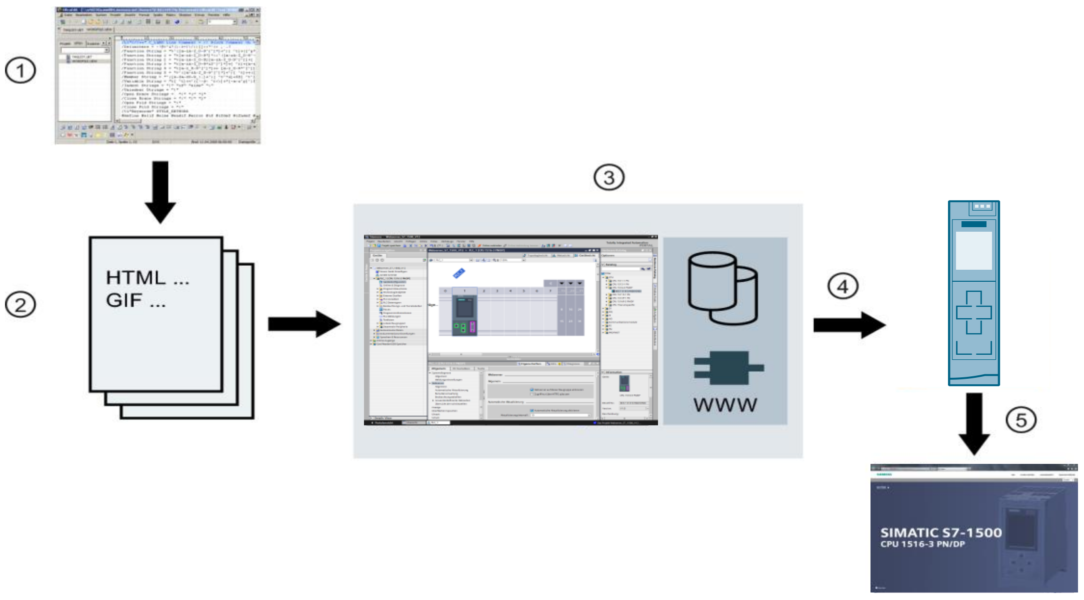


Table 3-1

No.	Command
1.	With an HTML editor, you can create the HTML file for the user-defined web page. (A standard editor such as Notepad/Notepad++ is sufficient for this.)
2.	The web application can be composed of different source files, e.g.: *.html, *.gif, *.js, etc.
3.	With SIMATIC STEP 7 (TIA Portal), the HTML files with images, etc. are stored in data blocks. Call the WWW command in the S7 program.
4.	Transfer all blocks onto the CPU.
5.	Open the web page of the CPU via a web browser. Accessing the web server of the CPU can be accomplished irrespective of the configuration computer. In other words, every output device with access to the PN interface of the CPU can display the web page.

You can find detailed explanations about creating a web page and programming in SIMATIC STEP 7 (TIA Portal) in Chapter 5.3.

Delimitation

This application example is an introduction to user-defined web pages for beginners. Simple methods are shown for accessing the web page of a CPU with HTML and SIMATIC STEP 7 (TIA Portal).

This application example does not include a complete description of HTML.

You can find more information about HTML and JavaScript in Chapter 9.2.

3.4 Structure of the Application Example

This application example was implemented with a CPU 1511-1 PN and a CPU 1214C DC/DC/DC. A PC is connected via the PROFINET interface. The PC serves for the creation of the S7 program and the HTML files, as well as for displaying the web pages in a web browser.

Shown are all steps necessary to create a web page and to subsequently call it via the CPU.

Content of the application example

You will find the following in-depth content in the application example:

- Configuration of the web server for a CPU with PN interface
- Creation of a user-defined web page for the CPU with the following functions:
  - Displaying CPU variables.
  - Graphic display of CPU variables.
  - Setting of CPU variables.
  - Displaying of texts which are linked with CPU variables.
  - Display of images that are linked to variables of the CPU.
  - Going to web pages with links in the navigation bar.
  - Cyclic update of the web pages with HTML.
  - Cyclic update of the web pages with JavaScript.
- Particularities in the S7 program creation:
  - Providing variables for the web page.
  - Further processing of variables from the web page in the S7 program.

3.5 Hardware and Software Components Used

The application example was created with the following components.

Hardware components

Note For this application example, the current firmware version of the CPU is required. Depending on the CPU type, the following entries contain links to the corresponding downloads:

- S7-1500: http://support.automation.siemens.com/WW/view/en/56926947/133200
- S7-1200: http://support.automation.siemens.com/WW/view/en/104546617/133200

Table 3-2

Components	Qty.	Item number	Note
CPU 1511-1 PN or CPU 1214C DC/DC/DC	1	6ES7511-1AK02-0AB0 6ES7214-1AG40-0XB0	FW 2.8 FW 4.4
Optional when using S7-1500: CP 1543-1	1	6GK7 543-1AX00-0XE0	PROFINET CP with firewall functionality for protection against unauthorized network access

Software components

Table 3-3

Components	Qty.	Item number	Note
SIMATIC STEP 7 Professional V16	1	6ES7822-1AA06-0YE5	-
Software tool for creating HTML files, e.g. Notepad++	1	-	-
Web browser, e.g. Google Chrome, Mozilla Firefox[1)]	1	-	The application example is optimized for Google Chrome.

[1)] The following web browsers were explicitly tested for communication with the CPU:

- Google Chrome (Version 80)
- Mozilla Firefox (Version 73.0)
- Internet Explorer (Version 11)

Note This application example is optimized for Google Chrome and Firefox.

If you are using Internet Explorer, **deactivate** “**Compatibility View**” in its settings (“Extras” menu).

If using other browsers, adjustments may have to be made.

Sample files and projects

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

Table 3-4

Components	Comments
68011496_S7-1200_1500_Webserver_CODE_v4.zip	The zip file contains the STEP 7 project with the corresponding HTML files. The HTML files with the associated files are located in the project directory under "\UserFiles\Webpages".
68011496_S7-1200_1500_Webserver_DOC_v4_de.pdf	This document.

4 Fundamentals of Standard Web Pages

Requirements

In SIMATIC STEP 7 (TIA Portal), the following settings are required in the properties of the CPU:

- The web server must be activated.
- If you require safe access to the standard web pages, enable the “Permit access only with HTTPS” check box.
  The Web API only supports the “HTTPS” transmission protocol.

Access via HTTP or HTTPS

With the URL “http://ww.xx.yy.zz” or “https://ww.xx.yy.zz”, you get access to the standard web pages. Here, “ww.xx.yy.zz” corresponds to the IP address of the SIMATIC S7-1200/S7-1500 CPU.

HTTPS is used to encrypt and authenticate the communication between browser and web server. If the check box “Only allow access via HTTPS” is checked, the web pages can only be accessed via HTTPS.

Log in

In the user list, a default user named “Everybody” has been created with administrative access pre-assigned. Registration on the controller’s web pages is therefore not necessary.

Note For more information on user handling and the integration of the login window into the user-defined pages, refer to “Web Server Examples for SIMATIC S7-1200/S7-1500”.

These examples can be downloaded from the same web page as this application example.

Standard web pages of SIMATIC S7-1500/S7-1200

The web server of SIMATIC S7-1500/S7-1200 already offers a lot of information about the respective CPU via integrated standard web pages.

You can find a detailed description of the setup of the standard web pages in the application description 59193560.

5 Function Mechanisms of this Application Example

Using a tank system as an example, three different methods are shown for exchanging data with the web server of the CPU and a web browser. The three options can be called up individually via the HTML file “Startpage.html”.

You can find the files for the three different options in the project directory under “Userfiles\Webpages” in the following folders:

- WebpageStandard: HTML without JavaScript
- WebpageOptimized: HTML with JavaScript
- Web API: HTML with Web API and JavaScript (only possible via HTTPS)

The individual examples each have a Start Page, an Overview Page, and a page for displaying measurement data. The three examples are loaded all at once onto the CPU. The examples are structured so that they may also be loaded onto the CPU individually.

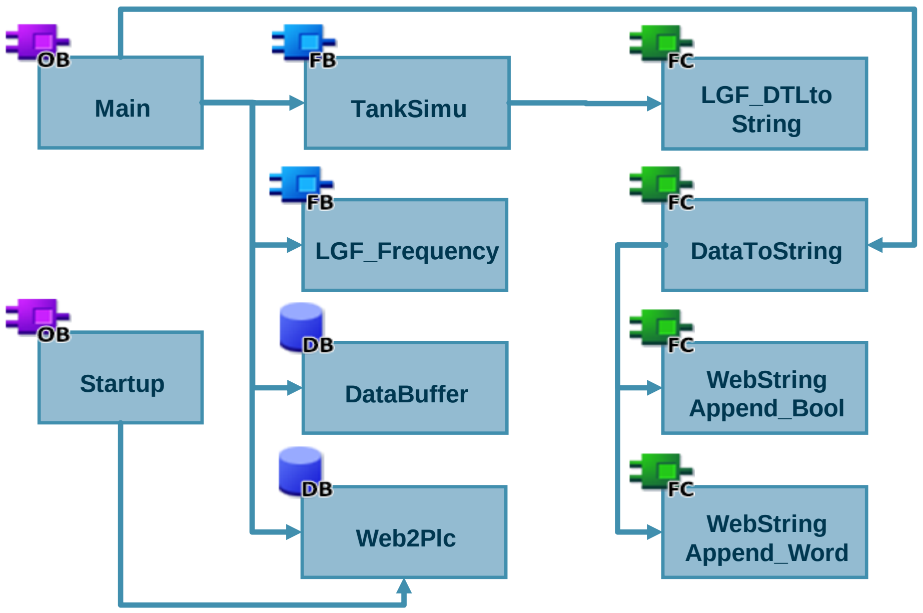
For the creation of the HTML pages, only fixed values are used for the position and size of the elements. This prevents the elements from moving and overlapping when the browser window is made smaller.

5.1 Functional Principle of the S7 Program

The S7 program of this application example only serves to illustrate individual functionalities of SIMATIC STEP 7 (TIA Portal).

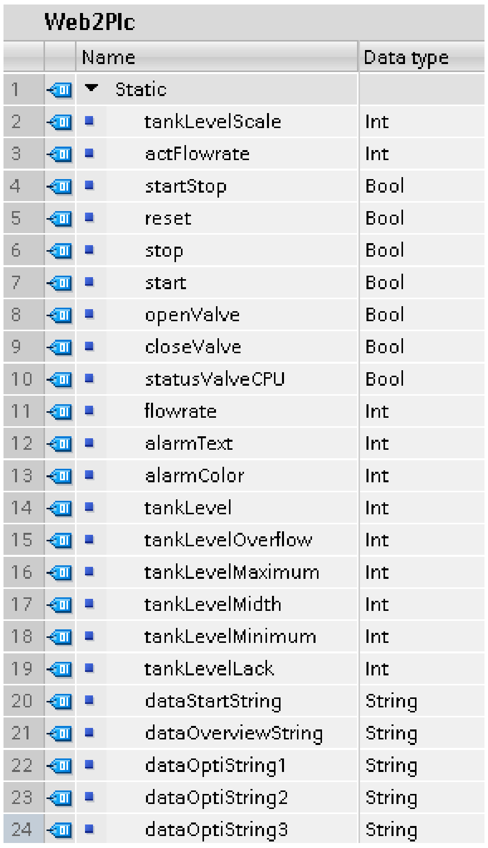
The following Figure shows the call structure in the S7 program.

Figure 5-1: S7 program call structure



The following variables are used in the data blocks “Web2Plc” and “DataBuffer”:

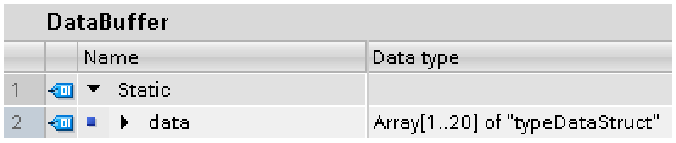
Figure 5-2: DB “Web2Plc”



Web2Plc

	Name	Data type
1	▼ Static	
2	tankLevelScale	Int
3	actFlowrate	Int
4	startStop	Bool
5	reset	Bool
6	stop	Bool
7	start	Bool
8	openValve	Bool
9	closeValve	Bool
10	statusValveCPU	Bool
11	flowrate	Int
12	alarmText	Int
13	alarmColor	Int
14	tankLevel	Int
15	tankLevelOverflow	Int
16	tankLevelMaximum	Int
17	tankLevelMidth	Int
18	tankLevelMinimum	Int
19	tankLevelLack	Int
20	dataStartString	String
21	dataOverviewString	String
22	dataOptiString1	String
23	dataOptiString2	String
24	dataOptiString3	String

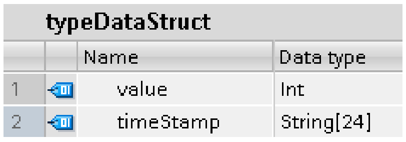
Figure 5-3: DB “DataBuffer”



DataBuffer

	Name	Data type
1	▼ Static	
2	▶ data	Array[1..20] of "typeDataStruct"

Figure 5-4: PLC data type “typeDataStruct”



typeDataStruct

	Name	Data type
1	value	Int
2	timeStamp	String[24]

5.1.1 Startup (OB100)

In OB “Startup” (OB100), a start value for the flow rate “Web2Plc.flowrate” and the limit values of the variables are stored.

5.1.2 Main (OB1)

In OB “Main” (OB1), the status of DB333 is polled cyclically to be able to recognize a request from the web browser. The request is made because a variable changed by the user is to be transferred from the web browser to the web server.

Synchronizing user-defined pages

The “WWW” (SFC99) command initializes the web server of the CPU. The error information is output via “RET_VAL”.

Calling the tank simulation

To ensure that filling or emptying of the tank does not happen too quickly, the “TankSimu” function block is called in OB1 only once per second.

Querying the “Start” or “Stop” and “Reset” buttons

The status of the “Start” and “Stop” buttons are polled by the web page. If one of the buttons is pressed, the state is stored in the PLC variable “Web2Plc.startStop”.

Additionally, the status of the “Reset” button is queried.

Querying the buttons for the “OpenValve” or “CloseValve” valve position

The status of the “OpenValve” (tank deflates) and “CloseValve” (tank closed) buttons is polled by the web page.

If one of the buttons is pressed, the state is stored in the PLC variable “Web2Plc.statusValveCPU”.

5.1.3 TankSimu (FB1)

How the FB “TankSimu” works

In the FB “TankSimu”, the filling or emptying of a tank is simulated, depending on the flow rate and the valve position.

The block is run through once per second.

On the web page, you determine the flow rate via the variable “Web2Plc.flowrate”. The tank level is increased or decreased with the flow rate when the FB “TankSimu” is called up. The current fill level is stored in the PLC variable “Web2Plc.tankLevel”.

The valve position is read and stored in the CPU in the PLC variable “Web2Plc.statusValveCPU” via the two PLC variables “Web2Plc.openValve” and “Web2Plc.closeValve”.

Dependent on the tank level, the following heights are displayed:

- Tank has been fully drained (TankLevelLack)
- Tank level is at minimum (TankLevelMinimum)
- Tank level is 50% (TankLevelMidth)
- Tank level is at maximum (TankLevelMaximum)
- Tank is overflowing (TankLevelOverflow)

With the variable “Web2Plc.alarmText”, the tank level is displayed in plain text. The variable “Web2Plc.alarmColor” is used to display the plain text in color. Enumerations or arrays (JavaScript) are used to display text and color.

StartStop status

The tank level is only changed and values are only entered into the ring buffer if the “Web2Plc.startStop” bit is set.

Valve status

Via the “Web2Plc.statusValveCPU” bit, the most recently pressed button (OpenValve or CloseValve) is saved.

Dependent on this bit, the tank is either emptied or filled.

Fill tank

The filling of the tank starts with a query to check if the tank is already full.

If the tank is not full, the tank level is increased with the flow rate. The tank level is limited by the “Web2Plc.tankLevelOverflow” value.

Empty tank

The emptying of the tank is similar to the filling of the tank. The tank level is reduced with the flow rate and is limited at 0.

AlarmText and AlarmColor status

Subsequently, the tank level is compared with the specifications for the limit values of the tank level.

Depending on which level is reached, the values “0” to “5” are stored in the variable “Web2Plc.alarmText”, and the values “0” to “3” are stored in the variable “Web2Plc.alarmColor”. The values of the two variables are stored in HTML text or colors as enumerations or arrays (JavaScript). This displays the level of the tank in colored, plain text.

5.1.4 DataToString (FC24)

Functionality of the FC "DataToString"

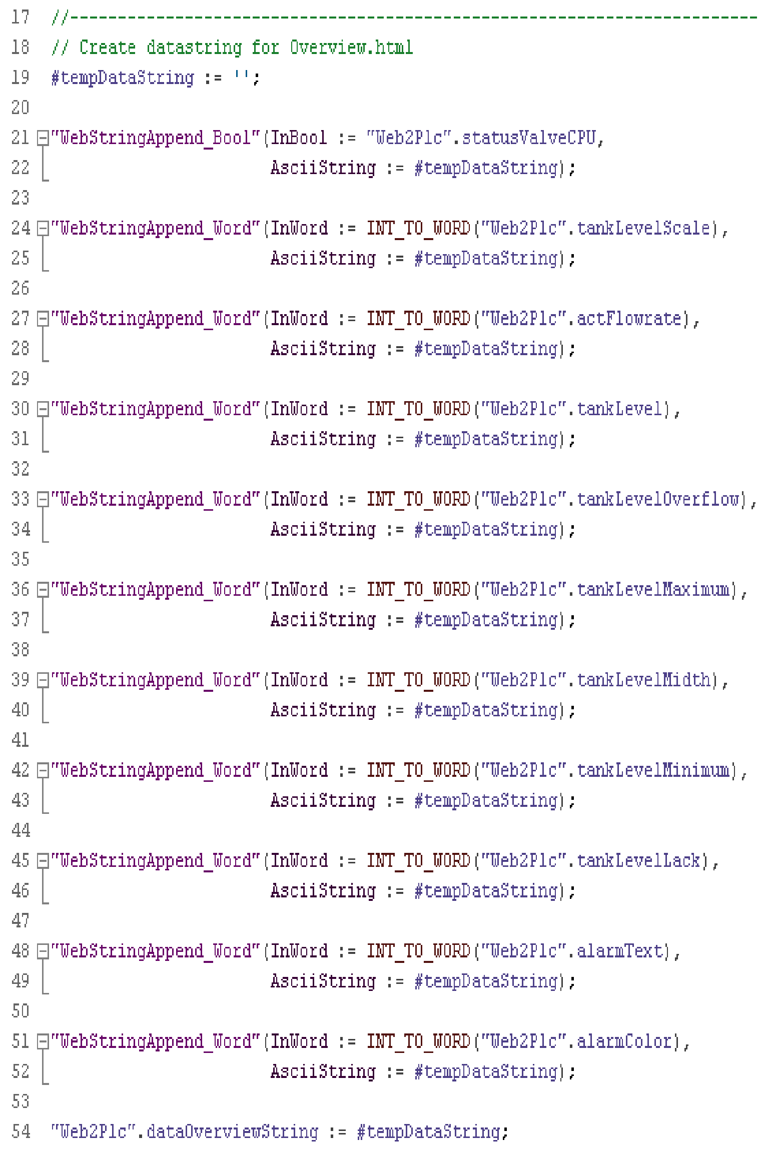
In the function, the values of several variables are combined into data strings using the functions "WebStringAppend_xxx". For each HTML file, one or, in the case of larger amounts of data, several data strings are generated. After reading the values in the web page, they are made available as an array via JavaScript (see Chapter 5.3).

Note It is important that the data string is assembled in a temporary variable and then copied to the variable responsible for the web page.

This ensures that the string is always completely assembled, since the web server accesses the program asynchronously.

The function is only needed for the "Web page optimized" example.

Figure 5-5: Generation of the data string for Overview.html



```
//----------------------------------------------------------------------
// Create datastring for Overview.html
#tempDataString := '';

"WebStringAppend_Bool"(InBool := "Web2Plc".statusValveCPU,
                       AsciiString := #tempDataString);

"WebStringAppend_Word"(InWord := INT_TO_WORD("Web2Plc".tankLevelScale),
                       AsciiString := #tempDataString);

"WebStringAppend_Word"(InWord := INT_TO_WORD("Web2Plc".actFlowrate),
                       AsciiString := #tempDataString);

"WebStringAppend_Word"(InWord := INT_TO_WORD("Web2Plc".tankLevel),
                       AsciiString := #tempDataString);

"WebStringAppend_Word"(InWord := INT_TO_WORD("Web2Plc".tankLevelOverflow),
                       AsciiString := #tempDataString);

"WebStringAppend_Word"(InWord := INT_TO_WORD("Web2Plc".tankLevelMaximum),
                       AsciiString := #tempDataString);

"WebStringAppend_Word"(InWord := INT_TO_WORD("Web2Plc".tankLevelMidth),
                       AsciiString := #tempDataString);

"WebStringAppend_Word"(InWord := INT_TO_WORD("Web2Plc".tankLevelMinimum),
                       AsciiString := #tempDataString);

"WebStringAppend_Word"(InWord := INT_TO_WORD("Web2Plc".tankLevelLack),
                       AsciiString := #tempDataString);

"WebStringAppend_Word"(InWord := INT_TO_WORD("Web2Plc".alarmText),
                       AsciiString := #tempDataString);

"WebStringAppend_Word"(InWord := INT_TO_WORD("Web2Plc".alarmColor),
                       AsciiString := #tempDataString);

"Web2Plc".dataOverviewString := #tempDataString;
```

5.2 Functionality of the HTML File Without JavaScript

The following chapter explains each section of the HTML files in the Web Page Standard folder.

5.2.1 AWP Commands

AWP commands are inserted as HTML comments in HTML files. AWP commands can be located at any position in the HTML file. However, for reasons of clarity, it is appropriate to list the central AWP commands at the beginning of the HTML file

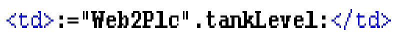
Table 5-1: AWB commands used

Function	Presentation
Reading PLC variables	`:=<Varname>:`
Writing PLC variables	`<!-- AWP_In_Variable Name='<Varname1>' -->`
Defining Enum types	`<!-- AWP_Enum_Def Name='<Name Enum-Typ>' Values='0:"<Text_1>",1:"<Text_2>",...,x:"<Text_y>"' -->`
Assigning Enum types to variables	`<!-- AWP_Enum_Ref Name='<Varname>' Enum='<Name Enum-Typ>' -->`

5.2.2 Outputting CPU Variables

Variables of the CPU are represented by the symbol name.

Figure 5-6 Display of variables in the HTML file



```
<td>:="Web2Plc".tankLevel:</td>
```

Instead of ":="Web2Plc".tankLevel:", the web page displays the current value from the CPU.

5.2.3 Outputting Texts via Enumerations

Using enumerations, texts and colors are assigned to the individual values of a CPU variable.

Defining Enum types in an HTML file

In the HTML file, one Enum type each has been defined for the alarm text and the text color.

Figure 5-7: Defining Enum types in an HTML file

```
<!-- AWP_Enum_Def Name="AlarmText" Values='
    0:"Tank empty!",
    1:"Tank level below minimum!",
    2:"Tank level between minimum and midth!",
    3:"Tank level between midth and maximum!",
    4:"Tank level over maximum!",
    5:"Tank level overflow!"'
-->
<!-- AWP_Enum_Def Name="AlarmColor" Values='
    0:"red",
    1:"yellow",
    2:"black",
    3:"green"'
-->
```

Assigning Enum types to variables in an HTML file

In the HTML file, the “AlarmColor” Enum type is assigned to the “Web2Plc.alarmColor” variable, and the “AlarmText” Enum type is assigned to the “Web2Plc.alarmText” variable.

Figure 5-8: Assigning Enum types to variables in an HTML file

```
<span style="color:<!-- AWP_Enum_Ref Name='"Web2Plc".alarmColor' Enum="AlarmColor" --> :="Web2Plc".alarmColor: ">
<!-- AWP_Enum_Ref Name='"Web2Plc".alarmText' Enum="AlarmText" -->:="Web2Plc".alarmText:</span>
```

Instead of the values of “:="Web2Plc".alarmText:” and “:="Web2Plc".alarmColor:”, the texts previously assigned in an HTML are output in the assigned colors.

5.2.4 Setting Variables in the CPU with Value and Button

Identifying PLC variables as “AWP_In_Variable” in an HTML file

All variables transferred to the CPU must be identified as AWP_In_Variable.

Figure 5-9: Designating PLC variables as “AWP_In_Variable” in an HTML file

```
<!-- AWP_In_Variable Name='"Web2Plc".start' -->
<!-- AWP_In_Variable Name='"Web2Plc".stop' -->
<!-- AWP_In_Variable Name='"Web2Plc".reset' -->
<!-- AWP_In_Variable Name='"Web2Plc".flowrate' -->
```

Note Keep in mind that the quotation marks are nested. The DB name is written within quotation marks and framed by a single quotation mark together with the variable name.

Transferring PLC variables to an HTML file

To transfer variables to the CPU via the web page, you may use forms.

Figure 5-10: Transferring PLC variables

```
<form>
    <input type="number" min="1" max="10" step="1" id="flowrate" name='"Web2Plc".flowrate' value = ':="Web2Plc".flowrate:'>
    <input type="submit" value="Set a new Flowrate">
</form>
```

When the button “Set a new Flowrate” of type “submit” is pressed, the data is transferred to the CPU.

5.2.5 Setting Variables in the CPU via Button Only

Identifying PLC variables as “AWP_In_Variable” in an HTML file

All variables transferred to the CPU must be identified as AWP_In_Variable.

Figure 5-11: Designating PLC variables as “AWP_In_Variable” in an HTML file

```
<!-- AWP_In_Variable Name='"Web2Plc".start' -->
<!-- AWP_In_Variable Name='"Web2Plc".stop' -->
<!-- AWP_In_Variable Name='"Web2Plc".openValve' -->
<!-- AWP_In_Variable Name='"Web2Plc".closeValve' -->
```

Transferring PLC variables to an HTML file

To assign a predefined value to variables in the CPU, use a form, the “POST” method, and a hidden value.

Figure 5-12 Setting variables with buttons in HTML

```
<td>
    <form method="post" action="">
        <input type="submit" value="OpenValve">
        <input type="hidden" name='"Web2Plc".openValve' value="1">
        <input type="hidden" name='"Web2Plc".closeValve' value="0">
    </form>
</td>

<td>
    <form method="post" action="">
        <input type="submit" value="CloseValve">
        <input type="hidden" name='"Web2Plc".closeValve' value="1">
        <input type="hidden" name='"Web2Plc".openValve' value="0">
    </form>
</td>
```

The form is called with the post method. No specification is necessary for action, because the current page is called by default.

Pressing the “OpenValve” button with input type=“hidden” assigns the value 1 to the “Web2Plc.openValve” variable and the value 0 to the “Web2Plc.closeValve” variable. Then the values of the variables are sent to the web server of the CPU.

Clicking the “CloseValve” button with input type=“hidden” assigns the 0 value to the “Web2Plc.openValve” variable and the 1 value to the “Web2Plc.closeValve” variable. Then the values of the variables are sent to the web server of the CPU.

5.3 Functionality of the HTML File with JavaScript

The following chapter explains the individual sections of the HTML files and the associated JavaScript files in the “WebpageOptimized” folder.

In this example, data is combined into strings for a higher data transfer performance with the FC “DataToString” (see 5.1.4). These strings are transferred to the web browser and split into individual data again using JavaScript. The JSON file contains the information on how to interpret the data in the string.

To display a web page, e.g. the Overview Page, the following files are required:

- OverviewOpti.html
- OverviewOpti.js
- OverviewOpti.json
- siemens_Stylesheet.css
- bootstrap.min.css

All web pages additionally use the Siemens library “S7_framework.js” and the generally accessible library “jquery-2.2.0.min.js”.

Note You can find more information about the optimized data transfer with the Siemens library “S7_framework.js” in the second application example “Examples of the SIMATIC S7 1200/S7 1500 Web Server” in Chapter 18 “High-performance Communication via String” on the same article page.

5.3.1 Outputting CPU Variables

HTML file

In this example, the variables of the CPU are not directly addressed in the HTML file. Instead, an “id” is assigned to the output field.

Figure 5-13: Assign an “id” to an output field in the HTML file

```
<td id="tankLevel"></td>
```

JSON file

The JSON file contains the variables to be read and information on how to interpret the values it contains. The data string is generated with FC “DataToString”.

Figure 5-14: Variable to be read with specification of length and type of the contained values

```
{
"val" : ":="Web2Plc".dataOverviewString:",
"len" : "1;4;4;4;4;4;4;4;4;4;4",
"typ" : "0;2;2;2;2;2;2;2;2;2;2",
"str" : ""
}
```

The following assignments apply to the codes used in the JSON file:

Table 5-2: Interpretation of the codes in the “len” line

len	Data type	Length
1	BOOL	1 bit
2	BYTE	8 bits
4	INT, WORD	16 bits
8	DINT, DWORD, REAL	32 bits
16	LINT, LWORD, LREAL	64 bit

Table 5-3: Interpretation of the codes in the “type” line

type	Data type
0	BOOL
1	UINT
2	INT, DINT
3	REAL
4	LREAL
5	STRING

Note To ensure a correct assignment of the values from the data string, the entries under “len” and “type” must correspond to the PLC variables. Also note the order of the PLC variables as they are combined in FC “DataToString”.

JavaScript file

In a JavaScript file, the variables (data strings) specified in the JSON file are read out with the S7Framework function “readData”.

Figure 5-15: Reading the variables

```
// URL for dataFile
var URL = "Script/OverviewOpti.json"
...
// read Values first time
S7Framework.readData(URL, "init read data", deployValues);
```

The values from the data strings are read one after the other and assigned to the IDs of the HTML page in the “deployValues” function. The sequence of the values corresponds to the sequence in the user program.

Figure 5-16: Assigning the values of the variables to the IDs

```
// function to deploy values into Webpage
function deployValues(values)
{
    var img;
    if (values[0] == false)
    {
        img = 0;
    }
    else
    {
        img = 1;
    }

    $('#statusValveCPU').attr({
        src: img_array[ img ].src,
        title: img_array[ img ].alt,
        alt: img_array[ img ].alt
    });

    $('#tankLevelScaleOverviewImg').height( values[1] + "px");

    $('#actFlowrate').html( values[2] );
    $('#tankLevel').html( values[3] );
    $('#tankLevelOverflow').html( values[4] );
    $('#tankLevelMaximum').html( values[5] );
    $('#tankLevelMidth').html( values[6] );
    $('#tankLevelMinimum').html( values[7] );
    $('#tankLevelLack').html( values[8] );


    $('#Alarm')
        .html( alarmArrayText[values[9]] )
        .css('color', alarmArrayColor[values[10]]);

    // read Values cyclically
    setTimeout(S7Framework.readData(URL, "init read data", deployValues), 1000);
}
```

The S7Framework function “readData” is called again at the end of the “deployValues” function so that the values are read cyclically.

5.3.2 Outputting Colored Texts via Arrays

Arrays for text and colors are defined in a JavaScript file. The values of the variables in the CPU are used as indices for the defined arrays.

Defining arrays in a JavaScript file

In JavaScript, one array was defined for each the alarm text and the text color.

Figure 5-17: Defining an array

```
// Arrays for Alarm message and Text color
var alarmArrayText = new Array(
    "Tank empty!",
    "Tank level below minimum!",
    "Tank level between minimum and midth!",
    "Tank level between midth and maximum!",
    "Tank level over maximum!",
    "Tank level overflow!"
    );
var alarmArrayColor = new Array(
    "red",
    "yellow",
    "black",
    "green"
);
```

Assigning variables to array indices in a JavaScript file

The values of the variables “Web2Plc.alarmText” and “Web2Plc.alarmColor” correspond to the variables “values[9]” and “values[10]” in the JavaScript file. To output the colored texts instead of the values, these variables are used in the “deployValues” function as indices for the defined arrays.

Figure 5-18: Assign colored text to the “Alarm” ID

```
$('#Alarm')
    .html( alarmArrayText[values[9]] )
    .css('color', alarmArrayColor[values[10]]);
```

The text is assigned to the output field with the “Alarm” ID using the “html” function, and the color of the text is assigned with the “css” function.

5.3.3 Outputting Graphics via Arrays

Arrays for graphics are defined in a JavaScript file. The values of the variables in the CPU are used as indices for the defined arrays.

Defining an array in a JavaScript file

In JavaScript, an array has been defined for the valve graphics.

Figure 5-19: Defining an array

```
// image array for valve image
var img_array = new Array();
img_array[0] = new Image();
img_array[0].src = "Images/Valve0.png"; // OFF
img_array[0].alt = "Valve := Closed"; // OFF
img_array[1] = new Image();
img_array[1].src = "Images/Valve1.png"; // ON
img_array[1].alt = "Valve := Open"; // ON
```

Assigning the array index to variables in a JavaScript file

The state of the boolean variable “Web2Plc.statusValveCPU” corresponds to the variable “values[0]” in the JavaScript file. Depending on the state of the variables, the local variable “img” is set to the value “0” or “1”. To output the graphic instead of the values, the variable “img” is used in the “deployValues” function as the index for the defined array.

Figure 5-20: Assign a graphic to the “statusValveCPU” ID

```
var img;
if (values[0] == false)
{
    img = 0;
}
else
{
    img = 1;
}

$('#statusValveCPU').attr({
    src: img_array[ img ].src,
    title: img_array[ img ].alt,
    alt: img_array[ img ].alt
});
```

The output field with the “statusValveCPU” ID is assigned the corresponding graphic.

Note In addition to the possibility to read graphics from arrays, there is also the possibility of “CSS Sprites”, but their definition would be too in-depth for this example.

5.3.4 Setting Variables in the CPU with Value and Button

Identifying PLC variables as “AWP_In_Variable” in a JSON file

All variables transferred to the CPU must be identified as “AWP_In_Variable”.

Figure 5-21: PLC variables as “AWP_In_Variable”

```
<!-- AWP_In_Variable Name='"Web2Plc".start' -->
<!-- AWP_In_Variable Name='"Web2Plc".stop' -->
<!-- AWP_In_Variable Name='"Web2Plc".reset' -->
<!-- AWP_In_Variable Name='"Web2Plc".flowrate' -->
```

Note Keep in mind that the quotation marks are nested. The DB name is written within quotation marks and framed by a single quotation mark together with the variable name.

Input field for variables in an HTML file

To enter variables via the web page, you may use forms. IDs are also used here for the entries.

Figure 5-22: Input field for variables

```
<form>
    <input type="text" id="flowrate" size="2">
    <input type="button" id="flowrateButton" value="Set a new Flowrate">
</form>
```

When the “Set a new Flowrate” button of the “button” type is pressed, the following JavaScript code is executed to write the value to the CPU.

Writing PLC variables with JavaScript

In the JavaScript file, the variable “flowrate” is assigned the value of the input field with the “flowrate” ID. This value is checked to see if it is between 1 and 10. The “data” variable is assigned a string consisting of the PLC variable name and the input value. This string is transferred to the CPU with the S7Framework function “writeData”.

Figure 5-23: Writing PLC variables

```
// URL for dataFile
var URL = "Script/OverviewOpti.json"
 ...
// FLOWRATE Button
$("#flowrateButton").click(function(){
    var flowrate = $("#flowrate").val();
    if ( (flowrate >= 1 && flowrate <= 10 ) == false)
    {
        alert("Value must be between 1 and 10!");
        return;
    }
    else{
        var data = '"Web2Plc".flowrate=' + flowrate;
        S7Framework.writeData(URL, data, "FLOWRATE");
    }
})
```

5.3.5 Setting Variables in the CPU via Button Only

Identifying PLC variables as “AWP_In_Variable” in a JSON file

All variables transferred to the CPU must be identified as “AWP_In_Variable”.

Figure 5-24: PLC variables as “AWP_In_Variable”

```
<!-- AWP_In_Variable Name='"Web2Plc".openValve' -->
<!-- AWP_In_Variable Name='"Web2Plc".closeValve' -->
<!-- AWP_In_Variable Name='"Web2Plc".start' -->
<!-- AWP_In_Variable Name='"Web2Plc".stop' -->
```

Buttons in an HTML file

To set variables in the CPU only by means of buttons, you may use forms. IDs are also used here for the entries.

Figure 5-25: Buttons in an HTML file

```
<tr>
    <td><input type="button" id="openValve" value="OpenValve"></td>
    <td><input type="button" id="closeValve" value="CloseValve"></td>
</tr>
```

When the “OpenValve” button of the “button” type is pressed, the following JavaScript code is executed to set the values to the CPU.

Writing PLC variables with JavaScript

In the JavaScript file, the “data” variable is assigned a string consisting of the PLC variable “Web2Plc.openValve” with the value “1” and the PLC variable “Web2Plc.closeValve” with the value “0”. This string is transferred to the CPU with the S7Framework function “writeData”.

Figure 5-26: Writing PLC variables

```
// URL for dataFile
var URL = "Script/OverviewOpti.json"
 ...
// Open Valve Button
$("#openValve").click(function(){
    var data = '"Web2Plc".openValve=1'+'&'+'"Web2Plc".closeValve=0';
    S7Framework.writeData(URL, data, "OPEN VALVE");
});
```

5.4 How the HTML File Works with Web API (S7-1500)

The following chapter explains the individual sections of HTML files and the associated JavaScript files of the “WebpageAPI” folder.

For displaying a web page (e.g. “OverviewApi.html”), the following additional files are required:

- WebApi.js
- jquery-2.2.0.min.js
- toastr.min.js
- siemens_Stylesheet.css
- bootstrap.min.css

The CPU S7-1500 from firmware version V2.8 offers a web-based API (Web API) as interface for reading and writing CPU data.

As RPC protocol, JSON-RPC V2.0 is based on HTTP. The Web API is accessible via POST requests to the following URL:

https://[ip_address]/api/jsonrpc

Features and benefits of the Web API

- The Web API only supports the “HTTPS” transmission protocol.
- For the output of CPU data, AWP commands are no longer necessary.
- The web pages no longer need to be loaded onto the CPU and, as a result, are independent of user-defined web pages. Additionally, synchronization between the user program and the web server via the “www” command is no longer necessary. This makes it possible to develop “offline” or to connect other applications to the interface.
  The web pages can also be loaded onto the CPU as user-defined web pages, thus they are bound to the PLC and are also loaded with the project from the TIA Portal into the PLC.
- The communication load between Server and Client is reduced compared to the use of AWP commands, since smaller data packets are transferred with JSON.

Note You can find further information on Web API in the function manual for “SIMATIC S7-1500, SIMATIC Drive Controller, ET 200SP, ET 200pro Webserver”

https://support.industry.siemens.com/cs/ww/en/view/59193560/129802886283

5.4.1 Logging into Web API

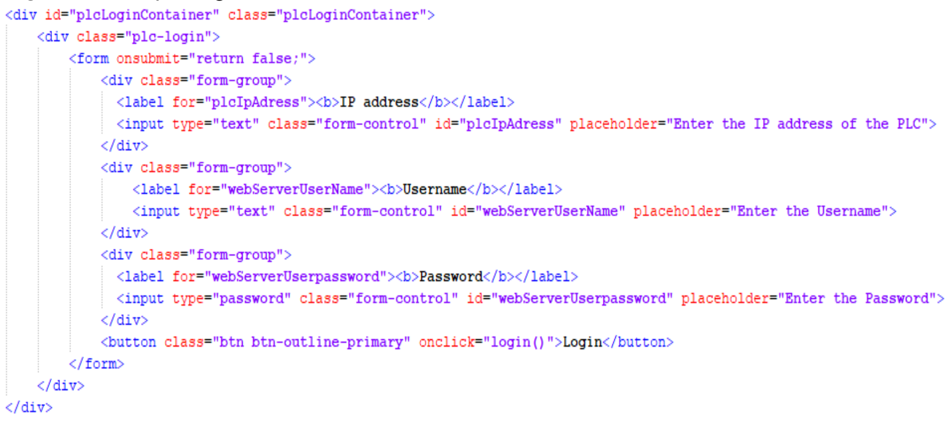
Before data can be exchanged with the CPU, a new Web API session must be opened. To do this, the user must log into the Web API with a user name and password.

HTML file

The web pages (e.g. StartApi.html) consist of two areas: the login area “plcLoginContainer” and the user area “plcTankContainer”.

The login area “plcLoginContainer” contains the login input fields “IP address”, “Username”, and “Password”.

Figure 5-27: “plcLoginContainer” area in the HTML file



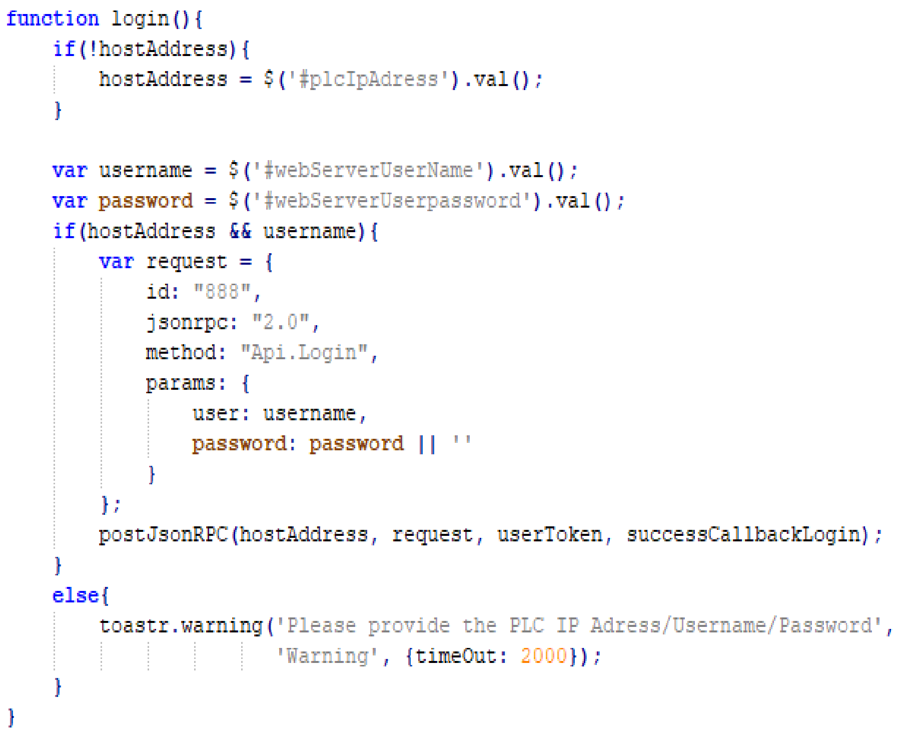
```html
<div id="plcLoginContainer" class="plcLoginContainer">
    <div class="plc-login">
        <form onsubmit="return false;">
            <div class="form-group">
              <label for="plcIpAdress"><b>IP address</b></label>
              <input type="text" class="form-control" id="plcIpAdress" placeholder="Enter the IP address of the PLC">
            </div>
            <div class="form-group">
                <label for="webServerUserName"><b>Username</b></label>
                <input type="text" class="form-control" id="webServerUserName" placeholder="Enter the Username">
            </div>
            <div class="form-group">
              <label for="webServerUserpassword"><b>Password</b></label>
              <input type="password" class="form-control" id="webServerUserpassword" placeholder="Enter the Password">
            </div>
            <button class="btn btn-outline-primary" onclick="login()">Login</button>
        </form>
    </div>
</div>
```

By clicking the “Login” button, the JavaScript function “login()” is executed.

JavaScript file “WebApi.js”

The function “login()” is used to log into the Web API. It reads the login data of the user from the web page and generates a request with the method “Api.Login”. The request is transferred to the Server with the function “postJsonRPC()”. If authentication is successful, a new Web API session is opened and the user receives a token. The token identifies the user as successfully authenticated against the API.

Figure 5-28: JavaScript function “login()” in the JavaScript file



```javascript
function login(){
    if(!hostAddress){
        hostAddress = $('#plcIpAdress').val();
    }

    var username = $('#webServerUserName').val();
    var password = $('#webServerUserpassword').val();
    if(hostAddress && username){
        var request = {
            id: "888",
            jsonrpc: "2.0",
            method: "Api.Login",
            params: {
                user: username,
                password: password || ''
            }
        };
        postJsonRPC(hostAddress, request, userToken, successCallbackLogin);
    }
    else{
        toastr.warning('Please provide the PLC IP Adress/Username/Password',
                       'Warning', {timeOut: 2000});
    }
}
```

After successful login, the “updateWebpage()” function hides the “plcLoginContainer” login area and displays the “plcTankContainer” user area.

5.4.2 Transmitting the Token and IP Address When Changing Web Pages

So that the user does not have to login every time they change web pages, the token of the Web API session and the IP address of the called web page are passed as attributes in the "updateWebpage()" function.

Figure 5-29: Transmitting the token and IP address when changing web pages

```
document.getElementById('overviewLink').setAttribute("href",
"OverviewApi.html?userToken=" + userToken + "&hostAddress=" + hostAddress);
```

When opening the called page, the function "getUrlParameter()" reads the attributes and assigns them to the global variables "userToken" and "hostAddress" for later use.

Figure 5-30: Read out of the transferred token and IP address

```
$(document).ready(function(){
    var urlParameter = getUrlParameter();
    var tokenParam = urlParameter["userToken"];
    var hostParam = urlParameter["hostAddress"];
    if (tokenParam && hostParam)
    {
        userToken = tokenParam;
        hostAddress = hostParam;

        updateWebpage();
    }
```

5.4.3 Outputting CPU Variables

HTML file

In this example, the variables of the CPU are not directly addressed in the HTML file. Instead, an "id" is assigned to the output field.

Figure 5-31: Assign an "id" to an output field in the HTML file

```
<td id="tankLevel"></td>
```

JavaScript file

In the "updateWebpage()", the functions for reading the values from the CPU are called depending on the opened web page.

Figure 5-32: Calling the function "readValuesOverview()" to read the variables

```
setTimeout(readValuesOverview, 1000);
```

The function "readValuesOverview()" is used to read the variables from the CPU for the "OverviewApi.html" web page. To read variables, a request is created with the name of the variable and the "PlcProgram.Read" method. The request is transferred to the Server with the function "postJsonRPC()". The Server returns a response that contains the values of the variables as an array. The values are read from the response one by one with the JavaScript function "JSON.parse()" and assigned to the IDs of the HTML page. The order of the values corresponds to the order of the request.

The following Figure shows only a part of the function “readValuesOverview()”. For the sake of clarity, only the reading of the variable “Web2Plc.tankLevel” is shown.

Figure 5-33: Reading the variable “Web2Plc.tankLevel”

```
function readValuesOverview(){

    var request = [
        ...
        {
        id: "13",
        jsonrpc: "2.0",
        method: "PlcProgram.Read",
        params: {var: "\"Web2Plc\".tankLevel"}
        },
        ...
    ];

    postJsonRPC(hostAddress, request, userToken, function(data){
        ...
        var tankLevel = JSON.parse(data)[3]["result"];
        ...
        $('#tankLevel').html( tankLevel );
        ...
    });
    setTimeout(readValuesOverview, 1000);
}
```

So that the values are read cyclically, the function “readValuesOverview()” is called again at the end.

5.4.4 Outputting Colored Texts via Arrays

Arrays for text and colors are defined in a JavaScript file. The values of the variables in the CPU are used as indices for the defined arrays.

Defining arrays in a JavaScript file

In JavaScript, one array each for the alarm text and the text color was defined.

Figure 5-34: Defining an array

```
// Arrays for Alarm message and Text color
var alarmArrayText = new Array(
    "Tank empty!",
    "Tank level below minimum!",
    "Tank level between minimum and midth!",
    "Tank level between midth and maximum!",
    "Tank level over maximum!",
    "Tank level overflow!"
    );
var alarmArrayColor = new Array(
    "red",
    "yellow",
    "black",
    "green"
);
```

Assigning variables to array indices in a JavaScript file

The values of the variables “Web2Plc.alarmText” and “Web2Plc.alarmColor” are read with the function “readValuesOverview()” and assigned to the variables “alarmText” and “alarmColor”. To output the colored texts instead of the values, these variables are used as indices for the defined arrays.

Figure 5-35: Assign colored text to the “Alarm” ID

```
$('#Alarm')
    .html( alarmArrayText[alarmText] )
    .css('color', alarmArrayColor[alarmColor]);
```

The text is assigned to the output field with the “Alarm” ID using the “html” function, and the color of the text is assigned with the “css” function.

5.4.5 Outputting Graphics via Arrays

Arrays for graphics are defined in a JavaScript file. The values of the variables in the CPU are used as indices for the defined arrays.

Defining an array in a JavaScript file

In JavaScript, an array has been defined for the valve graphics.

Figure 5-36: Defining an array

```
// image array for valve image
var img_valve_array = new Array();
img_valve_array[0] = new Image();
img_valve_array[0].src = "Images/Valve0.png"; // OFF
img_valve_array[0].alt = "Valve := Closed"; // OFF
img_valve_array[1] = new Image();
img_valve_array[1].src = "Images/Valve1.png"; // ON
img_valve_array[1].alt = "Valve := Open"; // ON
```

Assigning the array index to variables in a JavaScript file

The state of the boolean variable “Web2Plc.statusValveCPU” is assigned to the variable “statusValveCPU1” in the JavaScript file. Depending on the state of the variables, the local variable “img” is set to the value “0” or “1”. To output the graphic instead of the values, the variable “img” in the function “readValuesOverview()” is used as index for the defined array.

Figure 5-37: Assign a graphic to the “statusValveCPU” ID

```
postJsonRPC(hostAddress, request, userToken, function(data){
    var statusValveCPU1 = JSON.parse(data)[0]["result"];
    ...
    var img;

    if (statusValveCPU1 == false)
    {
        img = 0;
    }
    else
    {
        img = 1;
    }

    $('#statusValveCPU').attr({
        src: img_valve_array[ img ].src,
        title: img_valve_array[ img ].alt,
        alt: img_valve_array[ img ].alt
    });
```

The output field with the “statusValveCPU” ID is assigned the corresponding graphic.

Note

In addition to the possibility to read graphics from arrays, there is also the possibility of “CSS Sprites”, but their definition would be too in-depth for this example.

5.4.6 Setting Variables in the CPU with Value and Button

Input field for variables in an HTML file

To enter variables via the web page, you may use forms. IDs are also used here for the entries.

Figure 5-38: Input field for variables in an HTML file

```
<form>
    <input type="text" id="flowrate" size="2">
    <input type="button" id="flowrateButton" value="Set a new Flowrate">
</form>
```

When the “Set a new Flowrate” button of the “button” type is pressed, the following JavaScript code is executed to write the value to the CPU.

Writing PLC variables with Web API

In the JavaScript file, the variable “flowrate” is assigned the value of the input field with the “flowrate” ID. This value is checked to see if it is between 1 and 10. If this is the case, the “writeFlowrate()” function is executed.

Figure 5-39: “flowrateButton” event handling

```
// FLOWRATE Button
$("#flowrateButton").click(function(){
    var flowrate = $("#flowrate").val();
    if ( (flowrate >= 1 && flowrate <= 10 ) == false)
    {
        alert("Value must be between 1 and 10!");
        return;
    }
    else{
        writeFlowrate(flowrate);
    }
})
```

The function “writeFlowrate()” is used to write the variable “flowrate”. To write the variables, a request is created with the name of the PLC variable, the value, and the “PlcProgram.Write” method. The request is transferred to the Server with the function “postJsonRPC()”.

Figure 5-40: Writing PLC variables with Web API

```
function writeFlowrate(flowrate){
    var request =       {
        id: "110",
        jsonrpc: "2.0",
        method: "PlcProgram.Write",
        params: {var: "\"Web2Plc\".flowrate", value: parseInt(flowrate)}
        };

    postJsonRPC(hostAddress, request, userToken, function(data){
    });
}
```

5.4.7 Setting Variables in the CPU via Button Only

Buttons in an HTML file

To set variables in the CPU only by means of buttons, you may use forms. IDs are also used here for the entries.

Figure 5-41: Buttons in an HTML file

```
<tr>
    <td><input type="button" id="openValve" value="OpenValve"></td>
    <td><input type="button" id="closeValve" value="CloseValve"></td>
</tr>
```

When the “OpenValve” button of the “button” type is pressed, the following JavaScript function is executed to set the values in the CPU.

Writing PLC variables with Web API

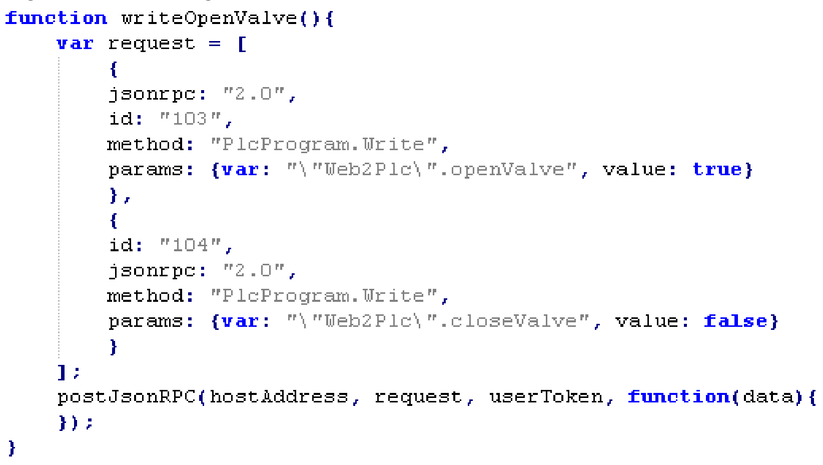
In JavaScript, clicking the “OpenValve” button executes the “writeOpenValve()” function.

Figure 5-42: “OpenValve” event handling

```
// Open Valve Button
$("#openValve").click(function(){
    writeOpenValve();
});
```

The function “writeOpenValve()” sets the PLC variable “Web2Plc.openValve” to the value “true” and the PLC variable “Web2Plc.closeValve” to the value “false”. A request with the method “PlcProgram.Write” is created for writing the two variables. The request is transferred to the Server with the function “postJsonRPC()”.

Figure 5-43: Writing PLC variables with Web API



```
function writeOpenValve(){
    var request = [
        {
        jsonrpc: "2.0",
        id: "103",
        method: "PlcProgram.Write",
        params: {var: "\"Web2Plc\".openValve", value: true}
        },
        {
        id: "104",
        jsonrpc: "2.0",
        method: "PlcProgram.Write",
        params: {var: "\"Web2Plc\".closeValve", value: false}
        }
    ];
    postJsonRPC(hostAddress, request, userToken, function(data){
    });
}
```

6 Project Planning and Configuration

This chapter contains all information on how you can create and operate a web page for a CPU with PN interface for yourself.

The CPU 1511-1 PN is used as an example in this chapter. All steps are illustrated using the finished application example.

If you only want to put the finished application example into operation, please continue reading in Chapter 7.

6.1 Procedure for Creating a Web Page

The configuration and settings in STEP 7 and the writing of the HTML file are closely linked. The following procedure is recommendable for that:

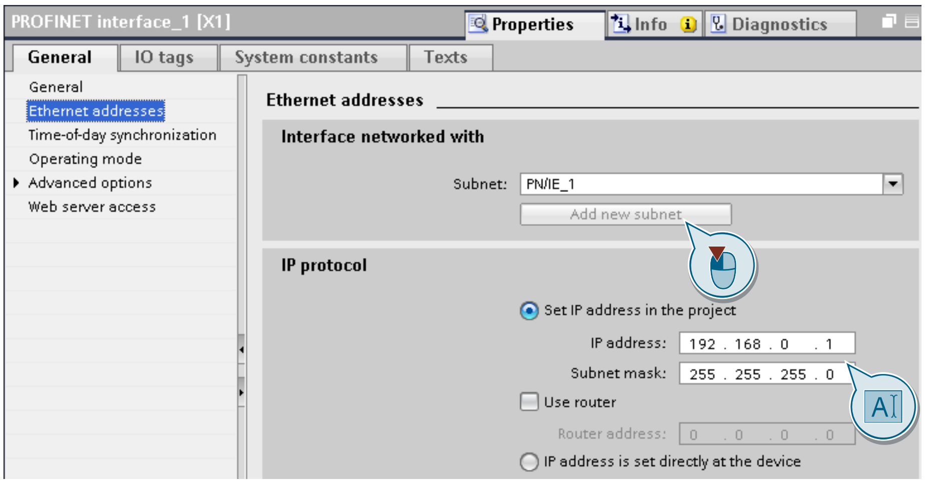
1. Configuring the Hardware
2. Creating the Variables in the S7 Program
3. Creating the HTML Files
4. Configuring Web Server Settings and Generating Data Blocks
5. Creating, Compiling, and Loading the S7 Program

6.2 Configuring the Hardware

1. Start the TIA Portal and select “Project > New…” to create a new project with the name “Webserver_S7_1500” or “Webserver_S7_1200”, for example.
2. Add a S7-1500 or S7-1200 station via “Add new device > CPU > SIMATIC S7-1500/1200”

   The Device View of the CPU opens.
3. Click “Add new subnet” in the properties of the Ethernet interface.
4. Assign the IP address 192.168.0.1 of the CPU to the Ethernet interface. Via this IP address, you may later use the web browser to access the web page of the CPU.

Figure 6-1



Subnet: PN/IE_1

Set IP address in the project

IP address: 192 . 168 . 0 . 1

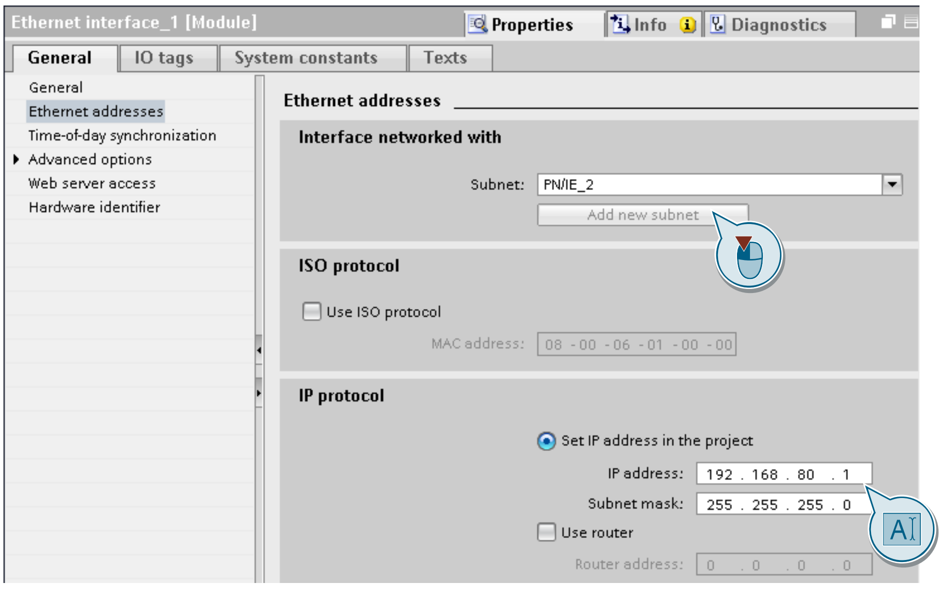
Subnet mask: 255 . 255 . 255 . 0

Router address: 0 . 0 . 0 . 0

The following steps are **optional** for added security when using a SIMATIC S7-1500 CPU and are **not** part of the TIA Portal project.

1. Insert the module ("Communication modules > PROFINET/Ethernet > CP 1543-1 > 6GK7 543 1AX00 0XE0") from the hardware catalog.
2. Click "Add new subnet" in the properties of the CP Ethernet interface.
3. Assign the IP address "192.168.80.1" of the CP to the Ethernet interface. Via this IP address, you may also later access the web page of the CPU with your web browser.

Figure 6-2


Ethernet interface_1 [Module]
Properties | Info | Diagnostics
General | IO tags | System constants | Texts

General
Ethernet addresses
Time-of-day synchronization
Advanced options
Web server access
Hardware identifier

Ethernet addresses

Interface networked with

Subnet: PN/IE_2
Add new subnet

ISO protocol

Use ISO protocol
MAC address: 08 - 00 - 06 - 01 - 00 - 00

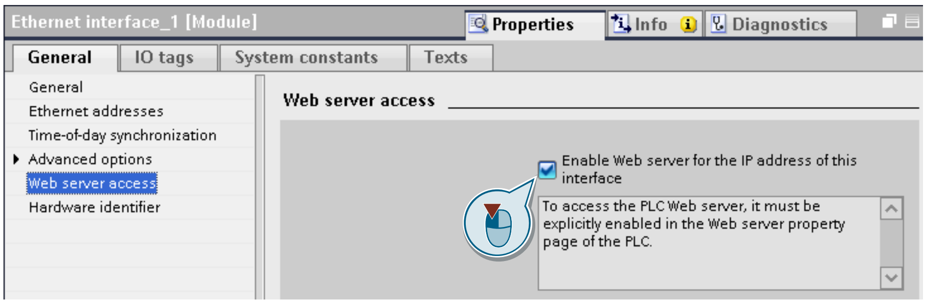
IP protocol

Set IP address in the project
IP address: 192 . 168 . 80 . 1
Subnet mask: 255 . 255 . 255 . 0
Use router
Router address: 0 . 0 . 0 . 0

4. Click "Web server access" in the properties of the CP Ethernet interface.

   Enable "Enable Web server for the IP address of this interface"

Figure 6-3


Ethernet interface_1 [Module]
Properties | Info | Diagnostics
General | IO tags | System constants | Texts

General
Ethernet addresses
Time-of-day synchronization
Advanced options
Web server access
Hardware identifier

Web server access

✓ Enable Web server for the IP address of this interface

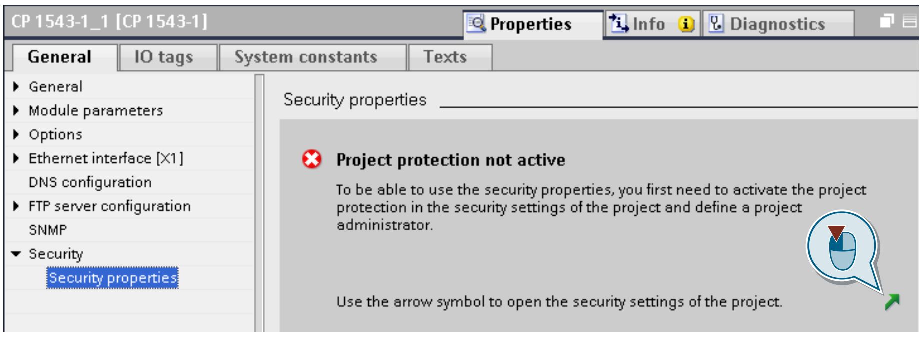
To access the PLC Web server, it must be explicitly enabled in the Web server property page of the PLC.

5. Click "Security" in the properties of the CP.

   Click the green arrow to open the project's security settings.

   Configuring the "Security" is only possible after a user with the appropriate configuration rights has logged in.

Figure 6-4



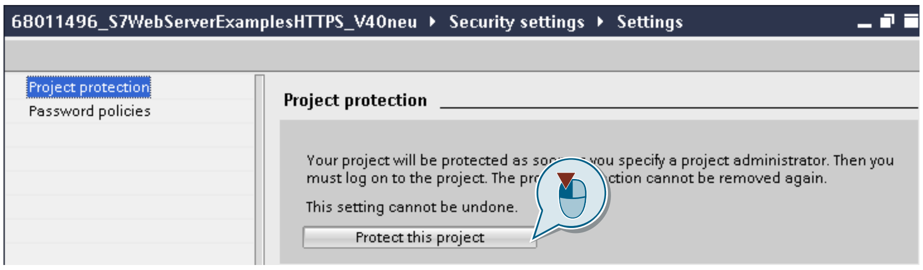
Project protection not active

To be able to use the security properties, you first need to activate the project protection in the security settings of the project and define a project administrator.

Use the arrow symbol to open the security settings of the project.

6. Click “Protect this project”.

Figure 6-5



Your project will be protected as soon as you specify a project administrator. Then you must log on to the project. The protection cannot be removed again.

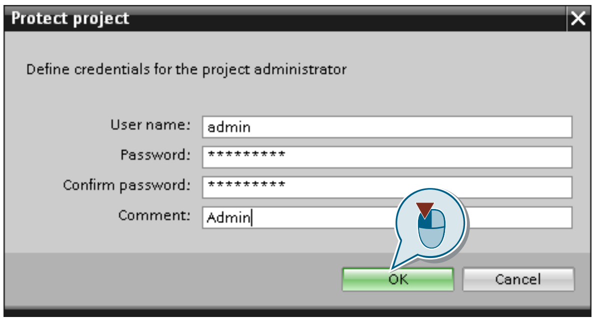
This setting cannot be undone.

7. Enter the following login data and confirm with “OK”:

User name:	admin
Password:	Security1

Caution: The project protection cannot be removed again.

Figure 6-6

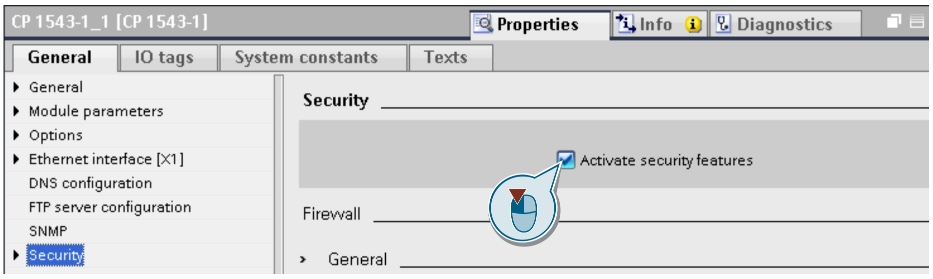


Define credentials for the project administrator

User name: admin
Password: *********
Confirm password: *********
Comment: Admin

8. Click “Security” in the properties of the CP and enable “Activate security features”.

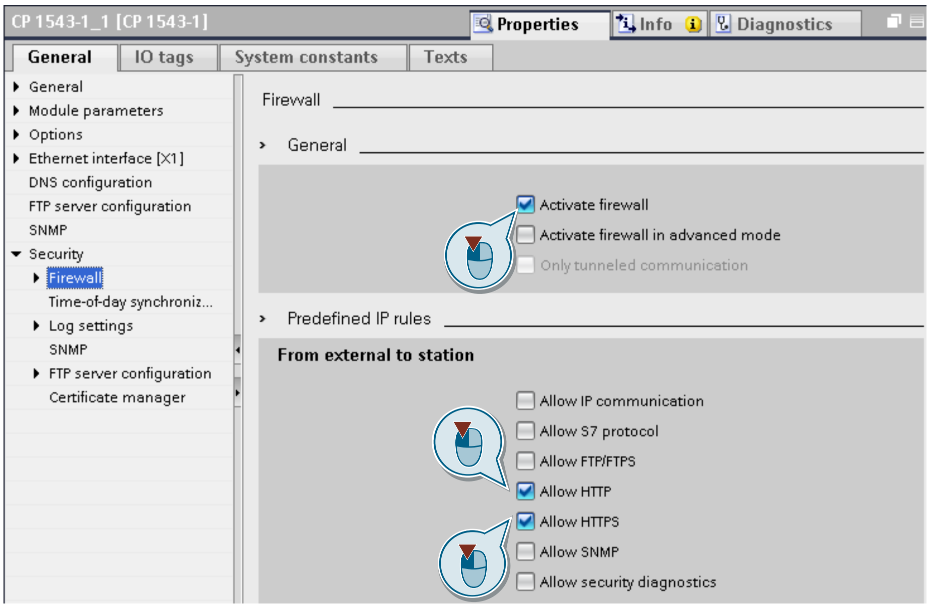
Figure 6-7



☑ Activate security features

9. Open the “Security” area and select “Firewall”.
   Enable the following:
   - “Activate firewall”
   - “Allow HTTP”
   - “Allow HTTPS”

Figure 6-8



CP 1543-1_1 [CP 1543-1]

Properties | Info | Diagnostics

General | IO tags | System constants | Texts

- General
- Module parameters
- Options
- Ethernet interface [X1]
- DNS configuration
- FTP server configuration
- SNMP
- Security
  - Firewall
  - Time-of-day synchroniz...
  - Log settings
  - SNMP
  - FTP server configuration
  - Certificate manager

Firewall

General

- [✓] Activate firewall
- [ ] Activate firewall in advanced mode
- [ ] Only tunneled communication

Predefined IP rules

From external to station

- [ ] Allow IP communication
- [ ] Allow S7 protocol
- [ ] Allow FTP/FTPS
- [✓] Allow HTTP
- [✓] Allow HTTPS
- [ ] Allow SNMP
- [ ] Allow security diagnostics

6.3 Creating Variables in the Variable Table or DB

Create a data block (DB) or a variable table and insert the desired variables.

The variables of this application example are described in Chapter 5.1.

6.4 Creating the HTML files

To create the HTML file, you need the list of variables from Chapter 5.1 and an appropriate editor.

There are convenient editors such as Microsoft Expression Web, which automatically create tags or mark color-correct entries during creation, or simple editors such as Notepad++.

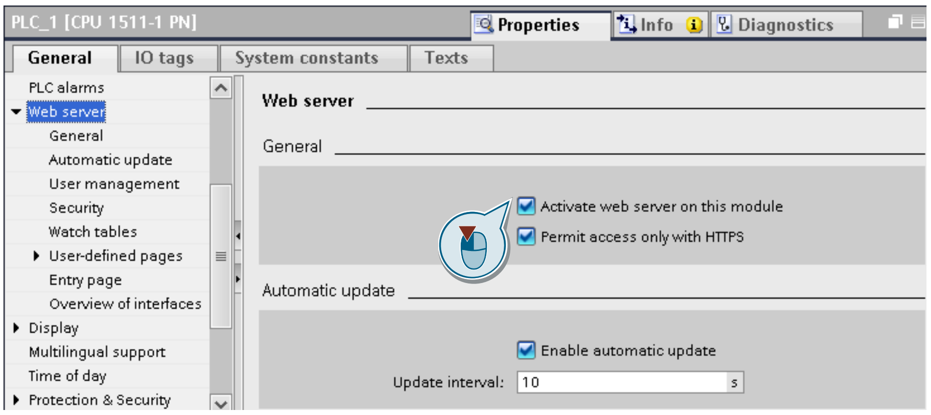
Create the HTML files with an editor. Save the HTML files with the required images, style sheets, and scripts in the “\UserFiles\Webpages” directory.

You can find detailed information on creating the HTML file in Chapters 5.2, 5.3, and 5.4.

6.5 Web Server Settings and Generating Data Blocks

1. Click “Web server” in the properties of the CPU.
2. Enable “Activate Web server on this module” and confirm the security warning.
3. The “Permit access only with HTTPS” option is enabled by default. If you do not want to allow secure access to the default Web pages, uncheck the check box.
4. Check whether “Enable automatic update” is enabled.

Figure 6-9



Web server

General

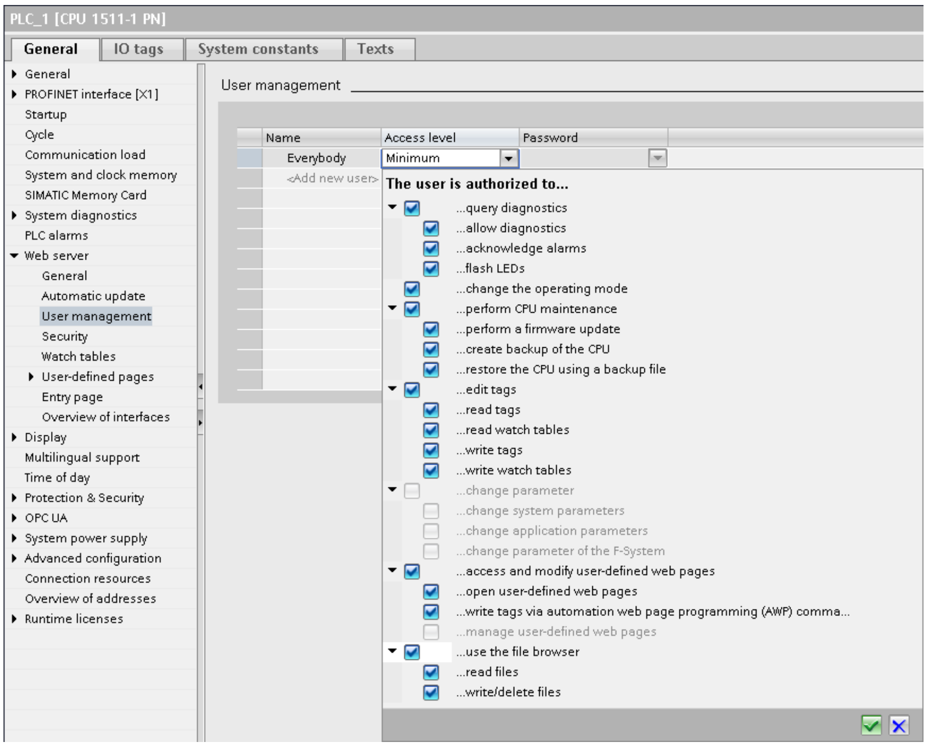
☑ Activate web server on this module
☑ Permit access only with HTTPS

Automatic update

☑ Enable automatic update
Update interval: 10 s

5. Give all rights to the user “Everybody”.
   Alternatively, you can create your own user with a password.

Figure 6-10



User management

Name	Access level	Password
Everybody	Minimum	
<Add new user>		

The user is authorized to...

- ☑ ...query diagnostics
  - ☑ ...allow diagnostics
  - ☑ ...acknowledge alarms
  - ☑ ...flash LEDs
- ☑ ...change the operating mode
- ☑ ...perform CPU maintenance
  - ☑ ...perform a firmware update
  - ☑ ...create backup of the CPU
  - ☑ ...restore the CPU using a backup file
- ☑ ...edit tags
  - ☑ ...read tags
  - ☑ ...read watch tables
  - ☑ ...write tags
  - ☑ ...write watch tables
- ☐ ...change parameter
  - ☐ ...change system parameters
  - ☐ ...change application parameters
  - ☐ ...change parameter of the F-System
- ☑ ...access and modify user-defined web pages
  - ☑ ...open user-defined web pages
  - ☑ ...write tags via automation web page programming (AWP) comma...
  - ☐ ...manage user-defined web pages
- ☑ ...use the file browser
  - ☑ ...read files
  - ☑ ...write/delete files

6. Specify the directory of your HTML files and select a start HTML page from the directory. Assign an application name (for example, “OwnWebsites”).

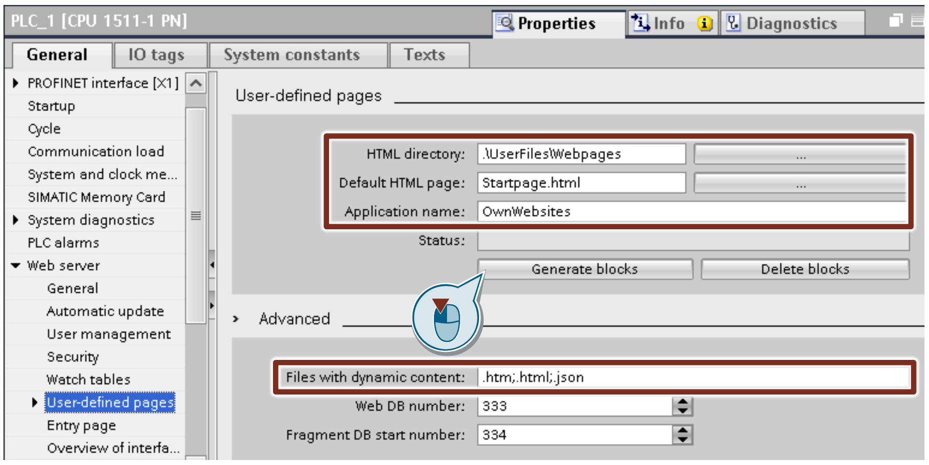
Note

The web pages are stored in the project directory under the “UserFiles” folder and are thus bound to the TIA Portal project.

If you only want to use one of the three examples, you can also select “.\UserFiles\Webpages\WebpageOptimized” as the HTML directory and “StartOpti.html” as the start HTML page.

7. Under “Files with dynamic content:”, add the extension “.json”.
8. Generate the Web_Control_DB (default: DB333) and the Fragment_DBs (default: from DB334) by clicking “Generate blocks”.

Figure 6-11



PLC_1 [CPU 1511-1 PN]

User-defined pages

HTML directory: .\UserFiles\Webpages

Default HTML page: Startpage.html

Application name: OwnWebsites

Status:

Advanced

Files with dynamic content: .htm;.html;.json

Web DB number: 333

Fragment DB start number: 334

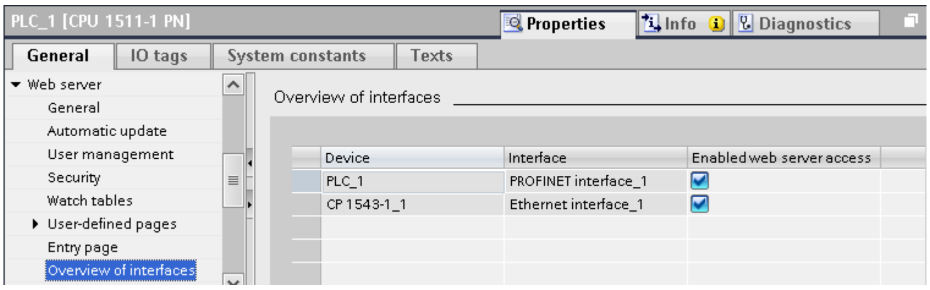
Note

STEP 7 verifies the project with regard to the tags, loads the complementary files, such as the enumerations or images, reads in the variables of the HTML file, verifies the fragments, and writes all data in DB333 and from DB334.

The status of the generation is displayed in an independent window or in the Inspector window under info.

9. Enable access to the web server for the required interfaces.

Figure 6-12



PLC_1 [CPU 1511-1 PN]

Overview of interfaces

Device	Interface	Enabled web server access
PLC_1	PROFINET interface_1	✓
CP 1543-1_1	Ethernet interface_1	✓

The CP is **not** configured in the application example.

6.6 Creating, Compiling, and Loading the S7 Program

In the Appendix of this article, you can find an example S7 program. When creating your own S7 program, you must pay attention to the following points:

- Call the “WWW” (SFC99) command. The “WWW” command initializes the web server of the CPU. With the cyclic calling of the “WWW” command, you ensure that changed variables of the CPU can be displayed on the web page.
  The cyclic calling of the “WWW” command is done in OB1.
- Specify the number of the Web Control DB (for example, 333) in the input parameter CTRL_DB of the “WWW” command.

1. Compile the project by right-clicking the CPU and choose “Compile > Hardware and software (only changes)”.
2. Download the project onto the CPU. To do this, right-click the CPU and choose “Download to device > Hardware and software (only changes)”.

   If necessary, set your PG/PC interface in the dialog window:

   Select the CPU and then click “Load”.

Note If you want to use another CPU, you may change the CPU under “Devices & Networks”.

7 Installation

7.1 Installing Hardware and Software

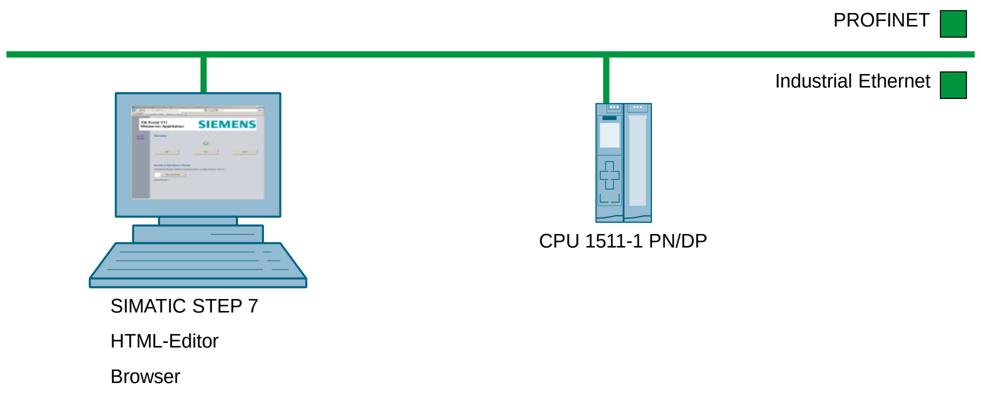
Hardware installation

The following figure shows the hardware setup of the application example.

The PC with the web browser must be connected to the CPU via Industrial Ethernet, e.g.

- directly at the PN interface of the CPU.
- directly at communication module CP 1543-1.
  (The CP is **not** configured in the application example)
- via a switch.

Figure 7-1: Hardware setup of the application example



Note Please observe the installation and connection guidelines from the corresponding manuals.

Installing the software

1. Install SIMATIC STEP 7 (TIA Portal).
2. Install a web page creation tool, such as Notepad++, on the PC that you want to use to create the web page.
3. Install a web browser such as Google Chrome on the PC with which you want to access the web page of the CPU.

7.2 Installation of the Application Example

To put the application example into operation, proceed as follows:

1. Unzip the file
   “68011496_S7-1200_1500_Webserver_CODE_v40.zip”
   into your project directory.
2. Start SIMATIC STEP 7 (TIA PORTAL).
3. Switch to the Device View.
4. If you are using a different CPU, change the device.
5. In the CPU properties, assign the IP address of your CPU to the Ethernet interface. (see Chapter 6.2)
6. Select the S7-1500 and load the entire project onto the CPU.
7. Start a web browser and access the web page of your CPU via the IP address. (see Chapter 8)

8 Operating the Application Example

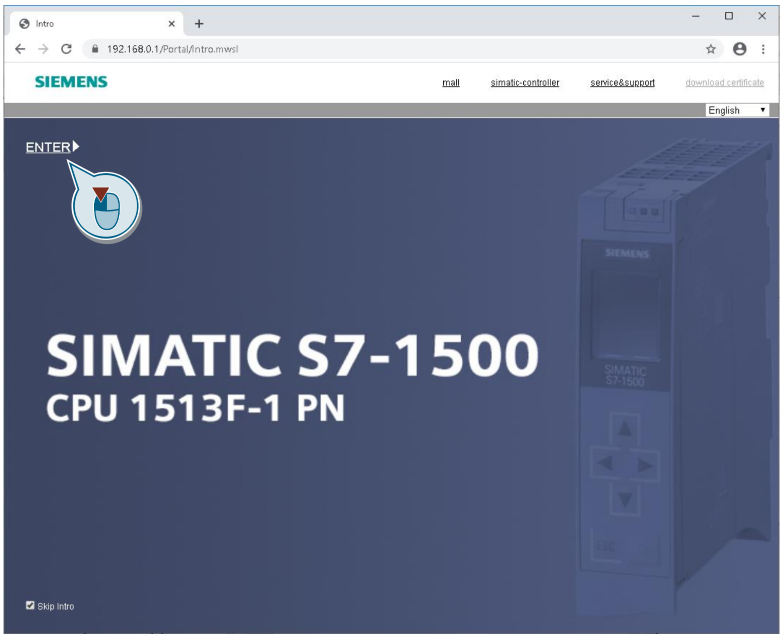
8.1 Opening User-Defined Pages

Open the custom page as follows:

1. Start a web browser, such as Google Chrome.
   For the address, enter the IP address of the CPU, e.g. https://192.168.0.1 or the CP, https://192.168.80.1

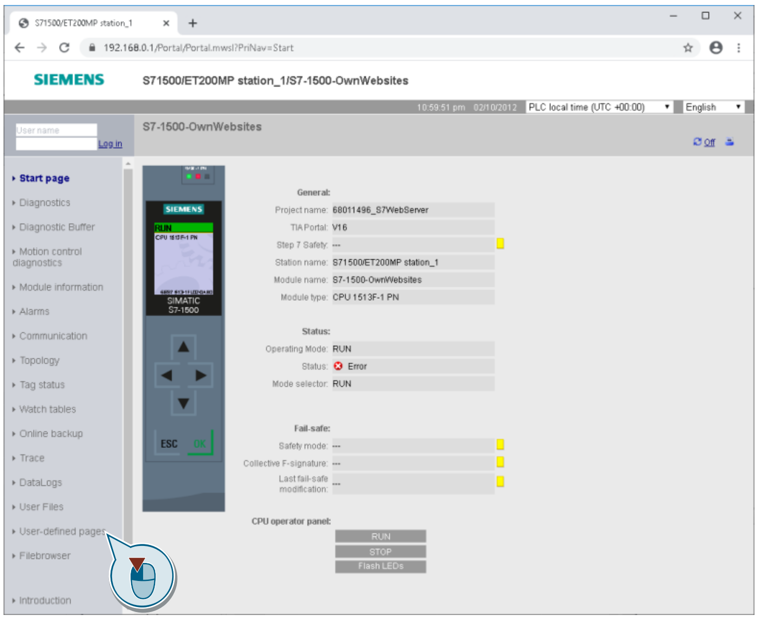
   The CPU's intro web page opens. Click "ENTER".

   Figure 8-1

2. Click "User-defined pages".

   Figure 8-2

   General:

Project name:	68011496_S7WebServer
TIA Portal:	V16
Step 7 Safety:	---
Station name:	S71500/ET200MP station_1
Module name:	S7-1500-OwnWebsites
Module type:	CPU 1513F-1 PN

   Status:

Operating Mode:	RUN
Status:	Error
Mode selector:	RUN

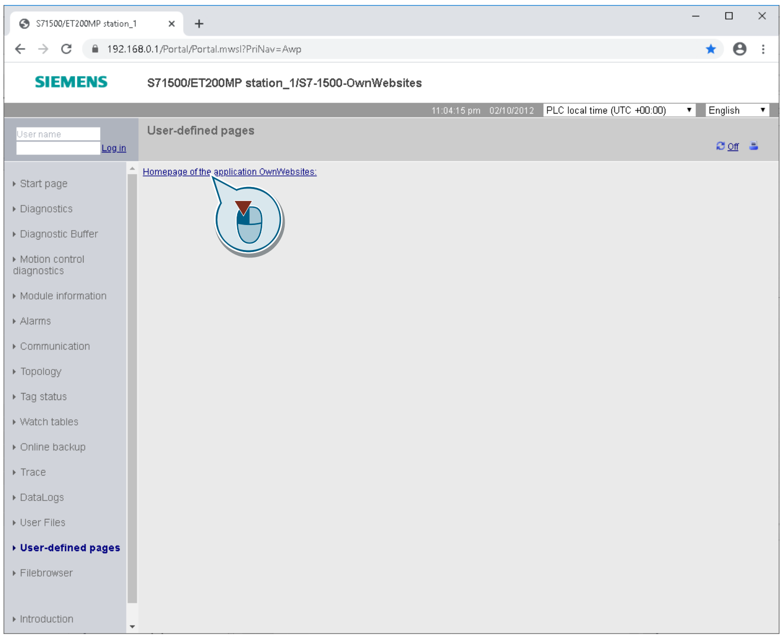
   Fail-safe:

Safety mode:	---
Collective F-signature:	---
Last fail-safe modification:	---

3. To start the example, click “Homepage of the application OwnWebsites:”.

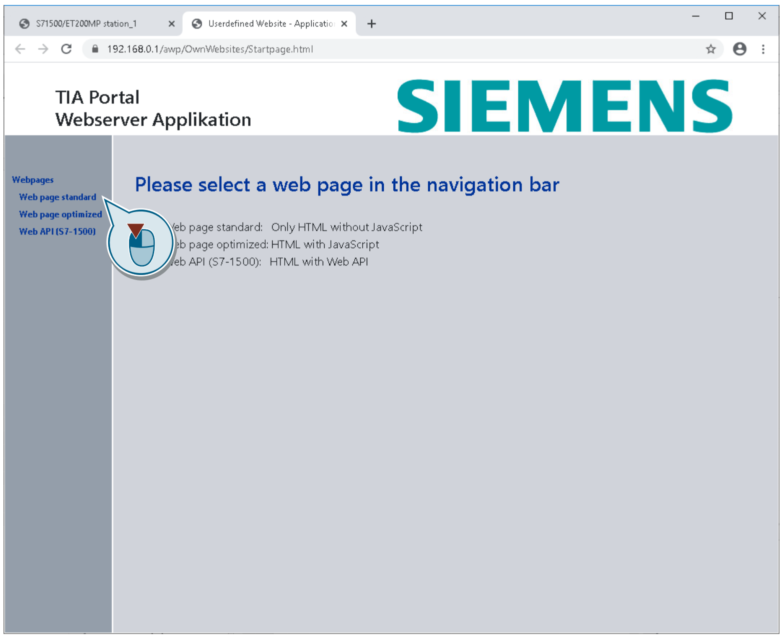
   The “Startpage” web page opens.

   Figure 8-3



4. On the “Startpage” web page, you can choose between the different examples in the navigation bar:
   - Web page standard: HTML without JavaScript
   - Web page optimized: HTML with JavaScript
   - Web API (S7-1500): HTML with Web API and JavaScript

   Figure 8-4

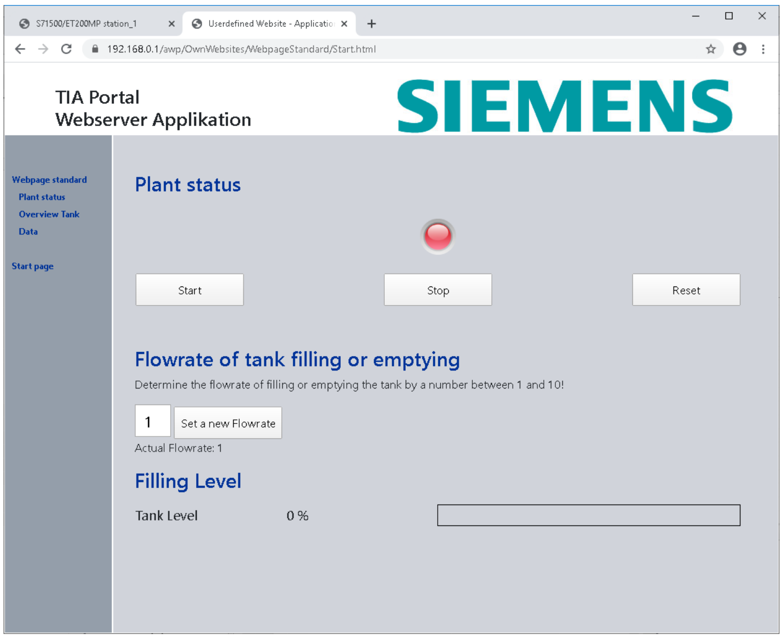

S71500/ET200MP station_1
Userdefined Website - Applicatio
192.168.0.1/awp/OwnWebsites/Startpage.html
TIA Portal
Webserver Applikation
SIEMENS
Webpages
Web page standard
Web page optimized
Web API (S7-1500)
Please select a web page in the navigation bar
Web page standard: Only HTML without JavaScript
Web page optimized: HTML with JavaScript
Web API (S7-1500): HTML with Web API

8.2 Operating the “Web Page Standard”

The “Web page standard” example is operated as follows:

1. The “Start” (“Plant Status”) web page has the following functions:
   - Clicking the “Start” button starts the application in the CPU.
   - By clicking “Stop”, the application is stopped.
   - The “Reset” button resets the application to its original state.
   - The operating status of the application is indicated by the LED (red: off; green: on).
   - You may change the flow rate with the arrow keys in the input field or by entering a value and sending it to the CPU via the button “Set a new Flowrate”. The permissible range is between 1 and 10. A flow rate of 1 is preset in the S7 program.
   - In the “Filling Level” area, the “Tank Level” is displayed as a percentage.
   - Via the links on the navigation bar, you can switch between the web pages.

Figure 8-5



Determine the flowrate of filling or emptying the tank by a number between 1 and 10!

Actual Flowrate: 1

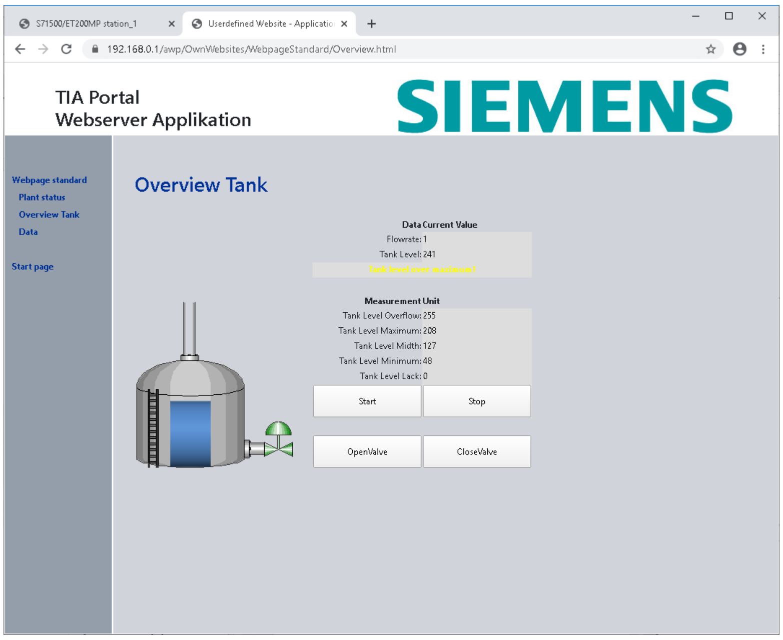
Tank Level 0 %

The web page is automatically updated after five seconds.

Note During the update of the web page, inputs (e.g. “Start”) are not transferred to the CPU. This also applies to the following web pages of this example.

2. The “Overview” (“Overview Tank”) web page has the following functions:
   - The fill level is displayed graphically in the tank.
   - In the “Data Current Value “ section, the “Flow Rate”, the “Tank Level”, and a colored message text is displayed. The colored message text depends on the current fill level.
   - The limit values of the filling level are displayed in the “Measurement Unit” area.
   - Clicking the “Start” button starts the application in the CPU.
   - Click “Stop” to stop the application and close the valve.
   - Click the “OpenValve” button to open the valve. To do this, the application must be started. The valve changes in color from “red” to “green”. The tank is emptied.
   - Click the “CloseValve” button to close the valve. The valve changes in color from “green” to “red”. The tank is refilled.
   - Via the links on the navigation bar, you can switch between the web pages.

Figure 8-6



Overview Tank

Data Current Value

Flowrate: 1
Tank Level: 241
Tank level over maximum!

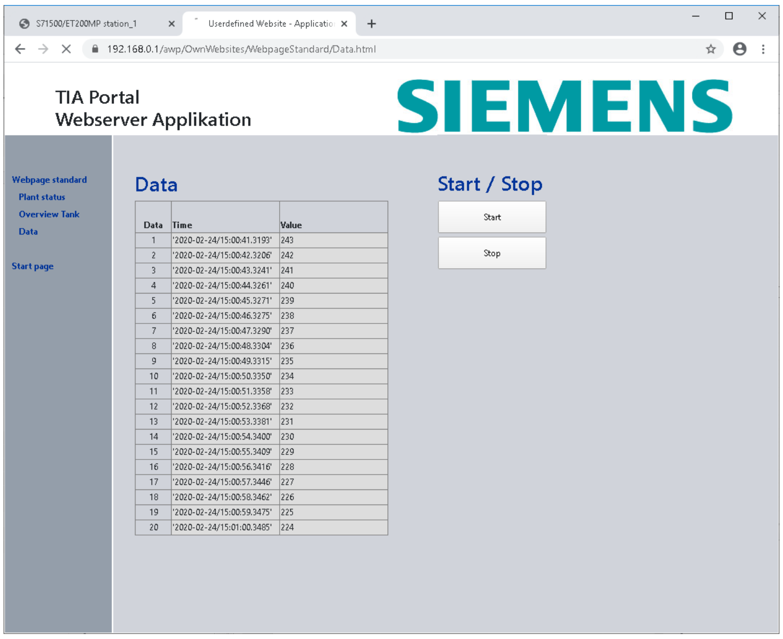
Measurement Unit

Tank Level Overflow: 255
Tank Level Maximum: 208
Tank Level Midth: 127
Tank Level Minimum: 48
Tank Level Lack: 0

The web page is automatically updated after two seconds.

3. The web page “Data” has the following functions:
   - The last 20 value pairs for “Tank Level” and timestamp are output in the table.
   - Clicking the “Start” button starts the application in the CPU.
   - By clicking “Stop”, the application is stopped.
   - Via the links on the navigation bar, you can switch between the web pages.

Figure 8-7



Data

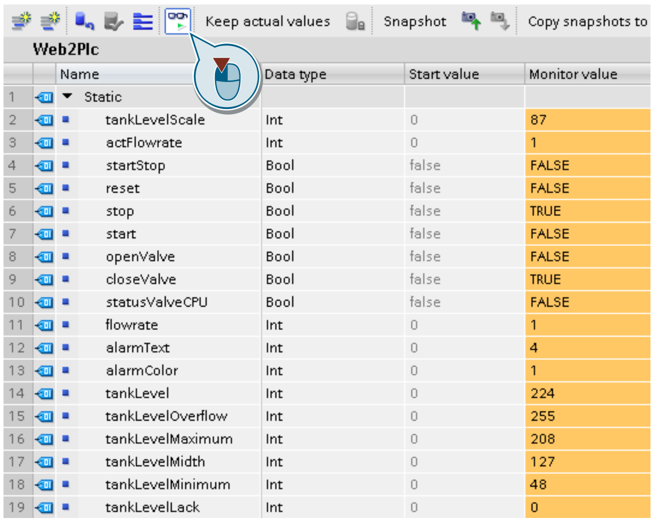
Data	Time	Value
1	'2020-02-24/15:00:41.3193'	243
2	'2020-02-24/15:00:42.3206'	242
3	'2020-02-24/15:00:43.3241'	241
4	'2020-02-24/15:00:44.3261'	240
5	'2020-02-24/15:00:45.3271'	239
6	'2020-02-24/15:00:46.3275'	238
7	'2020-02-24/15:00:47.3290'	237
8	'2020-02-24/15:00:48.3304'	236
9	'2020-02-24/15:00:49.3315'	235
10	'2020-02-24/15:00:50.3350'	234
11	'2020-02-24/15:00:51.3358'	233
12	'2020-02-24/15:00:52.3368'	232
13	'2020-02-24/15:00:53.3381'	231
14	'2020-02-24/15:00:54.3400'	230
15	'2020-02-24/15:00:55.3409'	229
16	'2020-02-24/15:00:56.3416'	228
17	'2020-02-24/15:00:57.3446'	227
18	'2020-02-24/15:00:58.3462'	226
19	'2020-02-24/15:00:59.3475'	225
20	'2020-02-24/15:01:00.3485'	224

The web page is automatically updated after two seconds.

Click the “Start page” link to return to the Start Page.

4. In SIMATIC STEP 7 (TIA Portal), you can also monitor changes to variables in the “Web2Plc” DB by clicking “Monitor all”.

Figure 8-8



Web2Plc

	Name	Data type	Start value	Monitor value
1	Static			
2	tankLevelScale	Int	0	87
3	actFlowrate	Int	0	1
4	startStop	Bool	false	FALSE
5	reset	Bool	false	FALSE
6	stop	Bool	false	TRUE
7	start	Bool	false	FALSE
8	openValve	Bool	false	FALSE
9	closeValve	Bool	false	TRUE
10	statusValveCPU	Bool	false	FALSE
11	flowrate	Int	0	1
12	alarmText	Int	0	4
13	alarmColor	Int	0	1
14	tankLevel	Int	0	224
15	tankLevelOverflow	Int	0	255
16	tankLevelMaximum	Int	0	208
17	tankLevelMidth	Int	0	127
18	tankLevelMinimum	Int	0	48
19	tankLevelLack	Int	0	0

8.3 Operating the “Web Page Optimized”

In this example, the entire web page is not updated, but only the values of the displayed variables.

The operation of the “Web page optimized” example differs from the operation of the “Web page standard” web pages only in the input of the flow rate. Here, the value cannot be changed with the arrow keys in the input field.

8.4 Operating “Web API (S7-1500)”

This example also does not update the entire web page, but only the values of the displayed variables.

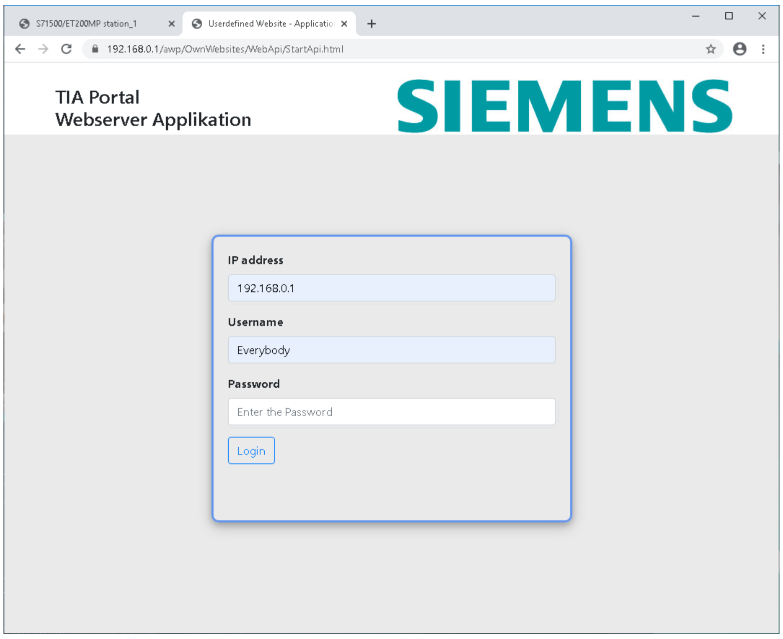
Operating the “Web API (S7-1500)” example differs from the operation of the “Web page standard” web pages when entering the flow rate. Here, the value cannot be changed with the arrow keys in the input field.

In addition, the user must first open a new Web API session by logging in with their user name and password. To open a Web API session, log in with the following login data:

- IP address: 192.168.0.1
- Username: Everybody
- Password: without

Then click the “Login” button.

Figure 8-9



S71500/ET200MP station_1
Userdefined Website - Applicatio
192.168.0.1/awp/OwnWebsites/WebApi/StartApi.html
TIA Portal
Webserver Applikation
SIEMENS
Login

IP address
192.168.0.1
Username
Everybody
Password
Enter the Password

After successful login, the “StartAPI” web pages are opened.

Note The web pages “Web API (S7-1500)” do not have to be loaded onto the CPU as user defined web pages. The web pages can also be opened directly by double-clicking the HTML file, e.g. “StartAPI.html” in Windows Explorer.

9 Appendix

9.1 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:
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:
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

9.2 Links and literature

Table 9-1

No.	Subject
\1\	Siemens Industry Online Support https://support.industry.siemens.com
\2\	Link to the article page of the Application Example https://support.industry.siemens.com/cs/ww/en/view/68011496
\3\	HTML5, JavaScript http://www.w3schools.com/html/ http://www.w3schools.com/js/
\4\	SIMATIC S7 S7-1200 Programmable Controller https://support.industry.siemens.com/cs/ww/en/view/109764129
\5\	S7-1500 web server function manual https://support.industry.siemens.com/cs/ww/en/view/59193560
\6\	S7-1500 system manual https://support.industry.siemens.com/cs/ww/en/view/59191792
\7\	Simple examples for the web server https://support.industry.siemens.com/cs/ww/en/view/68011496
\8\	SIMATIC STEP 7 Basic/Professional V16 and SIMATIC WinCC V16 system manual https://support.industry.siemens.com/cs/ww/en/view/109773506/119453612171
\9\	HTML and CSS, Practical Formulas for Beginners Robert R. Agular mitp ISBN 978-3-8266-1779-9
\10\	HTML manual Stefan Münz/Wolfgang Nefzger Franzis Verlag (Publishing House) ISBN 3-7723-6654-6
\11\	JavaScript and Ajax, the comprehensive manual Christian Wenz Galileo Press ISBN 978-3-8362-1128-4

9.3 Change documentation

Table 9-2

Version	Date	Change
V1.0	02/2014	First edition
V2.0	06/2015	The applications “Creating and Using User-Defined Web Pages for S7-1200” and “Creating and Using User-Defined Web Pages for S7-1500” were merged.
V2.1	10/2015	Correction
V2.2	10/2016	The application example is programmed exclusively with HTML5.
V3.0	09/2019	Update TIA Portal V15.1 (v3.0.1: project user added)
V4.0	03/2020	Extension of the application example with • HTML pages with JavaScript • HTML pages with Web API and JavaScript