Examples for the SIMATIC S7-1200 / S7-1500 Web Server

STEP 7 Basic (TIA Portal), STEP 7 Professional (TIA Portal)

Warranty and Liability

Note

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

We do not accept any liability for the information contained in this document. Any claims against us – based on whatever legal reason – resulting from the use of the examples, information, programs, engineering and performance data etc., described in this Application Example shall be excluded. Such an exclusion shall not apply in the case of mandatory liability, e.g. under the German Product Liability Act ("Produkthaftungsgesetz"), in case of intent, gross negligence, or injury of life, body or health, guarantee for the quality of a product, fraudulent concealment of a deficiency or breach of a condition which goes to the root of the contract ("wesentliche Vertragspflichten"). The damages for a breach of a substantial contractual obligation are, however, limited to the foreseeable damage, typical for the type of contract, except in the event of intent or gross negligence or injury to life, body or health. The above provisions do not imply a change of the burden of proof to your detriment.

Any form of duplication or distribution of these Application Examples or excerpts hereof is prohibited without the expressed consent of the Siemens AG.

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 only form one element of such a concept. Customer is responsible to prevent unauthorized access to its plants, systems, machines and networks. Systems, machines and components should only be connected to the enterprise network or the internet if and to the extent necessary and with appropriate security measures (e.g. use of firewalls and network segmentation) in place. Additionally, Siemens’ guidance on appropriate security measures should be taken into account. For more information about industrial security, please visit http://www.siemens.com/industrialsecurity.

Siemens’ products and solutions undergo continuous development to make them more secure. Siemens strongly recommends to apply product updates as soon as available and to always use the latest product versions. Use of product versions that are no longer supported, and failure to apply latest updates may increase customer’s exposure to cyber threats. To stay informed about product updates, subscribe to the Siemens Industrial Security RSS Feed under http://www.siemens.com/industrialsecurity.
## Table of Contents

### Warranty and Liability

1. Preface ................. 2
2. Hardware and Software Components Used ................. 5
3. Basics of Standard Web Pages ................. 6
4. Basics of User-Defined Web Pages ................. 8
5. Reading and Writing with different Variable Types ................. 9
   5.1 Automation task ................. 10
   5.2 Functional mechanisms and use ................. 12
   5.2.1 S7 program structure ................. 13
   5.2.2 User-defined web page (HTML file) structure ................. 19
6. Displaying the Date and Time ................. 15
   6.1 Automation task ................. 15
   6.2 Functional mechanisms and use ................. 17
   6.2.1 S7 program structure ................. 18
   6.2.2 User-defined web page (HTML file) structure ................. 20
7. Displaying an S7 Array (S7-1500 only) ................. 23
   7.1 Automation task ................. 23
   7.2 Functional mechanisms and use ................. 24
   7.2.1 S7 program structure ................. 25
   7.2.2 User-defined web page (HTML file) structure ................. 26
8. ENUM – Variable Value Replaced with Text ................. 27
   8.1 Automation task ................. 27
   8.2 Functional mechanisms and use ................. 28
   8.2.1 S7 program structure ................. 29
   8.2.2 User-defined web page (HTML file) structure ................. 30
9. HTTP Redirection following an Error (S7-1500 only) ................. 32
   9.1 Automation task ................. 32
   9.2 Functional mechanisms and use ................. 33
   9.2.1 S7 program structure ................. 34
   9.2.2 User-defined web page (HTML file) structure ................. 34
10. Change Language on User-Defined Pages ................. 36
   10.1 Automation task ................. 36
   10.2 Functional mechanisms and use ................. 37
   10.2.1 S7 program structure ................. 38
   10.2.2 User-defined web page structure ................. 39
   10.3 Extending the example ................. 40
11. Transferring Data without Reloading Pages Using AJAX ................. 42
   11.1 Automation task ................. 42
   11.2 Functional mechanisms and use ................. 43
   11.2.1 S7 program structure ................. 44
   11.2.2 User-defined web page structure ................. 45

© Siemens AG 2017 All rights reserved

Examples for the Web Server
Entry ID: 68011496, V2.0, 04/2017
**Table of Contents**

12  **Displaying Data of a Datalog as a Graph** ............................................. 47  
   12.1  Automation task.................................................................................. 47  
   12.2  Functional mechanisms and use....................................................... 48  
   12.2.1  S7 program structure................................................................. 49  
   12.2.2  User-defined web page structure .............................................. 50  
13  **Display Elements Using Scalable Vector Graphics** .................................... 54  
   13.1  Programming task ........................................................................... 54  
   13.2  Functional mechanisms and use....................................................... 54  
   13.2.1  Sample SVG element – rotary motion of a motor ...................... 55  
14  **S7 Diagnostics and Loading Indicator** ...................................................... 58  
   14.1  Automation task.................................................................................. 58  
   14.2  Functional mechanisms and use....................................................... 58  
   14.2.1  S7 program structure................................................................. 59  
   14.2.2  User-defined web page structure .............................................. 59  
15  **Changing Pictures Using JavaScript** .......................................................... 64  
   15.1  Programming task........................................................................... 64  
   15.2  Functional mechanisms and use....................................................... 64  
16  **Button for CPU Restart** ............................................................................ 66  
   16.1  Automation task.................................................................................. 66  
   16.2  Functional mechanisms and use....................................................... 66  
   16.2.1  S7 program structure................................................................. 67  
   16.2.2  User-defined web page structure .............................................. 67  
17  **Login on User-Defined Pages** .................................................................... 69  
   17.1  Automation task.................................................................................. 69  
   17.2  Functional mechanisms and use....................................................... 69  
   17.2.1  S7 program structure................................................................. 70  
   17.2.2  User-defined page structure...................................................... 70  
18  **High-Performance Communication via a String** ....................................... 73  
   18.1  Automation task.................................................................................. 73  
   18.2  Functional mechanisms and use....................................................... 74  
   18.2.1  S7 program structure................................................................. 75  
   18.2.2  User-defined web page structure .............................................. 78  
19  **Installation** .............................................................................................. 82  
   19.1  Installing the hardware and software ............................................. 82  
   19.2  Installing the application example .................................................. 83  
20  **Internet links** ........................................................................................... 84  
21  **History** .................................................................................................... 84
1  Preface

Objective of the application examples

The application examples in this document show you how to extend your own web pages on the S7-1200/1500 web server. This document describes the examples provided for downloading. It is recommended to run the examples on a computer and simultaneously analyze them with the aid of this document.

Main contents of the application examples

The application examples cover the following key points:

- Reading and writing with different variable types
- Displaying the time
- Outputting arrays (S7-1500 only)
- Using the ENUM data type
- HTTP redirection following an error (S7-1500 only)
- Change language on user-defined pages
- Transferring data without reloading pages
- Displaying a datalog generated by the controller as a graph
- Creating display elements
- Displaying diagnostic information
- Changing pictures using JavaScript
- Restarting the CPU using a button
- Logging in on the user-defined page
- High-performance data transfer

Advantages

Integrated web server in the S7-1200 and S7-1500

The standard web pages for easy display of service and diagnostic information are enabled with a single click.

In addition, you can create custom, user-defined web pages that are referred to below as "user-defined pages".

Access from anywhere

A web browser allows you to access the S7 web pages from anywhere in the world.

Application example

Universal use of the application example for the SIMATIC S7-1200 and S7-1500.

Benefits

No additional hardware and software required. The web server can be accessed over large distances using mobile communications devices such as tablet computers, smartphones, etc.

Note

The application examples in conjunction with the web server should not and cannot replace an HMI system.
2 Hardware and Software Components Used

The application examples were created with the following components.

**Hardware components**

Table 2-1: Hardware components used for the examples

<table>
<thead>
<tr>
<th>Component</th>
<th>No.</th>
<th>Order no.</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU 1511-1 PN</td>
<td>1</td>
<td>6ES7511-1AK01-0AB0</td>
<td>Firmware V2.0.5</td>
</tr>
<tr>
<td>CPU 1214C DC/DC/DC</td>
<td>1</td>
<td>6ES7214-1AG40-0XB0</td>
<td>Firmware V4.2</td>
</tr>
<tr>
<td>PG/PC with Ethernet interface</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>IE FC TP STANDARD CABLE</td>
<td>1</td>
<td>6XV1840-2AH10</td>
<td>IE connecting cable, minimum order quantity: 20m</td>
</tr>
<tr>
<td>RJ45 plug connector</td>
<td>2</td>
<td>6GK1901-1BB10-2AA0</td>
<td>Can be moulded</td>
</tr>
</tbody>
</table>

**Note**

For these application examples, you need the current CPU firmware version. Depending on the CPU type, the following entries provide related links to the appropriate downloads:


**Software components**

Table 2-2: Software components used for the examples

<table>
<thead>
<tr>
<th>Component</th>
<th>No.</th>
<th>Order no.</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIMATIC STEP 7 Professional V14</td>
<td>1</td>
<td>6ES7822-1AA04-0YE5</td>
<td>V14.0 (Update 2)</td>
</tr>
<tr>
<td>Software tool for creating HTML files such as FrontPage, Notepad++, …</td>
<td>1</td>
<td>-</td>
<td>Web pages created with Notepad++</td>
</tr>
<tr>
<td>Web browser such as Internet Explorer, Mozilla Firefox 1)</td>
<td>1</td>
<td>-</td>
<td>Created and tested with IE11 and Firefox</td>
</tr>
</tbody>
</table>

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

- Internet Explorer (version 11)
- Mozilla Firefox (version 50)

**Note**

The application examples are optimized for Firefox and IE11. Using other browsers may require changes regarding their display in the browser.
Overall structure

The individual application examples consist of an S7 program running on an S7-1200/1500 and a web page running on the integrated web server of the S7. Exceptions are the examples in Chapter 13 and 15 that show general web functions without an S7 program. The web pages are opened using a web browser and consist of an HTML file as a basis. More complex examples include additional JavaScript files with the “js” file extension.

Figure 2-1: Hardware configuration for the application examples

Sample files and projects

The following list contains the files required for this example, consisting of the S7 project and this PDF document.

Table 2-3: Examples – source files

<table>
<thead>
<tr>
<th>Component</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>68011496_examples_for_S7WebServer_CODE_v20.zip</td>
<td>The file contains the STEP 7 project with the appropriate HTML files in the \html directory.</td>
</tr>
<tr>
<td>68011496_examples_for_S7WebServer_en_DOC_v20.pdf</td>
<td>This document.</td>
</tr>
</tbody>
</table>
3 Basics of Standard Web Pages

Requirements
In STEP 7, the following settings are required in the CPU properties.
- The web server must be activated.
- If you require secure access to the web pages, check the “Permit access only with HTTPS” check box.
- Automatic refresh of web pages is enabled.
  The default refresh interval is 10 seconds.
  The interval can be set in the range from 1 to 999s.

Access via HTTP or HTTPS
The URL “http://ww.xx.yy.zz” or “https://ww.xx.yy.zz” provides you with access to
the standard web pages. “ww.xx.yy.zz” corresponds to the CPU’s IP address.
HTTPS is used for encrypting and authenticating communication between the
browser and the web server. When the “Permit access only with HTTPS” check
box is checked, the CPU’s web pages can only be opened via HTTPS.

CPU certificate
If the browser reports a certificate error, proceed as described in the following FAQ:

Access rights for login to the web pages in STEP 7
By factory default, the “Everybody” user is set in each SIMATIC S7-1200/1500
controller.
This user has limited access rights and no password.
(This prevents access to user-defined pages.)
To have full access to user-defined pages, you have to log in with a user who has
the required access rights. You can parameterize users, passwords and access
rights with STEP 7 in the S7-1200/1500 controller properties.

The login input fields are located in the top left corner of each standard web page
of the S7-1200/1500 controller.

Figure 3-1: Login window on standard web page

SIMATIC S7-1200 / S7-1500 standard web pages
Via the integrated standard web page, the S7-1200 and S7-1500 web server
already offers a large amount of information about the respective CPU.
The structure of the standard web pages is described in detail in the S7-1500 Web
Server Function Manual; it is not the subject of this document.
4 Basics of User-Defined Web Pages

For the basics of user-defined pages, please refer to this application example: “Creating and using user-defined web pages on S7-1200 / S7-1500”

4.1 Login and instructions for use

In TIA Portal, each example name starts with the chapter number:
Example: Chapter 5 corresponds to program name “05ReadAndWriteS7...”
The web contents for the examples can be found in the appropriate S7 project on your computer in the “…\html_...” folder.

Login of the user-defined pages in the application examples
User: admin
Password: s7

4.2 Web server – initializing the WWW system function

In the application examples, the “WWW” system function (SFC 99) is called by OB 1.
This block call is described in all examples, including the reference to this chapter.

<table>
<thead>
<tr>
<th>No.</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The “WWW” system function (SFC 99) is called in OB1. This function initializes the CPU web server.</td>
</tr>
</tbody>
</table>

Cyclically calling the function in OB1 allows interaction between the web server and the controller.

The Web DB (DB 333) is connected to the WWW function. The Web DB and the Fragment DB(s) store the structure of the user-defined pages.

| 2   | More instructions and block calls are described in the respective application examples. |

Note

For basic information about generating data blocks for user-defined pages, please refer to the “First steps” of the following entry:
4.3 Libraries – jQuery and S7 Framework

The web pages of the application examples consist of HTML documents, JavaScript files and JavaScript libraries.

The file structures can be found in the respective application examples.

Some application examples use the "jQuery" and S7 Framework JavaScript libraries. These examples are listed in the following table.

Table 4-1: Libraries used in the application examples

<table>
<thead>
<tr>
<th>Chapter no.</th>
<th>Chapter name</th>
<th>jQuery</th>
<th>S7 Framework</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Change Language on User-Defined Page</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Displaying Data of a Datalog as a Graph</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>14</td>
<td>S7 Diagnostics and Loading Indicator</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>16</td>
<td>Button for CPU Restart</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>17</td>
<td>Login on User-Defined Page</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>18</td>
<td>High-Performance Communication via a String</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Integrate the jQuery library and the appropriate JavaScript files that access or use this jQuery library into your HTML base document. For example, the “S7 Framework” JavaScript file uses the jQuery library; in the HTML code, it is therefore below the call of the jQuery library.

Figure 4-1: Call of JavaScript libraries in the HTML document

```html
<!--head-->
<meta http-equiv="content-type" content="text/html; charset=UTF-8">
<script src="script/jquery-2.1.3.min.js"></script>
<script src="script/S7_framework_0.1.7.js"></script>
<!--/head-->
```

**jquery-2.1.3.min.js (jQuery library)**

The “jquery-2.1.3.min.js” file is a library that enables more efficient programming with JavaScript. The library contains basic DOM, event, effects and Ajax functions that are not described in detail in this document. The library’s methods are typically accessed by the prefixed “jQuery” object identifier. This object identifier is replaced with a "$" sign, which reduces the total number of characters in a JavaScript file.

Example:

```javascript
jQuery.post(URL, DATA)

$.post(URL, DATA)
```

For general information about jQuery, visit: [https://jquery.com/](https://jquery.com/)
S7_framework.js

The JavaScript file contains a number of functions for converting and processing data from a SIMATIC S7-1200/1500.

The S7 Framework programming uses elements from the jQuery library. Therefore, in the HTML code, always call the S7 Framework after the jQuery library.

There are separate application examples for the S7-1200 and the S7-1500 controller. Both controllers access the same web files.

However, depending on the CPU, access to the default web server differs in terms of IDs, login and other functions. A controller type check is therefore implemented in the examples.

The following table lists the S7 Framework functions.

<table>
<thead>
<tr>
<th>No.</th>
<th>Code description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Assignment of variable types within S7 Framework</td>
</tr>
<tr>
<td></td>
<td>var S7Framework = (function($, undefined) {</td>
</tr>
<tr>
<td></td>
<td>// type of variable - 0=Bool, 1=unsigned INT, 2=signed INT, 3=real, 4=REAL, 5=String</td>
</tr>
<tr>
<td></td>
<td>var BOOL = 0, UINT = 1, INT = 2, REAL = 3, REAL = 4, STRING = 5;</td>
</tr>
<tr>
<td>2</td>
<td>PLC functions (different tasks as a library in the framework)</td>
</tr>
<tr>
<td></td>
<td>- Functions and variables for</td>
</tr>
<tr>
<td></td>
<td>- AJAX (data transfer)</td>
</tr>
<tr>
<td></td>
<td>- JSON (data processing of strings using parser)</td>
</tr>
<tr>
<td></td>
<td>- CPU type (access to S7-1200/1500 web server)</td>
</tr>
<tr>
<td></td>
<td>- Logon/logoff (referenced to S7 standard web page, “Portal.mwsl”)</td>
</tr>
<tr>
<td></td>
<td>- Loading indicator (definition and function for animated loading icon)</td>
</tr>
<tr>
<td></td>
<td>- Error handling within S7 Framework data processing</td>
</tr>
<tr>
<td></td>
<td>function Controller () {</td>
</tr>
</tbody>
</table>

**Note**

In the code, you will find information about the functions. A detailed description of the functions is not the subject of this guide.
5 Reading and Writing with different Variable Types

5.1 Automation task

The task is to read and rewrite variables of different data types.

Note

The DTL data type is only supported by the S7-1500.

Requirements for the automation task

Two HTML pages have to be programmed:

- One HTML page that allows you to read/write variables of different types.
- One HTML page that allows you to read special variables.
5.2 Functional mechanisms and use

Overall structure

The application example consists of an S7 program and two HTML documents that are displayed as web pages in a browser.

User-defined page structure

The following figure shows the user-defined page for displaying simple PLC tags that are read/write accessed.
The top “Simple Variables” access S7 flags, whereas the “Structure and Array” variables access a “DB 1” data block.

Figure 5-1: User-defined page for reading/writing simple variables

Table 5-1: Web page for reading/writing simple variables

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>This link takes you to the HTML web page with the special variables, “Reading special variables”.</td>
</tr>
<tr>
<td>2</td>
<td>This column displays the current values of the variables.</td>
</tr>
<tr>
<td>3</td>
<td>The value to be written is entered in this column.</td>
</tr>
<tr>
<td>4</td>
<td>The “Write” buttons transfer the values to the CPU.</td>
</tr>
</tbody>
</table>

Table 5-2: Instructions for using the user-defined page

<table>
<thead>
<tr>
<th>Step</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In the “New Value” column, enter a value corresponding to the data type.</td>
</tr>
<tr>
<td>2</td>
<td>Click the “Write” button.</td>
</tr>
<tr>
<td>3</td>
<td>The value appears in the “Current Value” column.</td>
</tr>
<tr>
<td>4</td>
<td>Click this link if you want to open the “Reading special variables” web page.</td>
</tr>
</tbody>
</table>
The following figure shows the structure of the user-defined page for reading special variables.

Figure 5-2: User-defined page for reading/writing special variables

Table 5-3: Web page for reading/writing special variables

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>This link takes you to the web page for reading/writing PLC tags.</td>
</tr>
<tr>
<td>2</td>
<td>This column displays the names of the special variables.</td>
</tr>
<tr>
<td>3</td>
<td>This column displays the information (value) of the respective special variable.</td>
</tr>
</tbody>
</table>
5 Reading and Writing with different Variable Types

5.2.1 S7 program structure

The diagrammatic representation below shows the S7 program structure. The HTML web page is called cyclically. The following sections describe the data exchange with the blocks.

Figure 5-3: S7 program for reading/writing variables

How OB 1 works

<table>
<thead>
<tr>
<th>No.</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The first block call in OB 1 is the WWW function (SFC 99). For more information about the function, please refer to Chapter 4.2.</td>
</tr>
</tbody>
</table>

Contents of the PLC tags

The top section of the first HTML page defines simple PLC tags (flags) that can be read/written using the web page.

Contents of DB1

The “TestDB” (DB1) defines variables that can be read/written using the web page. Exception: the “WWW_RET_VAL” variable. This variable contains the return value of the WWW function.

Contents of DB2

DB2 “HTTP_Info” stores the special variables; in this case: HEADER_Request variables.

The user can only read these variables using the web page. These variables are written using the web page via the web server.
5.2.2 User-defined web page (HTML file) structure

<table>
<thead>
<tr>
<th>No.</th>
<th>Code section</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>To read/write a PLC tag from the PLC tags (tag table) of a web page, only the &quot;variable name&quot; in the syntax is relevant. General :=&quot;variable name&quot;:</td>
</tr>
<tr>
<td></td>
<td>Example :=&quot;testBit&quot;:</td>
</tr>
</tbody>
</table>

**Default tag table**

<table>
<thead>
<tr>
<th>No.</th>
<th>Name</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>testBit</td>
<td>Bool</td>
</tr>
<tr>
<td>2</td>
<td>testByte</td>
<td>Byte</td>
</tr>
<tr>
<td>3</td>
<td>testWord</td>
<td>Word</td>
</tr>
<tr>
<td>4</td>
<td>testInt</td>
<td>Int</td>
</tr>
</tbody>
</table>

```html
<!-- AMP_In_Variable Name="testBit" -->
<form method="post" action="">

<td>BOOL</td><td>"testBit"</td><td>:="testBit":</td>
<td><input type="text" name="testBit" maxlenght="8"></td>
<td><input type="submit" value="Write"></td>
</form>
```


To read/write a variable from a DB, the DB name and the variable name are relevant.

### NormalVariables

<table>
<thead>
<tr>
<th>No.</th>
<th>Code section</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Static</td>
</tr>
<tr>
<td>2</td>
<td>testBit</td>
</tr>
<tr>
<td>3</td>
<td>testWord</td>
</tr>
<tr>
<td>4</td>
<td>testInt</td>
</tr>
<tr>
<td>5</td>
<td>testWordArray</td>
</tr>
<tr>
<td>6</td>
<td>testBitArray</td>
</tr>
<tr>
<td>7</td>
<td>testChar</td>
</tr>
<tr>
<td>8</td>
<td>testByte</td>
</tr>
<tr>
<td>9</td>
<td>testDWord</td>
</tr>
<tr>
<td>10</td>
<td>testDInt</td>
</tr>
<tr>
<td>11</td>
<td>testReal</td>
</tr>
<tr>
<td>12</td>
<td>testString</td>
</tr>
<tr>
<td>13</td>
<td>testTime</td>
</tr>
<tr>
<td>14</td>
<td>testDate</td>
</tr>
<tr>
<td>15</td>
<td>testToD</td>
</tr>
<tr>
<td>16</td>
<td>testDT</td>
</tr>
</tbody>
</table>

```xml
<!-- AXP_In_Variable Name="NormalVariables".testBit -->
<tr>
  <form method="post" action="">
    <td>BOOL</td>
    <td>:="NormalVariables".testBit</td>
  </form>
</tr>
```

```xml
<input type="text" name="NormalVariables".testBit" maxlengh="8"/>
```

```xml
<input type="submit" value="Write">
```

</tr>
The special variables (here: HTTP request variables) are saved in the DB variables with UDP commands. This DB variable is read by the HTML file.

Note:
http request variables are global variables within a script's scope.
6 Displaying the Date and Time

6.1 Automation task

The task is to display the CPU's time on a web page.

Requirements for the automation task

- Read the time in the STEP 7 program using the “RD_LOC_T” function.
- Program a web page that displays the time.
- Update the time with the aid of a second HTML file (inline frame) integrated in the HTML file.
  For a description of the term ‘inline frame’, please refer to Chapter 11.

6.2 Functional mechanisms and use

The application example consists of an S7 program and two HTML documents that are displayed as web pages in a browser.

User-defined page structure

The date and time are displayed horizontally and centered on the user-defined page.

Figure 6-1: User-defined page for displaying the date and time

13:18 16.2.2017

Setting the time in the controller

The first step is to set the current time in the data source, i.e., in the CPU.

1. Open TIA Portal and the program example.
2. In the Project tree, right-click your controller and select “Go online”.
3. Select “Online & diagnostics”.
4. In Functions, select “Set time”.
5. In Properties, select the time zone and, if necessary, standard time/daylight saving time.
### 6.2.1 S7 program structure

The diagrammatic representation below shows the S7 program structure. The HTML user-defined page is called cyclically. The following sections describe the data exchange with the blocks.

**Figure 6-2: S7 program for displaying the date and time**

<table>
<thead>
<tr>
<th>No.</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The first block call in OB 1 is the WWW function (SFC 99). For more information about the function, please refer to Chapter 4.2.</td>
</tr>
<tr>
<td>2</td>
<td>The “RD_LOC_T” instruction (SFC 154) reads the CPU’s time and saves it to the “Clock” DB (DB 1) to the “time” variable.</td>
</tr>
</tbody>
</table>

**Contents of the Clock DB (DB 1)**

This DB stores the time in “DTL” format. The user-defined page reads the time from this DB.
### 6.2.2 User-defined web page (HTML file) structure

<table>
<thead>
<tr>
<th>No.</th>
<th>Code section</th>
</tr>
</thead>
</table>
| 1   | The UDP commands initialize the variables.  
     | `<!-- AWP_In_Variable Name="Clock".time.YEAR -->`
     | `<!-- AWP_In_Variable Name="Clock".time.MONTH -->`
     | `<!-- AWP_In_Variable Name="Clock".time.DAY -->`
     | `<!-- AWP_In_Variable Name="Clock".time.HOUR -->`
     | `<!-- AWP_In_Variable Name="Clock".time.MINUTE -->`
     | `<!-- AWP_In_Variable Name="Clock".time.SECOND -->`
| 2   | The inline frame ("Update_Page.html" file) writes the time to this div box.  
     | `<div id="clock">
     |     ...
     |     </div>`
| 3   | The inline frame must be integrated into the “index.html” file in order to integrate it into the functional sequence.  
     | `<iframe src="Update_Page.html" style="display:none;"/>`
| 4   | The "Update_Page.html" file specifies which variables are cyclically updated and where to store them.  
     | (This is implemented using the ID in the JavaScript file and in the HTML code.)  
     | `/* Function for updating the variables */`
     | `function updateSingleVariablesTable()`  
     | `{ var table = document.getElementById("singleVariablesTable");`  
     | `for(i = 1; i < table.rows.length; i++)
     |     { hour = table.rows[i].cells[0].innerHTML;
     |     } if (hour < 10)`
     | `{ hour = "0" + hour;`  
     | `} if (min < 10)`  
     | `{ minute = "0" + minute;`  
     | `} document.write('hour = "' + hour + ";
     | `document.write('minute = "' + minute + ";
     | `document.write('hour = "' + min + ";
     | `document.write('day = "' + day + ");
     | `document.write('month = "' + month + ");
     | `document.write('year = "' + year + ");
     | `}
     | `}`
     | `}</script>`  
|     | `<td>![](read of date and time -->`  
|     | `<script type="text/javascript">
     |     var hour = "Clock".time.HOUR;
     |     var minute = "Clock".time.MINUTE;
     |     if (hour < 10)
     |     {
     |     hour = "0" + hour;
     |     }
     |     if (minute < 10)
     |     {
     |     minute = "0" + minute;
     |     }
     |     document.write('hour = "' + hour + ";
     |     document.write('minute = "' + minute + ";
     |     document.write('day = "' + day + ");
     |     document.write('month = "' + month + ");
     |     document.write('year = "' + year + ");
     |     `</script>`
     |     `</td>`
|     | `>clock`  
|     | `</td>`    

Referenced to:  
`<table id="singleVariablesTable" border="1">`

The 'if' statement is prefixed for displaying a "0", e.g. for 12:03h.

For information about this update method, please refer to the following FAQ:  
7 Displaying an S7 Array (S7-1500 only)

7.1 Automation task

The task is to read all fields of an S7 array and output them on a user-defined page as a table.

Requirements for the automation task

- Create an array in a DB of the STEP 7 program.
- Display the array fields in a table on a user-defined page.

7.2 Functional mechanisms and use

User-defined page structure

The user-defined page displays the array values structured in a table. The values in the array are updated by refreshing the user-defined page using “F5”.

Figure 7-1: User-defined page for displaying S7 arrays

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>This column displays the array index</td>
</tr>
<tr>
<td>2</td>
<td>This column displays the array field value</td>
</tr>
</tbody>
</table>

Table 7-1: User-defined page for displaying S7 arrays
### 7.2.1 S7 program structure

The diagrammatic representation below shows the S7 program structure. The change of array values is controlled using the DB1 online view and the user-defined page is refreshed as described in Chapter 7.2.

Figure 7-2: S7 program for displaying S7 arrays

#### How OB 1 works

<table>
<thead>
<tr>
<th>No.</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The first block call in OB 1 is the WWW function (SFC 99). For more information about the function, please refer to Chapter 4.2.</td>
</tr>
</tbody>
</table>

#### Contents of the Data DB (DB 1)

The Data DB1 defines the array that is read by the user-defined page.

### 7.2.2 User-defined web page (HTML file) structure

The array contains the variables displayed on the user-defined page.

<table>
<thead>
<tr>
<th>No.</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Adapt the UDP command to the respective array name (DB).</td>
</tr>
</tbody>
</table>

```html
<table border="1" width="120px">
<tr><td>Index</td><td>Value</td></tr>
<tr><td>ArrayName="Data.values" --></td><td>ArrayIndex="" ; Value :</td></tr>
<tr><td>- AMP_End_Array --></td></tr>
</table>
```
8 ENUM – Variable Value Replaced with Text

8.1 Automation task

The task is to change the integer value of a variable using different buttons on a user-defined page.

The values that are written to the variable are to be linked with different message texts.

A text display is to display the appropriate message texts.

Requirements for the automation task

- Read/write a variable defined as ENUM with a user-defined page.
- Define an ENUM variable.
- Create a variable in a PLC tag table.

Note

ENUM variables are defined in an HTML file. In ENUM variables, numerical values are replaced with a string. The S7 program uses a numerical variable.
8.2 Functional mechanisms and use

User-defined page structure

Figure 8-1: ENUM user-defined page – variable values are replaced with text

Table 8-1 ENUM user-defined page – variable values are replaced with text

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>These buttons allow you to set the “alarm” variable to a value between 0 and 3.</td>
</tr>
<tr>
<td>2</td>
<td>This table shows the assignment of the text to the numerical value.</td>
</tr>
<tr>
<td>3</td>
<td>This is where the appropriate text is output.</td>
</tr>
</tbody>
</table>

Table 8-2: Instructions for using the user-defined page

<table>
<thead>
<tr>
<th>Step</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Select one of the four possible buttons, in this case: “alarm=0”.</td>
</tr>
<tr>
<td>2</td>
<td>The “No Alarm! all okay.” text is output based on the numerical assignment, “alarm=0”.</td>
</tr>
</tbody>
</table>
8.2.1 S7 program structure

The diagrammatic representation below shows the S7 program structure. The value of the “alarm” variable in DB1 is set by the field selection of the user-defined page.

Figure 8-2: ENUM S7 program – variable values are replaced with text

How OB 1 works

<table>
<thead>
<tr>
<th>No.</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The first block call in OB 1 is the WWW function (SFC 99). For more information about the function, please refer to Chapter 4.2.</td>
</tr>
</tbody>
</table>

Contents of the AlarmVariable DB (DB 1)

<table>
<thead>
<tr>
<th>No.</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>This DB stores the “alarm” variable that can be read/written by the user-defined page. The variable is modified by the selection of the user-defined page (here: “alarm=0“)</td>
</tr>
</tbody>
</table>

8.2.2 User-defined web page (HTML file) structure

<table>
<thead>
<tr>
<th>No.</th>
<th>Code section</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Definition of the ENUM variable in the &quot;.enumdefs.htm“ file</td>
</tr>
<tr>
<td>2</td>
<td>Initialization of the ENUM variable</td>
</tr>
<tr>
<td>3</td>
<td>The value of the PLC tag is read and displayed as text.</td>
</tr>
<tr>
<td>4</td>
<td>The assigned text is used when writing the PLC tag. The associated value (digit) is written to the PLC tag.</td>
</tr>
</tbody>
</table>
9 HTTP Redirection following an Error (S7-1500 only)

9.1 Automation task

If a variable has exceeded a certain value, a new user-defined page is to open displaying an appropriate message. The value of the variable is to be settable using an input field on the user-defined page.

Requirements for the automation task
- Program a user-defined page with an input field for setting a variable value.
- The ENUM variable is specified by the STEP 7 program.
- An ENUM variable is to control the URL (Uniform Resource Locator).
9.2 Functional mechanisms and use

User-defined page structure

Figure 9-1: User-defined page for HTTP redirection following an error

Table 9-1: User-defined page for HTTP redirection following an error

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In this field, enter a new input value.</td>
</tr>
<tr>
<td>2</td>
<td>The “Check the value” button sends the value to the CPU.</td>
</tr>
<tr>
<td>3</td>
<td>If the input value is greater than “100”, a new user-defined page opens. In this example, the new user-defined page has red text.</td>
</tr>
</tbody>
</table>

Table 9-2: Instructions for using the user-defined page

<table>
<thead>
<tr>
<th>Step</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In the input field, first enter a value less than “100”.</td>
</tr>
<tr>
<td>2</td>
<td>Select the button and repeat the step with “101”.</td>
</tr>
<tr>
<td>3</td>
<td>The new value is displayed and the new user-defined page opens if the input value &gt; “100”.</td>
</tr>
</tbody>
</table>

Note

If the checked variable is influenced by the controller, the user-defined page has to be cyclically refreshed. Otherwise, no HTTP redirect will take place.

To go to the home page, set the variable value in TIA Portal to less than “100” and reload the user-defined page.
9 HTTP Redirection following an Error (S7-1500 only)

9.2.1 S7 program structure

The diagrammatic representation below shows the S7 program structure. By comparing the values in FC1, the system switches to the user-defined page.

Figure 9-2: S7 program for HTTP redirection following an error

How OB 1 works

<table>
<thead>
<tr>
<th>No.</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The first block call in OB 1 is the WWW function (SFC 99). For more information about the function, please refer to Chapter 4.2.</td>
</tr>
<tr>
<td>2</td>
<td>The “Check” function determines whether the “100” limit was exceeded. If this is the case, the “headerLocation” variable is set to “1” and “headerStatus” is set to “302”. Due to this, the system switches to the user-defined page with the error message. If the value was not exceeded, both variables are set to “0”.</td>
</tr>
</tbody>
</table>

Contents of the Data DB (DB 1)

<table>
<thead>
<tr>
<th>No.</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DB1 stores the variables for FC 1. If the value of the “testword” variable &gt; 100, the system switches the user-defined page.</td>
</tr>
</tbody>
</table>
### 9.2.2 User-defined web page (HTML file) structure

<table>
<thead>
<tr>
<th>No.</th>
<th>Code section</th>
</tr>
</thead>
</table>
| 1   | To change the location, the appropriate URL must be assigned to the numbers stored in the "headerLocation" variable.  

An ENUM variable must be defined for this purpose.  

```html
<!-- AWP_Enum_Def Name="redirEnum" Values='
1:"
2:Demo09_redir.html",
0:"
```  
| 2   | Initialization of the CPU variables and assignment of the ENUM variable.  

```html
<!-- AWP_In_Variable Name="Data.testword" -->  
<!-- AWP_Out_Variable Name="HEADER:Status"
Use="Data.headerStatus" -->  
<!-- AWP_Out_Variable Name="HEADER:Location"
Use="Data.headerLocation" Enum="redirEnum" -->
```  
| 3   | Description of the variable to be checked.  

```html
<form method="POST" action="">
  <input type="text" name="Data.testword" size="10">
  <input type="submit" value="Check the value" >  
</form>
```
10 Change Language on User-Defined Pages

10.1 Automation task

The task is to implement a change language option (German/English) with flag icons on the user-defined pages and add custom languages to this option.

Requirements for the automation task

- Program a user-defined page without texts.
- Create a text file with all texts in German and English.
- Change the language using the country’s flag.
- Translate texts selected by JavaScript.
- Synchronize the change language option of the user-defined pages with the change language option of the standard web pages.
- Expansion capability of the change language option.

10.2 Functional mechanisms and use

User-defined page structure

On the user-defined page, you can change between German and English. To do this, click the appropriate flag icon.

Figure 10-1: User-defined page for change language
10.2.1 S7 program structure

In the S7 program, only the WWW function is called.

Figure 10-2: S7 program for change language

How OB 1 works

<table>
<thead>
<tr>
<th>No.</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The first block call in OB 1 is the WWW function (SFC 99). For more information about the function, please refer to Chapter 4.2.</td>
</tr>
</tbody>
</table>

10.2.2 User-defined web page structure

The diagrammatic overview below shows the basic file structure of the user-defined pages.

The term “i18n” that recurs in the following sections stands for a software development process for implementing the change language option independently of the actual program code. The term “i18n” results from internationalization, where 18 stands for the number of letters between “i” and “n”.

For efficient programming of the “Demo10.js” script code and compatible programming of the function of the user-defined pages across browsers, the jQuery library for JavaScript is integrated.

Figure 10-3: Diagrammatic overview of the user-defined web page
10 Change Language on User-Defined Pages

Demo10.html
The HTML document defines the structure of the user-defined page and the calls of the JavaScript files used, see Chapter 10.2.2.1. Throughout the file structure, the language assignment is ensured by the “de” and “en” identifiers.

Demo10.js
Contains the interactive functions used to implement the user inputs, see Chapter 10.2.2.2. Also defines the behavior of the cookie used. The JavaScript file’s code syntax is supported by the jQuery library.

i18n.txt
The “i18n.txt” text file contains the multilingual text elements, “Demo10.html” and the script functions that are called from “i18n:0.0.1.js”.

i18n.0.0.1.js
Contains the functions for changing the language and transferring the texts with the specific language from “i18n.txt” to the HTML document.

Figure 10-4: Code element for change language
```javascript
// replace text of all elements with data-118n attribute
function translateLabels() {
  $('[data-118n]').each(function()
    // Load language definition file to var dictionary
    function load()
      var filename = "script/i18n.txt";
      $.ajax({
        type: "text", url: url, data: ",", datatype: "text"})
```

jquery.cookie.js
The JavaScript file provides functions for defining and managing cookies. In this context, the “Demo10.js” JavaScript file uses these functions and creates or modifies the existing cookie of the standard web page, see Chapter 10.2.2.2. For more information about the JavaScript file used here, visit the following web page:
- [https://github.com/carhartl/jquery-cookie](https://github.com/carhartl/jquery-cookie)
  - [https://github.com/carhartl/jquery-cookie/blob/972bbd507d296e30400ebf61aa3ace1d9076312419543/src/jquery.cookie.js](https://github.com/carhartl/jquery-cookie/blob/972bbd507d296e30400ebf61aa3ace1d9076312419543/src/jquery.cookie.js)
  - [https://github.com/js-cookie/js-cookie/tree/master/](https://github.com/js-cookie/js-cookie/tree/master/)

jquery-2.1.3.min.js (jQuery library)
The jQuery library file allows more efficient programming with JavaScript. For more information about the jQuery library, please refer to Chapter 4.3.
10.2.2.1 Demo10.html structure

<table>
<thead>
<tr>
<th>No.</th>
<th>Code section</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The JavaScript files used are integrated into the HTML document’s <code>&lt;head&gt;</code> with the below syntax:</td>
</tr>
<tr>
<td></td>
<td><code>&lt;head&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;title data-lang=&quot;title&quot;&gt;&lt;/title&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;meta http-equiv=&quot;content-type&quot; content=&quot;text/html; charset=UTF-8&quot;&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;!-- script type=&quot;text/javascript&quot; src=&quot;/file.js&quot;&gt;&lt;/script--&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;script type=&quot;text/javascript&quot; src=&quot;/script/jquery-2.1.3.min.js&quot;&gt;&lt;/script&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;script type=&quot;text/javascript&quot; src=&quot;/script/jquery.cookie.js&quot;&gt;&lt;/script&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;script type=&quot;text/javascript&quot; src=&quot;/script/l10n.js&quot;&gt;&lt;/script&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;script type=&quot;text/javascript&quot; src=&quot;/script/demo10.js&quot;&gt;&lt;/script&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/head&gt;</code></td>
</tr>
<tr>
<td></td>
<td>The “src” property specifies the respective file path.</td>
</tr>
<tr>
<td>2</td>
<td>The web page header is not integrated into the change language option.</td>
</tr>
<tr>
<td></td>
<td>It consists of the two structuring classes, “demohead” and “demorow”, and the two below text elements with an appropriate formatting instruction.</td>
</tr>
<tr>
<td></td>
<td><code>&lt;body&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;!-- Head for the Webpage --------------------------------------------------------------------- --&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;style&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>.demohead { background-color:#BECBED; width:100%}</code></td>
</tr>
<tr>
<td></td>
<td><code>.demorow {}</code></td>
</tr>
<tr>
<td></td>
<td><code>.demoooltitle { width:18px;font-weight:bold;vertical-align:top;}</code></td>
</tr>
<tr>
<td></td>
<td><code>.demoooltext {}</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/style&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;table class=&quot;demohead&quot; &gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;tr class=&quot;demorow&quot; &gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;td class=&quot;demoooltitle&quot; &gt;This page demonstrates: &lt;/td&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;td class=&quot;demoooltext&quot; &gt;How to realise a language switch. &lt;/td&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/tr&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/table&gt;</code></td>
</tr>
<tr>
<td>3</td>
<td>Changing the language using the flag icons is shown below.</td>
</tr>
<tr>
<td></td>
<td>The flag icons are linked to the “language” class.</td>
</tr>
<tr>
<td></td>
<td><code>&lt;span&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;img src=&quot;/images/flag-deutschland.png&quot;</code></td>
</tr>
<tr>
<td></td>
<td><code>class=&quot;language&quot; data-lang=&quot;de&quot; data-lang=&quot;lang.de&quot;</code></td>
</tr>
<tr>
<td></td>
<td><code>width=&quot;30px&quot; height=&quot;18px&quot; /&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/span&gt;</code></td>
</tr>
<tr>
<td></td>
<td>The “language” class with the (data-lang=&quot;...&quot;) property is assigned to the change language elements, here, e.g., “German”.</td>
</tr>
<tr>
<td></td>
<td>The (data-lang) property specifies the language the page is to change to.</td>
</tr>
<tr>
<td></td>
<td>Note: For other languages, the code section has the same structure.</td>
</tr>
<tr>
<td>No.</td>
<td>Code section</td>
</tr>
<tr>
<td>-----</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| 4   | In the same way as the text above the flag icons on the user-defined page, the following HTML text elements are changed between “de” and “en”.

**Textfield with HTML - (text.header in en)**

Simple text container structured with HTML syntax. - (text.content in de)

The “data-i18n” property refers to a uniquely identifiable name, e.g., “text.header” in the “i18n.txt” file.

```html
<div class="container" style="clear: left">
    <h1> <span data-i18n="text.header"> </span></h1>
    <span data-i18n="text.content"></span>
</div>
```

Reference from Demo10.html to “i18n.txt”

The “i18n.txt” file assigns the texts of the respective language to the identifying names such as (“text.header”). Therefore, “text field with HTML...” is displayed on the user-defined page instead of “text.header”.

```json
"text.header" : {
    "de": "Textfeld mit HTML - (text.header in de)",
    "en": "Textfield with HTML - (text.header in en)"
},
```
### 10.2.2.2 Demo10.js structure

<table>
<thead>
<tr>
<th>No.</th>
<th>Code section</th>
</tr>
</thead>
</table>
| 1   | The script first contains the variable declaration for the functions of the change language elements of the HTML page.  
    | \`\`var elements = {};\`\` |
| 2   | The following code section implements the assignment of the texts to the buttons and the output field.  
    | \$\text{\textinit} = \text{\text function}() { \  
    | \\  elements\text{\_language} = \$(\"\text{language}\\"); \  
    | \\  elements\text{\_button1} = \$(\"\#textByJavaButton1\\"); \  
    | \\  elements\text{\_button2} = \$(\"\#textByJavaButton2\\"); \  
    | \\  elements\text{\_output} = \$(\"\#textByJava\\"); \  
    | \  Left: elements.button1  
    | \  Right: elements.button2  
    | \  Bottom: elements.output |
| 3   | When the user clicks an element with (class="language"), the (language) property value from "data-lang" is transferred to the "I18n.setLanguage" function in the "i18n_0.0.1.js" script.  
    | Example:  
    | \  elements\text{\_language}.\text{\_click}(\text{\text function} () {  
    | \\  I18n.setLanguage( \$(\text{\_this}).\text{\_attr}(\"data-lang\"));  
    | \  At the same time, the language selection is saved in the "siemens_automation_language" cookie and synchronized with the standard web page.  
    | \  \$.\text{\_cookie}(\"siemens_automation_language\", \$(\text{\_this}).\text{\_attr}(\"data-lang\"));  
    | \  When the user-defined page is opened for the first time, the value of the "I18n.setLanguage" function is applied from the cookie.  
    | \  If there is no cookie, the language is set to "en" by default.  
    | \  \text{\_if}(\#\text{\_cookie}(\"siemens_automation_language\") === \text{\_null}) {  
    | \\  \#\text{\_cookie}(\"siemens_automation_language\", \"en\", [ \text{\_path} : '/']);  
    | \\  I18n.setLanguage(\#\text{\_cookie}(\"siemens_automation_language\"));  
    | 4   | When clicked, the function outputs the text in the current language based on the identifying name.  
    | The "data-i18n" property is set to the current ID, "java.output1", so that the text is translated the next time the language is changed.  
    | \  elements.button1.\text{\_click}(\text{\text function} () {  
    | \\  elements\text{\_output}(\text{\_html}((I18n.\text{\_translate}(\"java.output1\"))));  
    | \\  elements\text{\_output}.\text{\_attr}(\"data-i18n","java.output1\"));  
    | | 5   | Note:  
    | For more information about changing texts with JavaScript, please refer to the "Demo10.js" file's code.
10.3 Extending the example

Add a language to the example by duplicating and modifying code sections. Add the identifier for your desired language to the “de”/“en” identifiers – for example, “fr” for French or “xx” as a placeholder for any language.

Adding new flag icons

For the new language, you need an appropriate flag icon according to the example, in this case in PNG format. Copy the “xx.png” flag icon to the “../Images/” folder.

Extending Demo10.html

1. Extend the code section as shown in the figure.
2. In the “Images” path, provide a reference to the new flag icon (xx.png).
3. Change the text assignment for “data-i18n”.

Figure 10.5: Extension in the HTML code

```html
<img src="Images/flag-unitedkingdom.png" class="language" data-lang="en" data-i18n="lang.en" width="30px" height="18px" />
<img src="Images/xx.png" class="language" data-lang="xx" data-i18n="lang.xx" width="30px" height="18px" />
```

Extending i18n.txt

Add more languages to the “i18n.txt” text file. The text elements for “de” and “en” are assigned to the appropriate names (e.g., “lang.en”). As an example of these names, the following figure shows a code section that must be duplicated if you want to add a new language. Proceed in the same way to extend the text elements.

1. Duplicate the code section.
2. Add the new language item to the code section.
3. Change the name to assign the new code section, in this case “lang.xx”.

Figure 10.6: Extensions in the text file

```json
"lang.en": {
  "de": "Deutsch",
  "en": "English",
  "xx": "xxxxxxx"
},

"lang.xx": {
  "de": "Dexx",
  "en": "Semxx",
  "xx": "xxxxxxx"
}
```
11 Transferring Data without Reloading Pages Using AJAX

When data is exchanged between a web server and a web browser, web pages are fully reloaded. This limits the performance of updates that are executed quickly. Below you will find a definition of the two methods for data transfer that are used in the example.

AJAX

AJAX ("Asynchronous JavaScript and XML") provides a method for exchanging data asynchronously between a web browser and the web server. Data is exchanged without reloading the web page itself.

Inline frame

Inline frames allow web pages to be split into segments to separate web contents from the surrounding web page. The separated contents are available as independent HTML documents and displayed by the main web page in a defined browser area with a defined size. Therefore, the contents of the inline frame refresh simultaneously with the surrounding page.

The difference to AJAX becomes clear in Firefox as the tab displays an icon for loading the web page and the bottom status pane displays the “Waiting for 192.168.0.1” message. When using AJAX, these messages are not displayed as the web page itself is not reloaded.

Figure 11-1: Web page refresh in Firefox (inline frame)
11.1 Automation task

The task is to read values using two different methods: “AJAX” and “inline frame”. To this end, two user-defined pages with the same structure have to be programmed that are linked to each other.

Basic function:
- Read the input value for the flow velocity “Velocity”
- Display the level as a bar, depending on the flow velocity.

Requirements for the automation task
- Cyclically load a value with the aid of an inline frame.
- Write a value with the aid of a form.
- Cyclically load a value with the aid of AJAX.
- Write a value with the aid of AJAX.

Requirement for communication between STEP 7 and the user-defined page using AJAX

For the refresh using AJAX to work, integrate the file format with dynamic content in the CPU settings.

Figure 11-2: Requirement – refresh using AJAX with "dat" in STEP 7

Table 11-1: Instructions for adding a file format

<table>
<thead>
<tr>
<th>Step</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In the input field, add the “dat” file extension to “.htm;.html;.dat”</td>
</tr>
</tbody>
</table>
11 Transferring Data without Reloading Pages Using AJAX

11.2 Functional mechanisms and use

User-defined page structure

On the user-defined page, enter a value for the flow velocity “Velocity”. The blue bar in the bottom display corresponds to the current level. This level increases depending on the flow velocity.

Figure 11-3: User-defined page for reloading pages using AJAX/inline frame

Table 11-2: User-defined page for reloading pages using AJAX/inline frame

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>①</td>
<td>This is where you enter new values for the flow velocity “Velocity”.</td>
</tr>
<tr>
<td>②</td>
<td>This button transfers the entered value to the controller.</td>
</tr>
<tr>
<td>③</td>
<td>Depending on the input value, the bar is filled in blue.</td>
</tr>
<tr>
<td>④</td>
<td>This link toggles the user-defined page between inline frame and AJAX.</td>
</tr>
</tbody>
</table>

Table 11-3: Instructions for using the user-defined page

<table>
<thead>
<tr>
<th>Step</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>①</td>
<td>In the input field, enter a new value.</td>
</tr>
<tr>
<td>②</td>
<td>Select the button and the value will be transferred to the CPU.</td>
</tr>
<tr>
<td>③</td>
<td>Displays the new value.</td>
</tr>
<tr>
<td>④</td>
<td>The bar fills with the velocity depending on the value.</td>
</tr>
</tbody>
</table>

The system displays both methods for transferring the values. It accesses the same variables in the S7 program; as a result, the level display is synchronized on both user-defined pages.

Figure 11-4: Inline frame (left), AJAX (right) user-defined pages
11.2.1 S7 program structure

A restart resets the variables in DB 1 to their start values. When running OB 1, the user-defined page is refreshed by SFC 99.

Figure 11-5: S7 program for reloading pages using AJAX/inline frame

How OB 100 works

<table>
<thead>
<tr>
<th>No.</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The “DynValue” and “Velocity” variables are reset in OB100.</td>
</tr>
</tbody>
</table>

```
"VelocityVariables".dynValue := 0;
"VelocityVariables".velocity := 50;
```

- “dynValue” defines the bar level.
- “Velocity” defines the bar filling speed.

How OB 1 works

<table>
<thead>
<tr>
<th>No.</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The first block call in OB 1 is the WWW function (SFC 99). For more information about the function, please refer to Chapter 4.2.</td>
</tr>
</tbody>
</table>
| 2   | Each OB 1 cycle increments the “refValue” variable by “1”. When the value “2000” is reached, the “dynValue” variable is increased by “1”. This variable of the [Byte] type scales the level display to 255 steps. If the value is increased by 255+1, the “dynValue” variable is reset to “0”.

```
"VelocityVariables".refValue := "VelocityVariables".refValue + "VelocityVariables".velocity;
```

```
IF "VelocityVariables".refValue >= 2000 THEN
    "VelocityVariables".refValue := 0;
    "VelocityVariables".dynValue := "VelocityVariables".dynValue + 1;
END_IF;
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Display format</th>
<th>Monitor value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;VelocityVariables&quot;.dynValue</td>
<td>DEC</td>
<td>DEC 80</td>
<td>0</td>
</tr>
<tr>
<td>&quot;VelocityVariables&quot;.prevDynValue</td>
<td>DEC</td>
<td>DEC 1238</td>
<td>0</td>
</tr>
<tr>
<td>&quot;VelocityVariables&quot;.velocity</td>
<td>DEC</td>
<td>DEC 1</td>
<td></td>
</tr>
</tbody>
</table>

Contents of the VelocityVariables DB (DB 1)

DB1 contains the variables for this example.
### 11.2.2 User-defined web page structure

The diagrammatic overview below shows the basic file structure of the user-defined pages.

This description focuses on the JavaScript file’s code elements, which makes basic knowledge of HTML necessary for this application example.

Figure 11-6: Diagrammatic overview of the user-defined page

---

**Demo11.html**

Defines the structure of the user-defined page and contains the JavaScript functions for updating the S7 variables using the inline frame. The `<Head>` of the user-defined page contains the JavaScript functions.

**update11.html (inline frame for Demo11.html)**

The inline frame calls the “ForceUpdate” function from the “demo11.html” file to continuously update the “dynValue” (level) PLC tag.

**Demo11_ajax.html**

Defines the structure of the user-defined page and contains the JavaScript functions for AJAX in the user-defined page `<Head>`. The PLC tags are updated using the JavaScript functions.

**update11.dat**

Contains the ““dynValue” and “velocity” reference variables that are called from the JavaScript file in Demo11_ajax.html. (“DoHttpRequest” function).

**demo.css**

Defines the style sheet for the user-defined pages for AJAX and inline frame.

**ajaxbase.js**

The JavaScript file contains the functions for communication and data exchange between different web browsers and web servers.
### 11.2.2.1 Demo11.html structure (inline frame)

<table>
<thead>
<tr>
<th>No.</th>
<th>Demo.html</th>
</tr>
</thead>
</table>
| 1   | The “Start” function is executed after each page refresh.  
\[\text{<body onunload="Start()"}>\] |
| 2   | Two functions are called in the “Start” function.  
- “ForceUpdate”  
- “OnTimer” call is delayed by 1000 milliseconds.  
\[\text{function Start()}\]  
\[\text{ForceUpdate({"VelocityVariables":dynValue});}\]  
\[\text{setTimeout("OnTimer()",1000);}\] |
\[\text{“ForceUpdate” determines the bar length from the HTML document and sends it. Formatting for Firefox and IE is done for the bar display.}\]  
\[\text{function ForceUpdate(val)}\]  
\[\text{var width, barval;}\]  
\[\text{var tablem;}\]  
\[\text{tablem = parent.document.getElementById("bar");}\]  
\[\text{width = tablem.parentNode.clientWidth;}\]  
\[\text{barval = ((val*width)/256);}\]  
\[\text{if (barval == 0) barval = 1;}\]  
\[\text{tablem.style.width = Math.floor(barval)} + \text{"px";}\]  
\[\text{var td = parent.document.getElementById("td1");}\]  
\[\text{if (td.textContent)}\]  
\[\text{\hspace{1cm}td.textContent = val+"";}\]  
\[\text{\hspace{1cm}else}\]  
\[\text{\hspace{2cm}td.innerHTML = val+"";}\]  
\[\text{\hspace{1cm}g_bPageRequested = false;}\]  
\[\text{\}}\]  
\[\text{“OnTimer” updates the value of “DynValue” every 200ms. To do this, an inline frame calls the “ForceUpdate” function.}\]  
\[\text{function OnTimer()}\]  
\[\text{\hspace{1cm}if (! g_bPageRequested)}\]  
\[\text{\hspace{2cm}g_bPageRequested = true;}\]  
\[\text{\hspace{2cm}window.frames[\"HiddenFrame\"].document.location.replace(\"updateall.html\");}\]  
\[\text{\hspace{2cm}setTimeout("OnTimer()", 200);}\]  
\[\text{\}}\]  
| 3   | A form that sends values using a “submit” button is used for writing the “Velocity” value. |
### Demo11_ajax.html structure (AJAX)

<table>
<thead>
<tr>
<th>No.</th>
<th>Demo11_ajax.html</th>
</tr>
</thead>
</table>
| 1   | The “Start” function is executed after each page refresh.  
     | `<body onload="Start()">` |
| 2   | Three functions are called in the “Start” function.  
     | - “DetermineBrowser” determines the browser type (Mozilla, IE11, Chrome, etc.). The function is included in the “ajaxbase.js” file.  
     | - “ForceUpdate”: same function as when using the inline frame to determine the bar length.  
       |     | (See Chapter 11.2.2.1)  
     | - “OnTimer” call is delayed by 1000 milliseconds. |

```javascript
function Start()
{
    DetermineBrowser();
    ForceUpdate(="VelocityVariables\$.dynValue");
    setTimeout("OnTimer()",1000);
}
```

"OnTimer" updates the value of DynValue every 200ms using the "DoHttpRequest" function call from this file: “ajaxbase.js”. Four values are transferred to the function.
1. The object that has called the current function.  
2. The URL "update11.dat" with the variables to be updated.  
3. The “UpdateCallback” function that further processes the value and the status code.  
4. The “true” value indicates that the data transfer is asynchronous.

When all values have been updated, the “UpdateCallback” function is called.
```
function OnTimer()
{
    if (! g_bPageRequested)
    {
        g_bPageRequested = true;
        DoHttpRequest(this, "update11.dat", UpdateCallback, true);
    }
    setTimeout("OnTimer()", 200);
}
```
The “UpdateCallback” function processes the read values and the status code. The “response” variable contains the updated values of the PLC tags. For processing in a JavaScript file, the values must be split and reassigned. The “status” variable contains the current HTTP status code.

If the HTTP status code is less than 300, the “ForceUpdate” function is called and the length of the updated bar is calculated.

```javascript
function UpdateCallback(obj, response, status) {
  var ok;
  var results = response.split(" ");
  var signs = results[0].split(""');
  var i;
  var count = 0;
  for (i = 0; i < signs.length; i++) {
    if (true == isNaN(signs[i])) {
      count = count + 1;
    } else { break; }
  }
  dynValue = results[0].substr(count, signs.length);
  var dynValueInt = parseInt(dynValue);

  if (status < 300) {
    document.getElementById('veloDiv').innerHTML = results[i],
    ForceUpdate(dynValueInt);
    g_bPageRequested = false;
    setTimeout("OnTimer()", 200);
    return;
  } else { ok = confirm(dynValueInt);
    if (ok) {
      setImmediate("OnTimer()", 1000);
    } else { g_bPageRequested = false;
    setTimeout("OnTimer()", 1000);
    }
  }
}
```
### Demo11_ajax.html

<table>
<thead>
<tr>
<th>No.</th>
<th>Selecting the “Send via AJAX” button calls the “send_ajax_request” function.</th>
</tr>
</thead>
</table>
| 5   | The current velocity of change is<br><input type="button" onclick="send_ajax_request('velocityField')" value="Send via AJAX">

The “send_ajax_request” function transfers the entered value to the CPU.

```javascript
function send_ajax_request(variable, fieldId) {
    if (window.XMLHttpRequest)
    {
        req = new XMLHttpRequest();
    }
    else if (window.ActiveXObject)
    {
        req = new ActiveXObject("Microsoft.XMLHTTP");
    }
    else
    {
        alert("Der Browser unterstuezt kein Ajax");
    }
    var value = document.getElementById(fieldId).value;
    var req_url = "?"+variable+"=\"+value+\"&\"+Math.random();
    //debug alert(req_url);
    req.open("GET", req_url, false);
    req.onreadystatechange = ajax_callback;
    req.send(null);
}
```
12 Displaying Data of a Datalog as a Graph

12.1 Automation task

The task is to save values of variables to a datalog using the S7 controller. Datalogs are CSV files that are written to the memory card or the CPU’s internal load memory.

The data volume depends on the available memory.

The data from the datalog is to be displayed on a user-defined page as a graph. The data is not updated before the user clicks a button [Load new data].

Note

In this example, sine and cosine values are cyclically written to the SMC or the CPU’s memory.

Regarding the read/write cycles, please refer to this entry:


Alternatively, you can program a ring buffer for the values that is used to save a limited number of values and display them in the graph.

Requirements for the automation task

- Cyclically read a datalog.
- Chronologically sort the read data.
- Display the data as a graph.
12.2 Functional mechanisms and use

User-defined page structure

The user-defined page displays the data from the datalog in a line graph. The name of the datalog’s CSV file is “SinusUndCosinus.csv”.

Figure 12-1: User-defined page for displaying data from a datalog in a line graph

Table 12-1: User-defined page for displaying data from a datalog in a line graph

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Datalog stored on memory card as “SinusUndCosinus.csv” file</td>
</tr>
<tr>
<td>2</td>
<td>Graph displayed on user-defined page (source file: “SinusUndCosinus.csv”)</td>
</tr>
<tr>
<td>3</td>
<td>Button for updating data [Load new data]</td>
</tr>
</tbody>
</table>
12.2.1 S7 program structure

When running OB 1, the user-defined page is refreshed by SFC 99.

Figure 12-2: S7 program for displaying data from a datalog in a line graph

How OB 1 works

<table>
<thead>
<tr>
<th>No.</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The first block call in OB 1 is the WWW function (SFC 99). For more information about the function, please refer to Chapter 4.2.</td>
</tr>
<tr>
<td>2</td>
<td>FB 1 calculates sine and cosine values and creates a datalog. The parameters and name of the datalog CSV file are written to static variables. The sine and cosine values are cyclically saved to this datalog.</td>
</tr>
</tbody>
</table>

The status and error code and the input parameter for deleting the datalog content is stored in DB 2.
12.2.2 **User-defined web page structure**

The diagrammatic overview below shows the basic file structure of the example.

Figure 12-3: Diagrammatic overview of the user-defined page
Demo12.html with (graph container)

Defines the structure of the user-defined page and includes a container for displaying the graph. The JavaScript files are integrated into the HTML document to display the graph.

Figure 12-4: Demo12.html with graph

Table 12-2: Demo12.html with graph

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Demo12.html</td>
</tr>
<tr>
<td>2</td>
<td>Graph container with access to the datalog data</td>
</tr>
</tbody>
</table>

Demo.js

The “Demo12.js” file calls functions from the different libraries to create a graph from the data of a datalog (CSV file). For information about processing the data, please refer to the source code.

jquery.flot.min.js and jquery.flot.time.min.js (graph template)

These JavaScript files calculate the graph in the inline frame display. The programming used elements from the jQuery library.

For more information, visit: [http://www.flotcharts.org/](http://www.flotcharts.org/)

S7_framework.js

The JavaScript file contains a number of functions for converting and processing data from a SIMATIC S7.

For more information about the S7 Framework, please refer to Chapter 4.3.

S7_framework.css

Defines the style sheet for the HTML user-defined pages

jquery-2.1.3.min.js (jQuery library)

The jQuery library file allows more efficient programming with JavaScript. For more information about the jQuery library, please refer to Chapter 4.3.
12 Displaying Data of a Datalog as a Graph

12.2.2.1 Demo12.html structure

<table>
<thead>
<tr>
<th>No.</th>
<th>Demo12.html code section</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The JavaScript files used are integrated into the HTML document's <code>&lt;head&gt;</code> with the below syntax:</td>
</tr>
<tr>
<td></td>
<td><code>&lt;head&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;meta http-equiv=&quot;content-type&quot; content=&quot;text/html; charset= UTF-8&quot;&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;link rel=&quot;stylesheet&quot; type=&quot;text/css&quot; href=&quot;CSS/S7 Framework.css&quot;&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;script type=&quot;text/javascript&quot; src=&quot;script/jquery-2.1.3.min.js&quot;&gt;&lt;/script&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;script type=&quot;text/javascript&quot; src=&quot;script/jquery.flot.min.js&quot;&gt;&lt;/script&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;script type=&quot;text/javascript&quot; src=&quot;script/jquery.flot.time.min.js&quot;&gt;&lt;/script&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;script type=&quot;text/javascript&quot; src=&quot;script/S7 Framework 0.1.7.js&quot;&gt;&lt;/script&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;script type=&quot;text/javascript&quot; src=&quot;script/Demo12.js&quot;&gt;&lt;/script&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/head&gt;</code></td>
</tr>
<tr>
<td></td>
<td>The &quot;src&quot; property specifies the respective file path. User-defined pages cannot directly access a datalog.</td>
</tr>
<tr>
<td>2</td>
<td>The datalog CSV files are stored on the standard web page that must be opened for access.</td>
</tr>
<tr>
<td></td>
<td>The graph is called in a container element.</td>
</tr>
<tr>
<td></td>
<td><code>&lt;div id=&quot;graph-placeholder&quot;&gt;&lt;/div&gt;</code></td>
</tr>
<tr>
<td></td>
<td>The &quot;graph-container&quot; &quot;div&quot; tag is a placeholder for the graph.</td>
</tr>
<tr>
<td></td>
<td>The &quot;Demo12.js&quot; JavaScript file composes the graph in the placeholder.</td>
</tr>
</tbody>
</table>
### Demo12.js structure

<table>
<thead>
<tr>
<th>No.</th>
<th>Code section</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>Determining the CPU type (S7-1200 or S7-1500)</strong></td>
</tr>
<tr>
<td></td>
<td>In the &quot;Demo12.js&quot; JavaScript file, a function to determine whether an S7-1200 or S7-1500 is used is executed in &quot;$(document).ready(function())&quot;.</td>
</tr>
</tbody>
</table>
|     | ```javascript  
|     |   $(document).ready(function(){  
|     |     if (plcType == "1200" && plcType != "1500")  
|     |     $.ajax({type:"GET",url:".../Portal/Intro.mws",data:"",dataType:"text"})  
|     |     .done(function(webpageData){  
|     |     var search12 = webpageData.search("CPU12");  
|     |     var search15 = webpageData.search("CPU15");  
|     |     if (search12 >= 0)  
|     |     {  
|     |     plcType = "1200";  
|     |     s.init();  
|     |     }  
|     |     else if (search15 >= 0)  
|     |     {  
|     |     plcType = "1500";  
|     |     s.init();  
|     |     }  
|     |     })  
|     |     });  
|     |     });  
|     | });  
|     | ``` |
|     | Alternatively, you can directly call the functions of "$.init" in "$(document).ready(function())". Replace "plcType" with "1200" or "1500", including quotation marks. |
|     | The PLC type is transferred to the S7 Framework with the following function. |
|     | ```javascript  
|     |   $.init = function(){  
|     |     S7Framework.initialize(plcType, "");  
|     |     S7Framework.readDataLog("SinusUndCosinus","Read Datalog failed", decodeCSV);  
|     |   }  
|     | ``` |
|     | Alternatively, write "1200" or "1500" for plcType. Example: S7Framework.initialize("1200", ""); |
| 2   | **Datalog (read values)**  |
|     | The "readDataLog" function reads datalog files. |
|     | ```javascript  
|     |   S7Framework.readDataLog("SinusUndCosinus","Read Datalog Failed",decodeCSV);  
|     | ``` |
|     | The name of the "SinusUndCosinus" datalog has to be transferred to the function. Text as an error message is optional. |
|     | The function name of "decodeCSV" is the destination for the read values. |
| 3   | **Datalog (process values)**  |
|     | The "decodeCSV" function processes the values from the datalog. |
|     | ```javascript  
|     |   function decodeCSV(CSVData){  
|     |   All time values are converted to [Date.UTC] format and sorted chronologically. The data is prepared for the graph template and evaluated in the JavaScript file (jquery.flot.time.min.js). This function also defines the graph properties. (For example, connect points, no steps in graph.)
13 Display Elements Using Scalable Vector Graphics

13.1 Programming task

The task is to create a selection of display elements using Scalable Vector Graphics (SVG). HTML5-capable browsers support this technique.

Requirements for the automation task
- Create SVG elements
- Dynamize SVG elements using JavaScript

13.2 Functional mechanisms and use

Note
This example does not contain a specific S7 program. The display elements do not explicitly access data of a SIMATIC S7. You will find the HTML file in the project folder in “html_indicatingElements”.

User-defined page structure

The user-defined page shows different display elements that can be modified and controlled using a slider below the display elements. For connection to an S7, replace the slider with the appropriate UDP variables.

Figure 13-1: User-defined page with SVG (Scalable Vector Graphics) display elements
13 Display Elements Using Scalable Vector Graphics

13.2.1 Sample SVG element – rotary motion of a motor

Programming SVG elements

Define the shape of SVG display elements by the HTML code. Program a JavaScript file for the motion functions. In general, use the input variables to change the position/rotation of circular/rectangle elements. The input variables are implemented by sliders below the SVG elements.

The following figure shows an example of a motor’s rotary motion and the associated code section in the HTML document. The motor is composed of a total of three circular elements. The rotary motion is implemented by a dash line (white, red, white, red) that rotates inside a second circle for the outer boundary. The inner circle represents the rotor’s cam.

You can scale the motor by changing the values in quotation marks. Always change the value for the dash line such that the circle’s circumference exactly matches 4 dashes. The specific parameter for this is called “stroke-dasharray”.

Figure 13-2: Sample SVG display element – motor design

Table 13-1: Motor design display elements

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Circles: Define position, size and colors for three circular elements.</td>
</tr>
<tr>
<td>2</td>
<td>Slider: Graphic element for entering values.</td>
</tr>
<tr>
<td>3</td>
<td>Function: rotate motor</td>
</tr>
</tbody>
</table>

Note

To extend your application, copy the SVG element’s code section directly to your HTML document. If necessary, replace the input variable of the slider element as required for your application.
13.2.1.1 Demo13.html structure

<table>
<thead>
<tr>
<th>No.</th>
<th>Code sections from (Gauge)</th>
</tr>
</thead>
</table>
| 1   | `<svg width="110" height="110"> ...</svg>`  
Der "svg" tag defines the graphic area.  
The graphic area starts at the top left and the size of the coordinate axes  
corresponds to the graphics size in pixels.  
In the "svg" tag, you can create the following elements: |
| 2   | `<circle cx="55" cy="55" r="40" />`  
The "circle" tag creates a circle.  
The "cx" property specifies the position of the center point on the X axis, "cy"  
defines the corresponding position on the Y axis. The "r" property specifies the  
radius. |
| 3   | `stroke="#ff0033"`  
The "stroke" property specifies the border color/line color of SVG elements. |
| 4   | `stroke-width="20"`  
The "stroke-width" property specifies the border thickness/line thickness of SVG  
elements. |
| 5   | `stroke-dasharray="188.25 63.08"`  
The "stroke-dasharray" property deletes parts of the border or a line of SVG  
elements.  
The first value specifies the length to be shown. The second value specifies the  
length to be deleted. The two lengths alternate on the border/line until they have  
circled around the border/reached the end of the line. As a result, you get a dash  
border/line.  
Note: For circles, the border is the circumference.  
It is calculated as follows: 2 * π * r. |
| 6   | `fill="white"`  
The "fill" property specifies the fill color or font color of the SVG element. |
| 7   | `transform="rotate(135, 55, 55)"`  
The "transform rotate" property rotates SVG elements. The first number is the  
number of degrees and specifies how far the element will be rotated. The second  
and third number specify the point around which the element will be rotated. |
| 8   | `id="gauge.pointer"`  
The "id" property assigns an SVG element an ID to be accessed using JavaScript. |
<table>
<thead>
<tr>
<th>No.</th>
<th>Code section from (Tank)</th>
</tr>
</thead>
</table>
| 1   | `<rect x="10" y="10" width="30" height="10" />`  
The “rect” tag creates a rectangle. The “x” property specifies the top left point of the rectangle on the X axis, “y” defines the corresponding point on the Y axis. The “width” property specifies the width and “height” defines the height of the rectangle. |
| 2   | `<line x1="40" y1="10" x2="40" y2="110" />`  
The “line” tag creates a line. The “x1” and “y1” properties specify the line’s start point. The “x2” and “y2” properties specify the line’s end point. |
| 3   | `<polygon points="50,55 60,55 55,5" />`  
The “polygon” tag creates a polygon. The “points” property specifies the polygon’s corners. |
| 4   | `<text x="40" y="35" font-size="10" font-family="Helvetica, Arial, sans-serif">Speed</text>`  
The “text” tag creates text.  
The “x” property specifies the position of the bottom right point of the text field on the X axis, “y” defines the corresponding point on the Y axis.  
The “font-size” property specifies the font size.  
The “font-family” property specifies the font.  
The desired text is written to this tag. |

<table>
<thead>
<tr>
<th>No.</th>
<th>Code section (structure of a JavaScript script)</th>
</tr>
</thead>
</table>
| 1   | `<script>`  
  var newValue  
  …  
  document.getElementById("gauge.pointer").setAttribute("transform","rotate("+newValue+",55,55)")  
`</script>`  
  “document.getElementById()”  
  Specifies the element to be accessed or modified.  
  “setAttribute”  
  The function changes the property of elements. Elements that do not represent JavaScript variables are put in quotation marks. Prior to executing the function, JavaScript variables are replaced with the variable value. |

**Note**  
For more information about creating SVG elements, visit:  
http://www.w3schools.com/html/html5_svg.asp  
http://www.w3schools.com/graphics/svg_intro.asp  
http://wiki.selfhtml.org/wiki/SVG
14 S7 Diagnostics and Loading Indicator

14.1 Automation task

On the standard web page, the S7 web server provides the option to read the CPU's diagnostic buffer.

In practice, one possible requirement could be to display the diagnostic buffer in conjunction with other information on a user-defined page.

The task is to program a user-defined page that displays the diagnostic buffer of the S7 controller. Loading is to be indicated by an animated icon as a loading indicator.

Requirements for the automation task

- Display diagnostic information of the S7 controller on a user-defined page
- Integrate a loading indicator with the S7 Framework

14.2 Functional mechanisms and use

User-defined page structure

The user-defined page displays the diagnostic information of the CPU. Clicking a diagnostic message displays detailed information about this message below the table.

Figure 14-1: User-defined page with diagnostic data
### 14.2.1 S7 program structure

In the S7 program, only SFC 99 is called.

Figure 14-2: S7 program for the user-defined page with diagnostic data and loading indicator

<table>
<thead>
<tr>
<th>No.</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The first block call in OB 1 is the WWW function (SFC 99). For more information about the function, please refer to Chapter 4.2.</td>
</tr>
</tbody>
</table>

### 14.2.2 User-defined web page structure

The diagrammatic overview below shows the basic structure of the user-defined page.

**User-defined page structure**

Figure 14-3: Diagrammatic overview of the user-defined page
14 S7 Diagnostics and Loading Indicator

**Demo14.html**

The HTML document defines the structure of the user-defined page and the calls of the JavaScript files used, see Chapter 14.2.2.1. The document defines that clicking the diagnostic messages displays them in a table.

**Demo14.js**

The JavaScript file integrates the S7 Framework to start the loading indicator.

**diagnostic_0.0.1.js**

The “diagnostic_0.0.1.js” file accesses tags with a defined ID. The diagnostic display is inserted into these tags.

**S7_framework.js**

The JavaScript file contains a number of functions for converting and processing data from a SIMATIC S7.

For more information about the S7 Framework, please refer to Chapter 4.3.

**jquery-2.1.3.min.js (jQuery library)**

The jQuery library file allows more efficient programming with JavaScript. For more information about the jQuery library, please refer to Chapter 4.3.

**diag.css (style sheet)**

Style sheet for the loading indicator. This style sheet defines the position and web animation for the “…\img\sprites.png” file.

**S7_diagnostic.css (style sheet)**

Style sheet for designing the table with the diagnostic information.

**Load / Sprites (images)**

The loading indicator is an animated file in GIF format that accesses the graphic elements of a PNG file as shown below.

Figure 14-4: Designing image for load indicator

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PNG source file with 12 single images, “sprites.png”</td>
</tr>
<tr>
<td>2</td>
<td>GIF target file with time change of the single images (animation), “load.gif”</td>
</tr>
</tbody>
</table>
## 14.2.2.1 Demo14.html structure

<table>
<thead>
<tr>
<th>No.</th>
<th>Code description</th>
</tr>
</thead>
</table>
| 1   | The style sheets and JavaScript files used are integrated into the HTML document's `<head>`:  
     **Style sheets:** `<link rel=...>`  
     **JavaScript files:** `<script type=...>`  

```html
<head>
  <meta http-equiv="content-type" content="text/html; charset=UTF-8">
  <link rel="stylesheet" type="text/css" href="/S7_diagnostic.css">
  <script type="text/javascript" src="/jquery-2.1.3.min.js"></script>
  <script type="text/javascript" src="/diagnostic_0.0.1.js"></script>
  <script type="text/javascript" src="/S7_framework_0.1.7.js"></script>
  <script type="text/javascript" src="/Demo14.js"></script>
</head>
```

The `src` property specifies the respective file path.  

| 2   | The loading indicator is displayed in this "div" tag.  

```html
<body>
  <div id="loading_div">
    <div class="loading_pic"></div>
  </div>
</body>
```

| 3   | In this "div" tag, the diagnostic information is displayed as a table.  

```html
<diagramDiv id="diagTableDiv" width="1000px" height="250px"></diagramDiv>
```

| 4   | In this "div" tag, the "Event" information of a diagnostic message is displayed in the table when the appropriate message is clicked.  

```html
<diagramDiv id="diagDetailDiv" width="1000px" height="250px">
  <span id="diagDetailNo"></span>
  <span id="diagDetailEventID"></span>
  <span id="diagDetailText">DiagnosticDetail</span>
</diagramDiv>
```
### Demo14.js structure

<table>
<thead>
<tr>
<th>No.</th>
<th>Code description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Determining the CPU type (S7-1200 or S7-1500)</td>
</tr>
</tbody>
</table>
> In the "Demo12.js" JavaScript file, a function to determine whether an S7-1200 or S7-1500 is used is executed in "$(document).ready(function(){})."

```javascript
if (plcType == "1200" & plcType != "1500") {
    $.ajax({
        type: "GET", url: "../Portal/Intro.html", data: "", dataType: "text"
    }).done(function(webpageData) {
        var search12 = webpageData.search("CPU12");
        var search15 = webpageData.search("CPU15");
        if (search12 > 0) {
            plcType = "1200";
            $.init();
        } else if (search15 > 0) {
            plcType = "1500";
            $.init();
        }
    });
```  
Alternatively, you can directly call the functions of "$.init" in "$(document).ready(function(){})."
Replace "plcType" with "1200" or "1500", including quotation marks.

| 2   | This function transfers the CPU type and the ID of the "div" tag where the loading indicator is to be displayed to the S7 Framework. The S7 Framework detects when an AJAX function is executed and shows/hides the loading indicator accordingly. |
> This function runs the script from diagnostic_0.0.3.js. It inserts the diagnostic information into the "div" tags of the HTML file. The "div" tags are identified by the appropriate IDs.

```javascript
$.init = function() {
    S7Framework.initialize(plcType, "#loading_div");
}
```  

The loading indicator is shown when the "div" tag with ID "diagTableDiv" is empty. The following query is programmed for this.

```javascript
if ($("#diagTableDiv") == "[object Object]()") {
    $("#loading_div").show();
} else {
    $("#loading_div").hide();
}
```
### 14.2.2.3 diag.css structure

<table>
<thead>
<tr>
<th>No.</th>
<th>Code description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The “sprites.png” image for the loading indicator is in the “img” folder. The “diag.css” file references the sprites.png image as shown below.</td>
</tr>
</tbody>
</table>

```css
data:loading_pic{
  position: absolute;
  width: 64px;
  height: 64px;
  top: 50%;
  left: 50%;
  margin-top: -32px;
  margin-left: -32px;
  background-image: url(../../img/sprites.png);
  animation: play .8s steps(12) infinite;
  -moz-animation: play .8s steps(12) infinite;
  -ms-animation: play .8s steps(12) infinite;
  -o-animation: play .8s steps(12) infinite;
  animation-direction: reverse;
}
```
15 Changing Pictures Using JavaScript

15.1 Programming task

The task is to change a picture depending on a JavaScript variable.

Requirements for the automation task

- Display pictures depending on a JavaScript variable

15.2 Functional mechanisms and use

Note

This example has no S7 program. You will find the HTML file in the project folder in “html_pictureChange”.

User-defined page structure

The user-defined page displays a picture with two buttons below it.

Figure 15-1: User-defined page for changing pictures using a JavaScript variable

Table 15-1: Instructions for using the user-defined page

<table>
<thead>
<tr>
<th>Step</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Select the “Red” button and the red picture will be displayed.</td>
</tr>
<tr>
<td>2</td>
<td>Select the “Green” button and the green picture will be displayed.</td>
</tr>
</tbody>
</table>
15 Changing Pictures Using JavaScript

15.2.1.1 Demo15.html structure

Creating SVG elements

<table>
<thead>
<tr>
<th>No.</th>
<th>Code description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>With this HTML element, the red picture is displayed when starting the user-defined page.</td>
</tr>
<tr>
<td></td>
<td>&lt;img id=&quot;pic&quot; src=&quot;img\red.png&quot;&gt;</td>
</tr>
<tr>
<td>2</td>
<td>With these HTML elements, one button labeled “Red” and one button labeled “Green” are displayed.</td>
</tr>
<tr>
<td></td>
<td>![Red](button onclick=&quot;changePic('red')&quot;&gt;Red&lt;/button&gt;</td>
</tr>
<tr>
<td></td>
<td>![Green](button onclick=&quot;changePic('green')&quot;&gt;Green&lt;/button&gt;</td>
</tr>
<tr>
<td>3</td>
<td>The “changePic” function is defined in a JavaScript part of the HTML file.</td>
</tr>
<tr>
<td></td>
<td>To change a picture depending on a PLC tag, the PLC tag must be read cyclically. Cyclic reading is done by the IF statement.</td>
</tr>
<tr>
<td></td>
<td>When the “green” value has been transferred to the function, the previous path to the red picture is changed to the path of the green picture.</td>
</tr>
<tr>
<td></td>
<td>As a result, the green picture is displayed.</td>
</tr>
<tr>
<td></td>
<td>&lt;script&gt;</td>
</tr>
<tr>
<td></td>
<td>function changePic(color)</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>if (color == &quot;green&quot;)</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>document.getElementById(&quot;pic&quot;).src = &quot;img\green.png&quot;;</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>else</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>document.getElementById(&quot;pic&quot;).src = &quot;img\red.png&quot;;</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>&lt;/script&gt;</td>
</tr>
</tbody>
</table>
16 Button for CPU Restart

16.1 Automation task

The task is to stop and then automatically restart the CPU using a button.

Requirements for the automation task

- Implement a button for restarting the CPU

16.2 Functional mechanisms and use

User-defined page structure

Selecting the “PLC Restart” button stops and then automatically restarts the CPU. Technically, this is a warm restart of the CPU. Program execution restarts and retentive data is retained.

Figure 16-1: User-defined page with button for CPU restart

![User-defined page with button for CPU restart](image)
16.2.1 S7 program structure

In the S7 program, only SFC 99 is called.

Figure 16-2: S7 program for user-defined page with button for CPU restart

How OB 1 works

<table>
<thead>
<tr>
<th>No.</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The first block call in OB 1 is the WWW function (SFC 99). For more information about the function, please refer to Chapter 4.2.</td>
</tr>
</tbody>
</table>

16.2.2 User-defined web page structure

The diagrammatic overview below shows the basic file structure of the user-defined page.

Figure 16-3: Diagrammatic overview of the user-defined page

Demo16.html

In the “Demo16.html” file, all JavaScript files are integrated and the button is defined.

Demo16.js

The JavaScript file performs the restart with the aid of “S7_framework.js”.

S7_framework.js

The JavaScript file contains a number of functions for converting and processing data from a SIMATIC S7.

For more information about the S7 Framework, please refer to Chapter 4.3.

jquery-2.1.3.min.js (jQuery library)

The jQuery library file allows more efficient programming with JavaScript. For more information about the jQuery library, please refer to Chapter 4.3.
16.2.2.1 Demo16.html structure

<table>
<thead>
<tr>
<th>No.</th>
<th>Code description</th>
</tr>
</thead>
</table>
| 1   | The JavaScript files used are integrated into the HTML document’s `<head>`:  
  ```html
  <head>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8">
    <script type="text/javascript" src="script/jquery-2.1.3.min.js"></script>
    <script type="text/javascript" src="script/37.Framework 0.1.7.js"></script>
    <script type="text/javascript" src="script/Demo16.js"></script>
  </head>
  ```  
  The “src” property specifies the respective file path. |
| 2   | Create a button named “PLC Restart”.  
  ```html
  <div class="container">
    <h1>Restart the PLC</h1>
    <button id="restart" class="PLC Restart">Button for CPU Restart</button>
  </div>
  ``` |

16.2.2.2 Demo16.js structure

<table>
<thead>
<tr>
<th>No.</th>
<th>Code description</th>
</tr>
</thead>
</table>
| 1   | Determining the CPU type (S7-1200 or S7-1500)  
In the “Demo12.js” JavaScript file, a function to determine whether an S7-1200 or S7-1500 is used is executed in “$(document).ready(function());”.  
```javascript
$(document).ready(function() {  
  if (plcType == "1200" || plcType == "1500")  
  {  
    $.ajax({type:"GET",url:".../Portal/Intro.html",data:"",dataType:"text"})
      .done(function(webpageData) {  
        var search1 = webpageData.search("CPU1200");
        var search2 = webpageData.search("CPU1500");
        if (search1 >= 0)  
        {  
          plcType = "1200";
          $.init();
        }  
        else if (search2 >= 0)  
        {  
          plcType = "1500";
          $.init();
        }
      });
  }
});
```  
Alternatively, you can directly call the functions of “$.init” in “$(document).ready(function());”.  
Replace “plcType” with “1200” or “1500”, including quotation marks. |
| 2   | The “S7Framework.initialize(plcType,"")” function transfers the CPU type to the S7 Framework.  
Selecting the button with the “restart” ID restarts the CPU using the “S7Framework.restartCPU();” function.  
```javascript
$.init = function(){  
  S7Framework.initialize(plcType, "");
  $('.restart').click(function(){  
    S7Framework.restartCPU();
  });
}  ``` |
17 Login on User-Defined Pages

17.1 Automation task

The task is to display the user-defined pages via a separate login window.

Requirements for the automation task

- Integrate the login window
- Edit the style properties of the login window

17.2 Functional mechanisms and use

User-defined page structure

On the user-defined page, you log in using the input fields and the “Log in” button. Use the login information listed in Chapter 4.1.

Figure 17-1: Login on user-defined pages
17.2.1 S7 program structure

In the S7 program, only SFC 99 is called.

Figure 17-2: S7 program for login on the user-defined page

<table>
<thead>
<tr>
<th>No.</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The first block call in OB 1 is the WWW function (SFC 99). For more information about the function, please refer to Chapter 4.2.</td>
</tr>
</tbody>
</table>

How OB 1 works

17.2.2 User-defined page structure

The diagrammatic overview below shows the basic file structure of the user-defined page.

Figure 17-3: Diagrammatic overview of the user-defined page

- **Demo17.html**
  All JavaScript files are integrated into the “Demo17.html” file.

- **Demo17.js**
  The JavaScript file allows the user to log in and out with the aid of “S7_framework.js”.

- **S7_framework.js**
  The JavaScript file contains a number of functions for converting and processing data from a SIMATIC S7.
  For more information about the S7 Framework, please refer to Chapter 4.3.

- **jquery-2.1.3.min.js** (jQuery library)
  The jQuery library file allows more efficient programming with JavaScript. For more information about the jQuery library, please refer to Chapter 4.3.
17.2.2.1 Demo17.html structure

<table>
<thead>
<tr>
<th>No.</th>
<th>Code description</th>
</tr>
</thead>
</table>
| 1   | The JavaScript files used are integrated into the HTML document’s <head> with the below syntax:  

```html  
<head>  
    <title>Log In</title>  
    <meta charset="UTF-8">  
    <script type="text/javascript" src="/javascript/jquery-2.1.3.min.js"></script>  
    <script type="text/javascript" src="/javascript/s7_framework_01.7.js"></script>  
    <script type="text/javascript" src="/javascript/login17.js"></script>  
</head>  
```

The "src" property specifies the respective file path. |
| 2   | The "login" "div" box defines the login area as an inline frame with the "src=..." source from the standard web page. Using the JavaScript file, the login window is inserted into the "loginBox" "div" box.  

```html  
<html>  
    <div id="login">  
        <iframe id="WebserverIFrame"  
            style="display:none"  
            name="WebserverIFrame"  
            src="/Portal/Portal.mws1">  
        </iframe>  
        <div id="loginBox"></div>  
    </div>  
</html>  
```

| 3   | In the <style> tag, the appropriate IDs or the class is accessed to change the style properties of the input fields and buttons.  

```html  
/*S7-1500 */  

#Login_Area_Name_InputTag{ /*Input field username*/  
    width:100px;  
}  

#Login_Area_PW_InputTag{ /*Input field password*/  
    width:100px;  
}  

#Login_Area_SubmitButton{ /*Button for login*/  
    width:80px;  
}  

.Logout_Button{ /*Button for logout*/  
    width:100px;  
}  

#logout_user_name{ /*div of username after login*/  
    font-family: Arial,Helvetica Neue,Helvetica,sans-serif;  
}  
```
### 17.2.2.2 Demo17.js structure

<table>
<thead>
<tr>
<th>No.</th>
<th>Code description</th>
</tr>
</thead>
</table>
| 1   | **Determining the CPU type (S7-1200 or S7-1500)**  
In the "Demo1.js" JavaScript file, a function to determine whether an S7-1200 or S7-1500 is used is executed in "$(document).ready(function(){
   if (plcType != "1200" && plcType != "1500")
      $.ajax({type:"GET",url:"/.../Portal/Intro.mxml",data:"",dataType:"text"})
      .done(function(webpageData){
         var search12 = webpageData.search("CPU12");
         var search15 = webpageData.search("CPU15");
         if (search12 >= 0)
            plcType = "1200";
            $.init();
         else if (search15 >= 0)
            plcType = "1500";
            $.init();
   });
});

Alternatively, you can directly call the functions of “$.init” in "$(document).ready(function[]{}". Replace “plcType” with “1200” or “1500”, including quotation marks. |
| 2   | This function transfers the CPU type to the S7 Framework.  
$$.init = function(){
   S7Framework.initialize(plcType, "");
};

| 3   | When the page has loaded, the system checks whether the user is already logged in via the standard web page or a previous login operation.  
If no user is logged in, a pop-up appears that reads:  
"Please log in!" |

```javascript
$(window).load(function(){
   if(S7Framework.loginCheck()){
   }
   else{
      alert("Please log in!");
   }
});
```
18 High-Performance Communication via a String

When data is exchanged between a web server and a web browser, variables are transferred individually. If you want to implement comprehensive, high-performance data exchange between the web server and the controller, this is achieved by communication of a concatenated string. As another measure to increase performance, use AJAX.

AJAX

AJAX ("Asynchronous JavaScript and XML") provides a method for exchanging data asynchronously between a web browser and the web server. Data is exchanged without reloading the web page itself.

18.1 Automation task

High-performance transfer of larger data volumes to the controller.

Requirements for the automation task

- The values in the controller are converted to an ASCII string with a maximum of 255 characters.
- ASCII strings are cyclically read using the S7 Framework (JavaScript) and split into single variables.
- The values are individually sent to the controller.
- The values are sent from a form to the controller.

Requirement for communication between STEP 7 and the user-defined page using AJAX

Updating and writing PLC tags using AJAX is implemented on the user-defined page. To this end, the "json" file format must be added to the user-defined page properties in "Files with dynamic content:"

Figure 18-1: Requirement – update using AJAX with json in S7

<table>
<thead>
<tr>
<th>Step</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In the input field, add the &quot;json&quot; file extension to &quot;.htm;html;:.json&quot;.</td>
</tr>
</tbody>
</table>
18.2 Functional mechanisms and use

The user-defined page simulates a package (gray rectangle) that moves from the left to the right in the display area bordered in black. The value entered in “Speed” controls the speed of the package.

When the package reaches the sensor (green circle), this sensor changes from “green” to “red”. A black horizontal line marks the sensor’s detection range; a count below this line shows the detected packages.

The [send] button transfers the information entered for name and age to the controller.

Figure 18-2: User-defined page for high-performance data transfer using AJAX

Table 18-2: User-defined page for high-performance data transfer using AJAX

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>With the “Speed” value, you influence the speed of the package.</td>
</tr>
<tr>
<td>2</td>
<td>Signal state of the sensor: (green = free; red = package detected)</td>
</tr>
<tr>
<td>3</td>
<td>Count that shows the packages that have already passed.</td>
</tr>
<tr>
<td>4</td>
<td>[send] transfers the “name” and “age” to the controller.</td>
</tr>
</tbody>
</table>

Table 18-3: Instructions for using the user-defined page

<table>
<thead>
<tr>
<th>Step</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Change the “Speed”.</td>
</tr>
<tr>
<td>2</td>
<td>Enter the “name” and “age” of the package’s recipient.</td>
</tr>
<tr>
<td>3</td>
<td>Select the “send” button.</td>
</tr>
<tr>
<td>4</td>
<td>In TIA Portal, open the “Plc2Web” data block.</td>
</tr>
<tr>
<td>5</td>
<td>Click “Monitor”.</td>
</tr>
<tr>
<td>6</td>
<td>Verify that the name and age have been received.</td>
</tr>
</tbody>
</table>
18.2.1 S7 program structure

The WWW function from OB 1 is called in the S7 program. FB 1 “Counter” is used for simulating the package and the sensor. FC 1 “WebCom” is used for converting the variables to a string.

Figure 18-3: S7 program for high-performance data transfer using AJAX
How the PLC program works

<table>
<thead>
<tr>
<th>No.</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The first block call in OB 1 is the WWW function (SFC 99). For more information about the function, please refer to Chapter 4.2.</td>
</tr>
<tr>
<td>2</td>
<td><strong>FB 1 “Counter” – (simulation)</strong></td>
</tr>
</tbody>
</table>

The function block calculates the following output variables as simulation values for the sequence displayed on the web page.

- **position**: calculates the package’s position.
- **lightBarrier**: checks whether the package is below the sensor.
- **counterPackages**: If the sensor detects a package, a count is incremented.

![Diagram of FB 1 “Counter”]
<table>
<thead>
<tr>
<th>No.</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>FC 1 “WebCom” – (web communication)</td>
</tr>
</tbody>
</table>

The function initially converts the simulation values of FB 1 to a “webString” variable. The “webserverString” string variable follows from this variable.

If you want to transfer additional values, you must extend the string in this block.

The TIA Portal system function “HTA” converts the following variable types to an ASCII string.

- SINT
- INT
- LINT
- DINT

Example:
```
// Convert DINT to ASCII-String
$tempRetVal := HTA(IN := #position, N := 4, OUT => #tempWebHost);
```

If you want to convert variables of the REAL or LREAL type, make sure to first convert them to DWORD or LWORD. Then convert these variables to DINT or LINT.

A bit is converted to an ASCII string using an IF statement.

General:
```
IF bool THEN
  char := '1';
ELSE
  char := '0';
END_IF;
```

Example:
```
// Convert BOOL to ASCII-String
IF #lightBarrier THEN
  $tempVal := #TRUE;
ELSE
  $tempVal := #FALSE;
END_IF;
```

The “CONCAT” function concatenates two ASCII strings.

```
// Add variable 1 to variable 2
$tempWebHost := CONCAT(IN1 := $tempWebHost, IN2 := $tempVal);
```

Example:
```
$tempWebHost = '0000002A50029E'
CONCAT = '0000002A50029E'
$tempWebHost = '029E'
```

### 18.2.2 User-defined web page structure

The diagrammatic overview below shows the basic file structure of the user-defined page.

This description focuses on the JavaScript file’s code elements, which makes basic knowledge of HTML necessary for this application example.

**User-defined page structure**

**Figure 18-4: Diagrammatic overview of the user-defined page**

<table>
<thead>
<tr>
<th>No.</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td><strong>DB 1 “Plc2Web”</strong></td>
</tr>
</tbody>
</table>

DB1 contains all variables that are sent from the CPU to the user-defined page or read.
The data block also contains the concatenated string variable, “webserverString”.

<table>
<thead>
<tr>
<th>No.</th>
<th>Name</th>
<th>Data type</th>
<th>Start value</th>
<th>Monitor value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Static</td>
<td>String</td>
<td></td>
<td>'0000024E0029E'</td>
</tr>
<tr>
<td>2</td>
<td>webserverString</td>
<td>String[254]</td>
<td></td>
<td>FALSE</td>
</tr>
<tr>
<td>3</td>
<td>lightBarrier</td>
<td>Bool</td>
<td>false</td>
<td>FALSE</td>
</tr>
<tr>
<td>4</td>
<td>position</td>
<td>Dint</td>
<td>686</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>counterPackages</td>
<td>Int</td>
<td>670</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>speed</td>
<td>Int</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>age</td>
<td>Int</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>name</td>
<td>String</td>
<td>'Alex'</td>
<td></td>
</tr>
</tbody>
</table>

**JSON files**

- dataWrite.json
- dataRead.json

**HTML file**

- Demo18.html

**JavaScript files**

- Demo18.js
- S7_framework.js
- jquery-2.1.3.min.js

**Library from Siemens**

- Library from third-party manufacturer

**Controller**
Demo18.html

All JavaScript files are integrated into the “Demo16.html” file.

Demo18.js

The “Demo18.js” JavaScript file controls read/write from/to the controller by calling functions from the “S7_framework.js” library.

S7_framework.js

The JavaScript file contains a number of functions for converting and processing data from a SIMATIC S7.
For more information about the S7 Framework, please refer to Chapter 4.3.
The Demo18.js file uses S7 Framework functions to transfer data via the JSON files.

dataRead.json and dataWrite.json (JSON)

The files are used for reading/writing the concatenated string using the functions within the S7 Framework.

jquery-2.1.3.min.js (jQuery library)

The jQuery library file allows more efficient programming with JavaScript. For more information about the jQuery library, please refer to Chapter 4.3.
### 18.2.2.1 Demo18.html structure

<table>
<thead>
<tr>
<th>No.</th>
<th>Code description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The JavaScript files used are integrated into the HTML document’s <code>&lt;head&gt;</code> with the below syntax:</td>
</tr>
</tbody>
</table>

```html
<script type="text/javascript" src="script/jquery-2.1.1.min.js"></script>
<script type="text/javascript" src="script/S7_framework_0.1.7.js"></script>
<script type="text/javascript" src="script/demo18.js"></script>
```

The “src” property specifies the respective file path.

### 18.2.2.2 Demo18.js structure

<table>
<thead>
<tr>
<th>No.</th>
<th>Code description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Determining the CPU type (S7-1200 or S7-1500)</td>
</tr>
</tbody>
</table>

In the “Demo12.js” JavaScript file, a function to determine whether an S7-1200 or S7-1500 is used is executed in “$(document).ready(function(){

```javascript
if (plcType == "1200" && plcType != "1500")
{
    $.ajax({type:"GET",url:"/.../ForCell/Intro.xml",data:"",dataType:"text"})
    .done(function(webpageData){
        var search12 = webpageData.search("CPU12");
        var search15 = webpageData.search("CPU15");
        if (search12 >= 0) {
            plcType = "1200";
            $.init();
        } else if (search15 >= 0) {
            plcType = "1500";
            $.init();
        }
    
    });
```

Alternatively, you can directly call the functions of “$.init” in “$(document).ready(function(){

Replace “plcType” with “1200” or “1500”, including quotation marks.

| 2   | This function transfers the CPU type to the S7 Framework. |

```javascript
$.init = function()
    S7Framework.initialize(plcType, "");
```

| 3   | The “S7Framework.readData” function reads the PLC tags specified in the selected json file (dataRead.json). When the function has been executed, the specified event handler (updateValues) is run. |

```javascript
S7Framework.readData("script/dataRead.json", ...
... "init read data",updateValues);
```

| 4   | The “updateValues” function further processes the read values. The values are transferred to the “values” array. First value = values[0] Second value = values[1] etc. |

```javascript
function updateValues(values)
```
### dataRead.json and dataWrite.json structure

<table>
<thead>
<tr>
<th>No.</th>
<th>Code description</th>
</tr>
</thead>
</table>
| 1   | **dataRead.json**  
The “dataRead.json” file contains the information for reading the string.  
The string to be read is defined: **Plc2Web.webserverString**  
The string structure is defined: Length; type; other string variables  
```
{
  "val" : ":=Plc2Web.webserverString:",
  "len" : ":8;1;4",
  "typ" : ":2;0;2",
  "str" : ""
}
```

DINT (DINT) + BOOL (BOOL) + INT (INT)  

“val” corresponds to the string to be read.  

“len” describes the “length” of the variable (variable type)  
1 = 1 bit => corresponds to: BOOL  
2 = 8 bits => corresponds to: BYTE  
4 = 16 bits => corresponds to: INT / WORD  
8 = 32 bits => corresponds to: DINT / DWORD / REAL  
16 = 64 bits => corresponds to: LINT / LWORD / LREAL  

“typ” describes the variable type (defined in the S7 Framework)  
0 = BOOL;  
1 = UINT;  
2 = INT/DINT;  
3 = REAL;  
4 = LREAL;  
5 = STRING;  

“str” is used for loading variables that were not converted to the string. |
| 2   | **dataWrite.json**  
The file contains the parameters of the string and the UDP commands of the PLC tags to be written.  
The information for writing the string corresponds to the information for reading, see above.  
```javascript
<AWP_In_Variable Name="Plc2Web.speed" -->
<AWP_In_Variable Name="Plc2Web.name" -->
<AWP_In_Variable Name="Plc2Web.age" -->
{
  "val" : ":=Plc2Web.webserverString:",
  "len" : ":8;1;4",
  "typ" : ":2;0;2",
  "str" : ""
}
```
19 Installation

19.1 Installing the hardware and software

Installing the hardware

The following figure shows the hardware configuration for the examples.
The PC (web browser) is connected to the CPU (web server) via Industrial Ethernet
using the PN interface.
The SIMATIC S7 controllers were used separately as the examples use only one
IP address for the S7 controller.

Figure 19-1: Hardware configuration of the examples

![Hardware Configuration Diagram]

**Note**  
Follow the installation and connection guidelines provided in the appropriate
manuals and installation instructions.

Installing the software

Table 19-1: Installing the software packages for the examples

<table>
<thead>
<tr>
<th>No.</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Install SIMATIC STEP 7 Professional (TIA Portal).</td>
</tr>
</tbody>
</table>
| 2.  | Install a tool for creating the user-defined page, e.g. MS Expressions,
Notepad++, on the PC that you want to use to create the user-defined page. |
| 3.  | Install a web browser, e.g. Internet Explorer or Firefox, on the PC that you want
to use to access the CPU’s web pages. |
## 19.2 Installing the application example

Table 19-2: Procedure for using the examples

<table>
<thead>
<tr>
<th>No.</th>
<th>Action</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Unzip the “68011496_examples_for_S7WebServer_CODE_v11.zip” file to your project directory.</td>
<td>-</td>
</tr>
<tr>
<td>2.</td>
<td>Start SIMATIC STEP 7 V14.</td>
<td>-</td>
</tr>
<tr>
<td>3.</td>
<td>In SIMATIC STEP 7 V14, open the project.</td>
<td>-</td>
</tr>
<tr>
<td>4.</td>
<td>Select the desired program. The programs are numbered by the chapter numbers in the documentation.</td>
<td>-</td>
</tr>
<tr>
<td>5.</td>
<td>Go to the Device view.</td>
<td>-</td>
</tr>
<tr>
<td>6.</td>
<td>If you want to use a different CPU, replace the CPU in the hardware catalog.</td>
<td>-</td>
</tr>
<tr>
<td>7.</td>
<td>In the CPU properties of the Ethernet interface, assign the IP address of your CPU.</td>
<td>-</td>
</tr>
<tr>
<td>8.</td>
<td>Select the CPU and download the entire project to the CPU.</td>
<td>-</td>
</tr>
<tr>
<td>9.</td>
<td>Start a web browser and use the IP address to open the web page of your CPU. For more information, please refer to the “Use” chapter of the examples.</td>
<td>-</td>
</tr>
</tbody>
</table>
20 Internet links

This table provides a selection of links for related information.

Table 20-1: Internet links

<table>
<thead>
<tr>
<th>No.</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>/1/</td>
<td>Siemens Industry Online Support</td>
</tr>
<tr>
<td></td>
<td><a href="http://support.automation.siemens.com">http://support.automation.siemens.com</a></td>
</tr>
<tr>
<td>/2/</td>
<td>Reference to the entry</td>
</tr>
<tr>
<td>/3/</td>
<td>HTML, CSS, JavaScript,</td>
</tr>
<tr>
<td></td>
<td><a href="http://www.w3schools.com">http://www.w3schools.com</a></td>
</tr>
<tr>
<td>/5/</td>
<td>SVG, w3schools</td>
</tr>
<tr>
<td></td>
<td><a href="http://www.w3schools.com/svg/default.asp">http://www.w3schools.com/svg/default.asp</a></td>
</tr>
<tr>
<td>/6/</td>
<td>jQuery Flot</td>
</tr>
<tr>
<td></td>
<td><a href="http://www.flotcharts.org/">http://www.flotcharts.org/</a></td>
</tr>
<tr>
<td>/7/</td>
<td>jQuery</td>
</tr>
<tr>
<td></td>
<td><a href="https://jquery.com/">https://jquery.com/</a></td>
</tr>
</tbody>
</table>

21 History

Table 21-1: Document history

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Modifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1.0</td>
<td>08/2015</td>
<td>First version</td>
</tr>
<tr>
<td>V 2.0</td>
<td>04/2017</td>
<td>The following examples were revised:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Change language</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Creating a graph</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The following examples were added:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Creating display elements</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Changing pictures using JavaScript</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Integrating a button for restarting the CPU</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Integrating the login option</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• High-performance data transfer using variables converted to strings</td>
</tr>
</tbody>
</table>