SIEMENS

SIMATIC NET

S7Beans / Applets for IT-CPs

Programming Tips

Supplement to the Documentation of the IT-CPs
CP 243-1 IT
CP 343-1 IT
CP 443-1 IT

Release 06/2003
C79000–G8976–C180–01

Classification of Safety-Related Notices

This manual contains notices which you should observe to ensure your own personal safety, as well as to protect the product and connected equipment. These notices are highlighted in the manual by a warning triangle and are marked as follows according to the level of danger:

Danger

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

Warning

indicates that death or severe personal injury **can** result if proper precautions are not taken.

Caution

with a warning triangle indicates that minor personal injury can result if proper precautions are not taken.

Caution

without a warning triangle indicates that damage to property can result if proper precautions are not taken.

Notice

indicates that an undesirable result or status can occur if the relevant notice is ignored.

Note

highlights important information on the product, using the product, or part of the documentation that is of particular importance and that will be of benefit to the user.

Trademarks

SIMATIC®, SIMATIC HMI®and SIMATIC NET®are registered trademarks of SIEMENS AG.

Third parties using for their own purposes any other names in this document which refer to trademarks might infringe upon the rights of the trademark owners.

Safety Instructions Regarding your Product:

Before you use the product described here, read the safety instructions below thoroughly.

Qualified Personnel

Only **qualified personnel** should be allowed to install and work on this equipment. Qualified persons are defined as persons who are authorized to commission, to ground, and to tag circuits, equipment, and systems in accordance with established safety practices and standards.

Correct Usage of Hardware Products

Note the following:

Warning

This device and its components may only be used for the applications described in the catalog or the technical description, and only in connection with devices or components from other manufacturers which have been approved or recommended by Siemens.

Correct and safe operation of this product requires proper transport, storage, installation, and assembly as well as careful operator control and maintenance.

Before you use the supplied sample programs or programs you have written yourself, make certain that no injury to persons nor damage to equipment can result in your plant or process.

EU Directive: Do not start up until you have established that the machine on which you intend to run this component complies with the directive 89/392/EEC.

Correct Usage of Software Products

Note the following:

Warning

This software may only be used for the applications described in the catalog or the technical description, and only in connection with software products, devices, or components from other manufacturers which have been approved or recommended by Siemens.

Before you use the supplied sample programs or programs you have written yourself, make certain that no injury to persons nor damage to equipment can result in your plant or process.

Prior to Startup

Before putting the product into operation, note the following warning:

Disclaimer of Liability

We have checked the contents of this manual for agreement with the hardware and software described. Since deviations cannot be precluded entirely, we cannot guarantee full agreement. However, the data in this manual are reviewed regularly and any necessary corrections included in subsequent editions. Suggestions for improvement are welcomed.

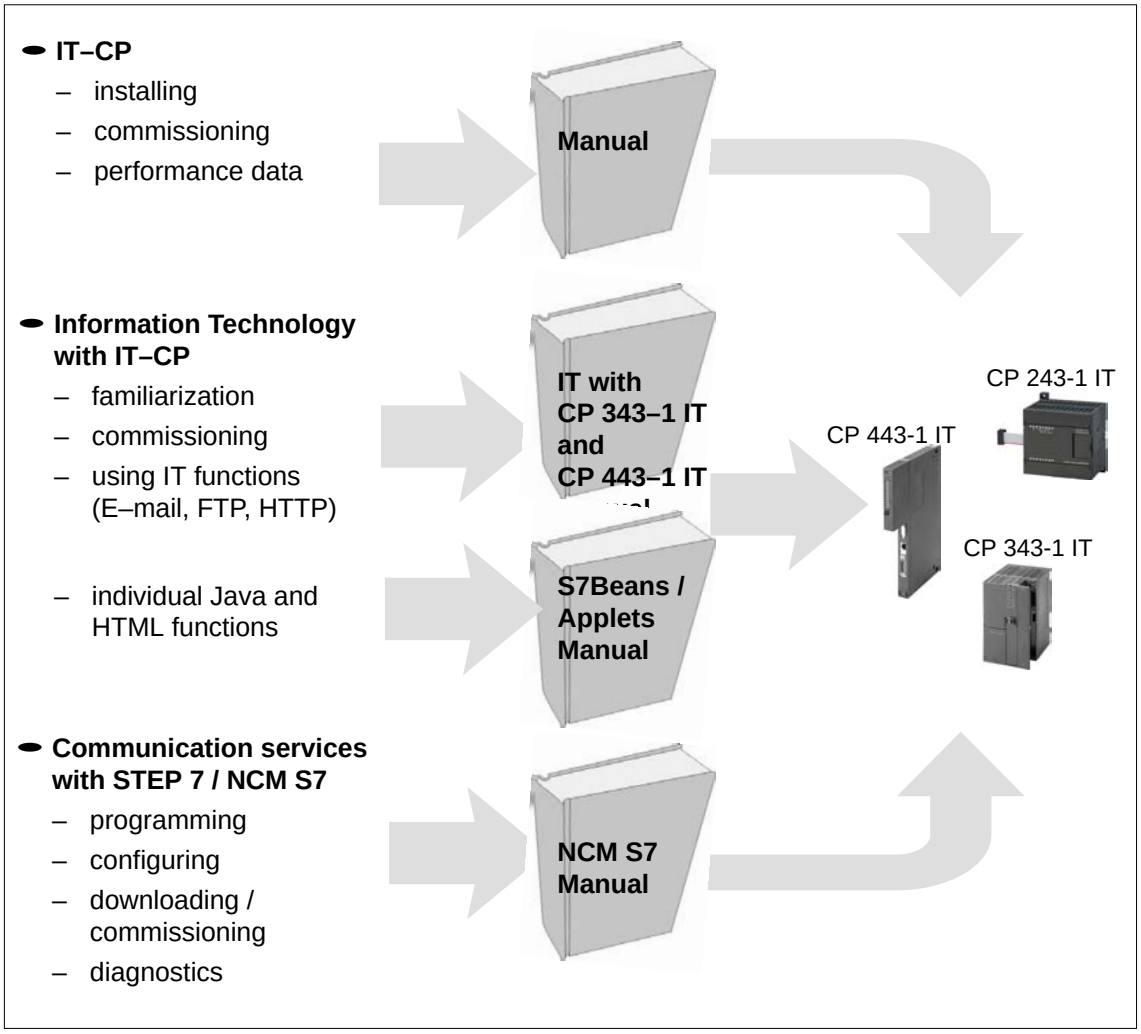
Siemens AG
Automation and Drives
Industrial Communication
Postfach 4848, D-90327 Nürnberg

Subject to technical change.

Preface / References

Manuals on the topic of IT in SIMATIC

Information technology with IT–CPs in SIMATIC is described in the following manuals:



You will also find these documents on the Manual Collection CD. This symbol is used at various points in the text to indicate that there is additional information and examples on the Manual Collection CD.

Validity of these Programming Tips

This document is valid

- for the CP 443-1 IT for the SIMATIC S7-400

  and

  for the CP 343-1 IT for the SIMATIC S7-300

  – with the required configuration software STEP 7 version 5.0 SP3 or higher with the NCM S7 for Industrial Ethernet option

- for the CP 243–1 IT for the SIMATIC S7–200

  – with the required configuration software STEP 7 MicroWin version 3.2 SP1 or higher

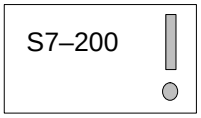
- S7 BeansAPI version 2.5 or higher
- JBuilder version 3.0 or higher

Note

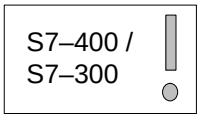
Notice

Please note that the S7–200 differs in some technical aspects from the S7–300/S7–400, where necessary in the description, these aspects are highlighted by the following symbols in the margin.

Text indicated by this symbol applies only to the S7–200.



Text indicated by this symbol applies only to the S7–300/S7–400 .



References /.../

References to further documentation are specified with documentation numbers in slashes /.../. Based on these numbers, you can check the title of the documentation in the list of references at the end of the manual.

❑

Contents

You will fnd the entire manual and the programming tips on the Manual Collection CD. This symbol indicates that there is additional information and examples relating to the topic on the Manual Collection CD.

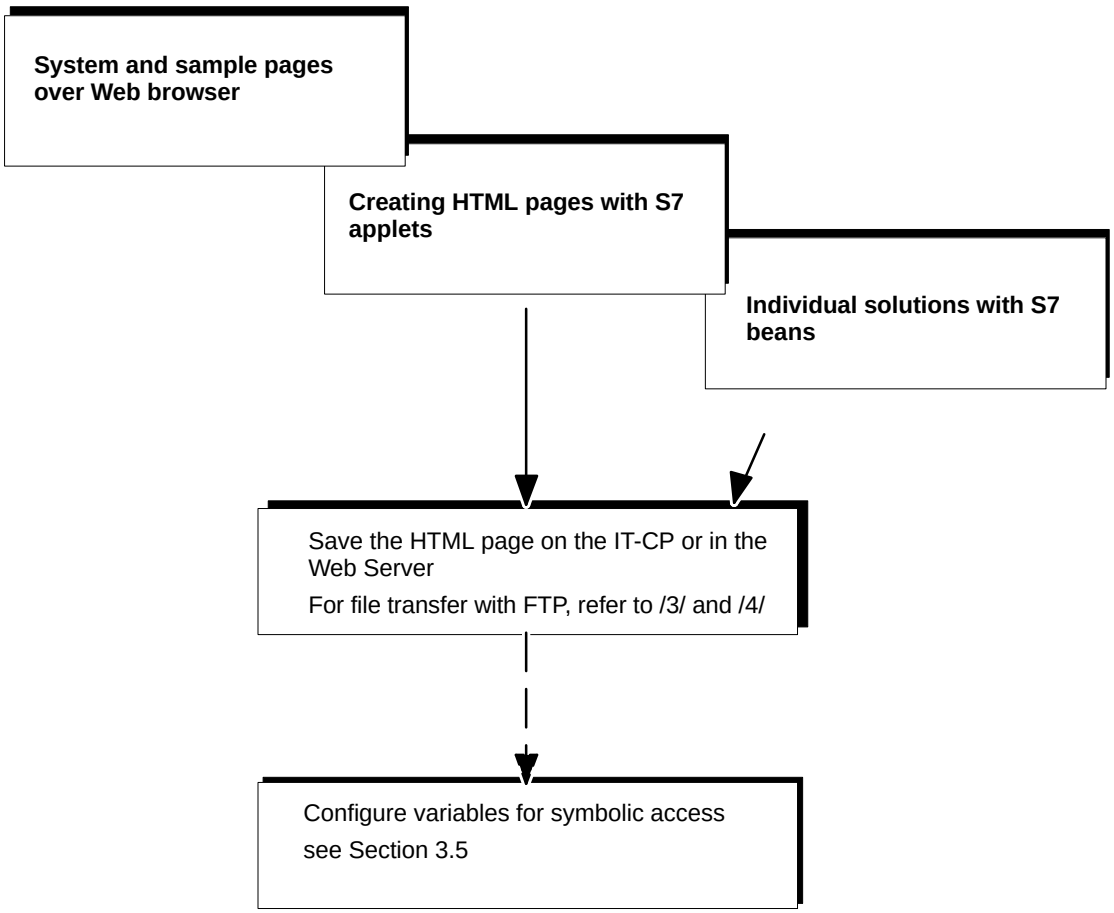
1 Creating HTML Pages – Overview

Multilevel Concept

The IT–CP provides several levels to implement device and process data monitoring with HTML pages:

How to...

...create or adapt your own individual HTML pages:



- System and sample pages over Web browser

  You want to use the options of the HTML process control predefined for the IT-CP without extensive programming.

  The possibilities available to you are introduced in /3/.

- Creating HTML pages with S7 applets

  The IT–CP provides you with predefined S7 applets with which you can create HTML pages and adapt them to your task.

The calls and call parameters are described in this manual on the S7 applets / beans.

- Individual solutions with S7 beans

  You want to use graphics options adapted to your application and create more complex applets.

  You not only want to display your process data in the plant pictures but also want to use the data, for example, for evaluation in a database.

  You can achieve this with the following options:

  – Create application–specific Java applets and Java applications and use predefined S7 beans, Java beans of other manufacturers and your own Java source text.

  – Create Java source code; use application-specific applets, Java beans and the supplied S7 beans.

2 Web Browsers

2.1 Contacting the IT-CP using a Web Browser

What is Required?

To access the HTML pages on the IT-CP or on the Web server you require a Web browser, for example Netscape Navigator or Internet Explorer. The Web browser must meet the following requirements:

- JDK (Java Development Kit) 1.1.X is supported; For further information refer to /8/.

The Netscape Navigator and Internet Explorer meet these requirements. Other Web browsers with the same range of functions can also be used.

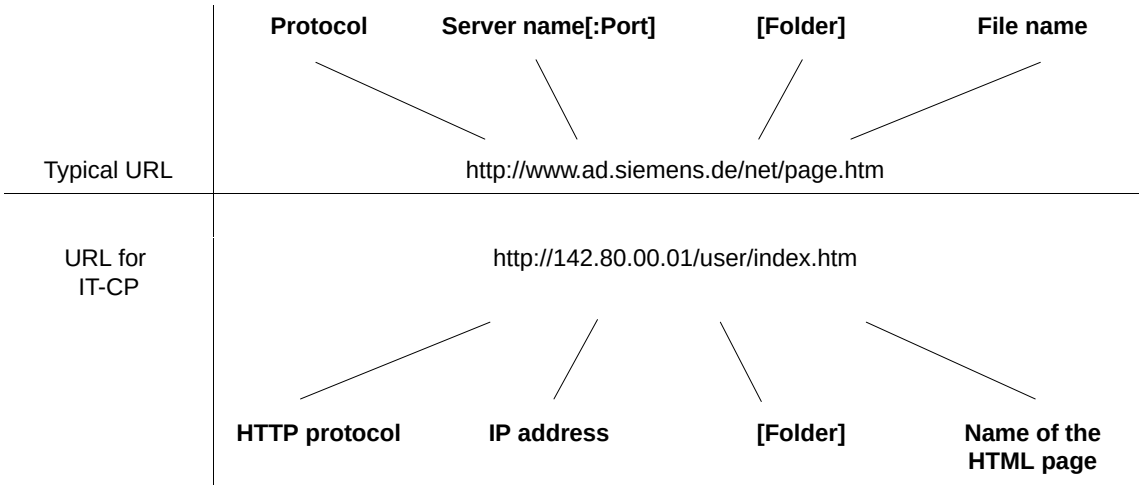
Other Web browsers meet these requirements only with certain restrictions. You require a plug-in component corresponding to the Java reference implementation of a SUN Java Virtual Machine.

Note the other information including the versions of the products mentioned here in the manuals /1/ on the Manual Collection CD.

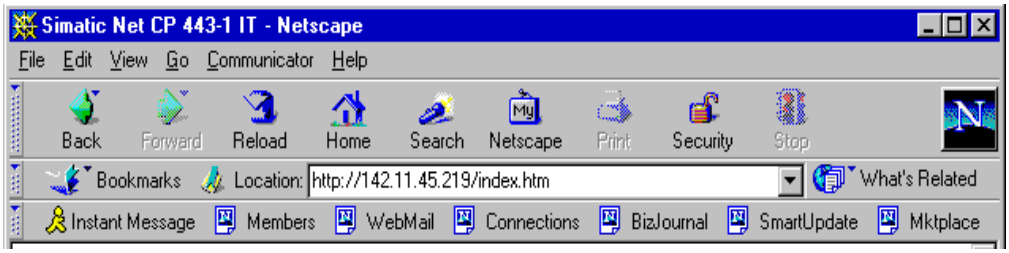
Further information and additional programs required are available on the Internet (refer to Appendix B).

URL: Uniform Resource Locator

In the World Wide Web, addressing using URL has become standard. You can also contact the IT-CP with your Web browser using the URL. This URL can have almost any complexity but consists in principle of four essential parts. The following schematic illustrates the structure (typical URL) and shows the contents for calling IT-CPs.



When accessing the IT-CP using a Web browser, use the HTTP protocol to address the Web server on the IT-CP:



You inform the CP of the IP address during configuration with STEP 7. If you have an attachment from Industrial Ethernet to your intranet or to the Internet, the CP can be contacted using the IP address in the intranet or Internet.

A detailed description of the structure of the IP address and the options of creating subnets or subnet masks is beyond the scope of this manual. You will find more detailed information in the STEP 7 online help and in the documentation listed in the references, for example in /7/.

2.2 Settings in the Web Browser

General

Before you can access the IT-CP using your Web browser, you must make or at least check certain settings. The settings are explained below based on the example of the Netscape Navigator.

The settings shown here have been selected so that the execution of the S7 applets and S7 beans (JavaBeans) used on the IT–CP is possible.

Settings in the Netscape Navigator/Communicator

In the Netscape Navigator, most settings are made in the “Preferences”. Select the menu command **Edit ▸ Preferences... (Edit ▸ Preferences...)**

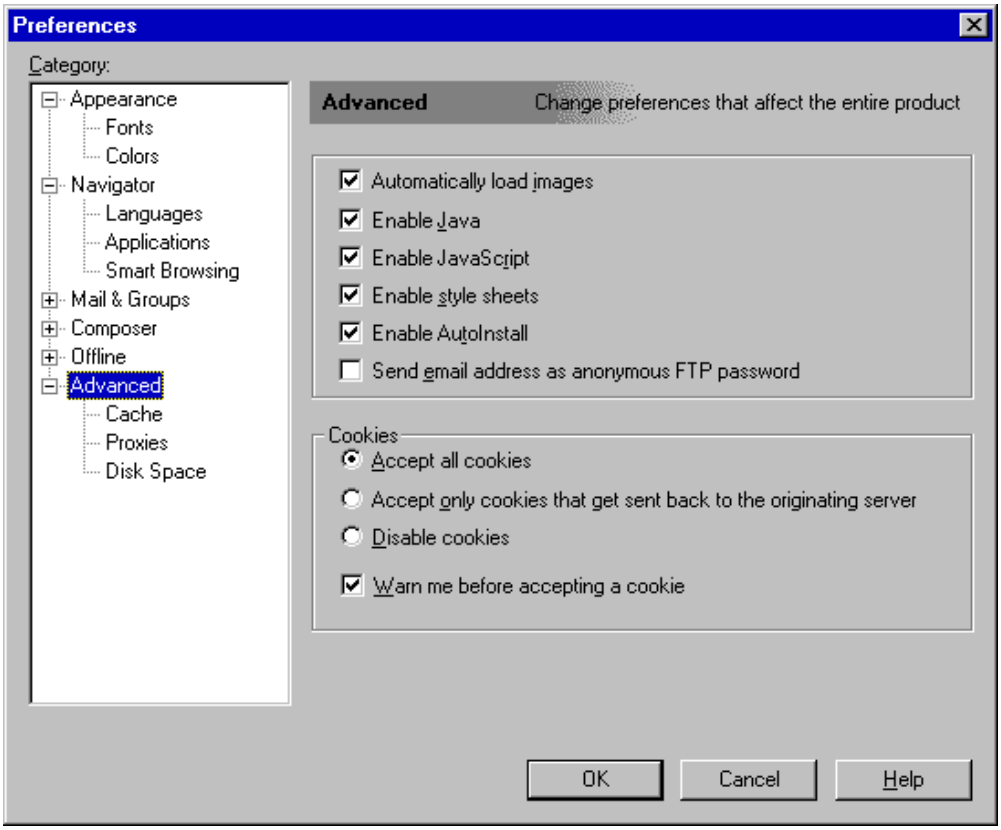


Figure 2-1

The following settings are recommended:

- Starting the Java Interpreter

  By selecting ”Enable Java” in the ”Category – Advanced” you start the Java Interpreter when an applet first occurs in an HTML page and permit execution of the applet.

Note

If you make this setting when an HTML page is already loaded, the Java applets contained in the page will not be executed even after making the setting. The HTML page must first be loaded again so that the Java Interpreter is started and the Java applets executed.

- Setting a Proxy Server

  For more information, check with your system administrator.

Settings in the Internet Explorer

- Starting the Java Interpreter

  You will find the functions for the Java applications by selecting the menu command ”Options – Internet Options” and the ”Advanced” tab under the entry “VM”.

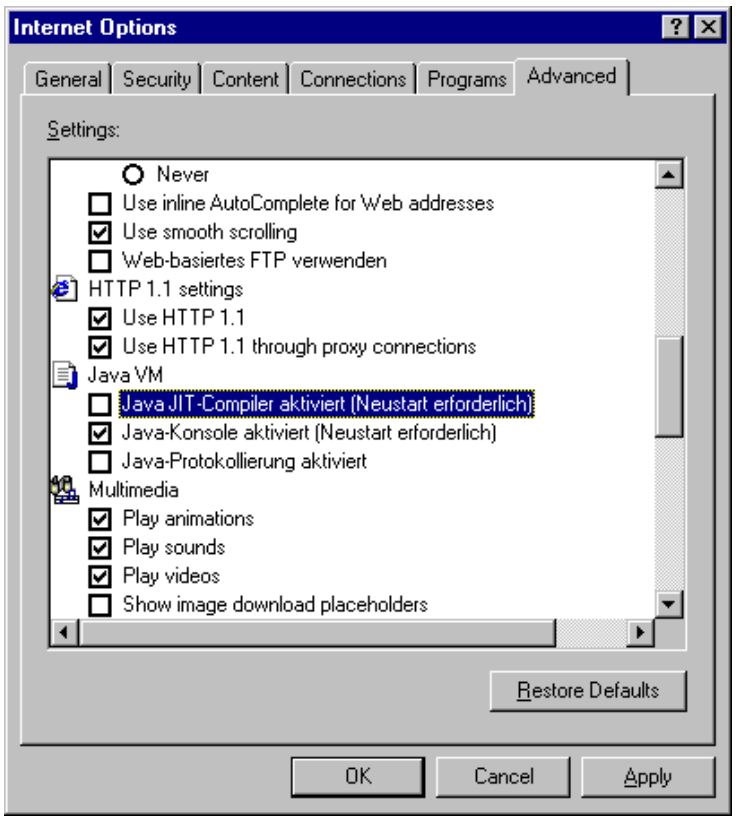


Figure 2-2

- Setting a Proxy Server

  For more information, check with your system administrator.

Starting the Java Console

To track the execution of the Java applet and to get information about problems, you can start the Java Console. Select

- the menu command **Communicator ▸ Extras ▸ Java Console (Communicator ▸ Java)** in the Netscape Navigator;
- In the Internet Explorer please use the option in the Internet Options dialog box in the ”Advanced” tab as shown above. To open the Java console, use the menu command **View ▸ Java Console**

Note:

With the IE, the Java console must first be activated before it can be used. To do this, enable the option ”Java Console enabled” (”Options –> Internet Options” menu command, ”Advanced” tab, ”Microsoft VM”).

❑

3 HTML Pages on the IT-CP

This chapter answers the following questions:

- How are HTML pages created that can access information on the S7 station?
- What are S7 applets and how are they used in HTML pages? What do I need to remember?
- How do I store my own HTML pages?
- How do I obtain a graphic representation of process information?
- How are HTML pages checked and tested?

3.1 Creating HTML Pages for the IT-CP

Uses

With your own HTML pages, the following possibilities are open to you:

- Process visualization in the Web browser adapted to your particular plant
- Process data represented numerically or graphically in the Web browser
- The ability to include the results of status queries in the display
- The querying and representation of process data in one HTML page on several S7 stations and on distributed systems.

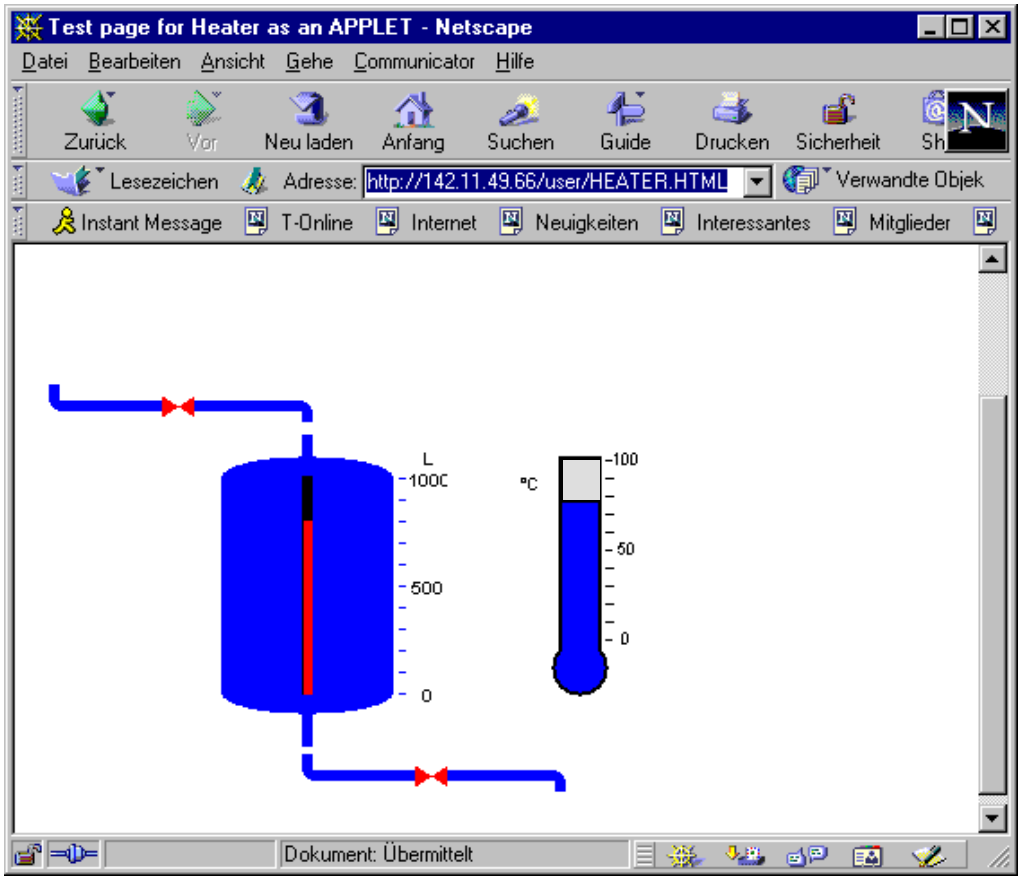


Figure 3-1

HTML Editor

To create your own HTML pages, you require an editor. It is easier to keep to the syntactic conventions for HTML pages if you use a suitable HTML editor. These normally allow the input of formatted text and the inclusion of graphics. The conversion to HTML syntax then takes place in the background. It is also normally possible to change the view and make your input directly in the HTML form.

Examples:

- AOLpress

  A powerful HTML editor that can be obtained as shareware and that has all the features listed above.

- Netscape Composer

  This is also an HTML editor providing functions comparable with AOLpress. The Composer is included with the Netscape Communicator.

- Front Page from Microsoft (can only be used with restrictions)

  A powerful HTML editor. Certain options can however lead to specific HTML codes that can only be edited with the Internet Explorer from Microsoft.

  To obtain HTML code with FrontPage that can be processed with as many browsers as possible, you should make the following settings in FrontPage (”Tools –> Page Options” menu command, ”Compatibility” tab):

  – The ”Enabled with Microsoft FrontPage Server Extensions” option must not be activated.

  – The ”Active Server Pages” option must not be activated.

  – The options ”ActiveX controls”, ”VBScript”, ”JavaScript”, ”Dynamic HTML”,and ”Frames” as well as the various versions of the ”Cascaded Style Sheets (CSS)” depend on the browser and their use is therefore restricted.

  – The ”Java applets” option is necessary to allow use of the S7BeansApi and the applets created from it and must be activated.

  Once the pages have been created, they can be transferred to the IT-CP using an FTP client or the FrontPage command ”File –> Publish Web”. The following folders and subfolders are created by FrontPage but are not required on the IT-CP:

  – ”_private”

  – ”_vti_cnf”

  – ”_vti_pvt”

  – ”_vti_bin”

  – ”_vti_log”

  – ”_vti_txt”

  In Frontpage, these folders can be excluded from the transmission by Frontpage using ”View – Reports – Publish Status”.

S7 applets are applets for SIMATIC S7

The IT-CP provides several applets with which you can access the controller from the Web browser on your PC. You do not need to be familiar with Java to use these S7 applets. By following the instructions below, you will be able to integrate the calls in your HTML page simply and quickly.

Extended Access and Display Options – The JavaBeans Concept

The JavaBeans concept allows you to create objects (Java components) and to link them simply to executable programs.

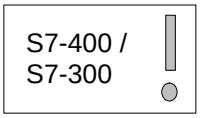
There is an S7 beans class library available for the IT-CP (S7BeansAPI). The object classes contained in this library can be used for object-oriented access to a variety of information on the SIMATIC S7 and for graphic display of process variables.

The S7 beans class library provides an open interface allowing you to extend process data evaluation for example with databases, table calculation or management information systems.

Organizing Files – Resources of the IT-CP

The IT-CP has memory available for storing your HTML pages. For further information refer to the manual of the IT-CP/1/.

Please note the information in the readme.htm file on the IT-CP.



The simplest way to open the readme.htm file is by clicking the “Information” link on the home page of the IT-CP.

This contains information about the meaning and purpose of the shipped files. You can then decide which files might be useful for your application. Using FTP functions (see Section LEERER MERKER), you can organize the files on the IT-CP to suit your requirements.

3.2 Designing HTML Pages – A Few Essentials

A Word of Introduction

There is a considerable body of excellent literature dealing with the design of HTML pages. Check the recommended literature dealing with the topics of Web, HTML etc. in the appendix to this manual. This manual itself is deliberately restricted to explaining how you can include the functions supplied with the IT-CP in your HTML application.

With the information here, you should be in a position to create and run HTML pages and use the S7 applets without needing a thorough study of HTML techniques.

Experienced authors of HTML pages will be able to use the information about assigning parameters for S7 applets in the following sections directly. For the less experienced user, we have included a certain amount of information on the topic. This information can then be extended by reading the literature mentioned above.

Planning the Structure

From the very beginning, you should be clear in your own mind about your document and page structure. HTML allows you to jump from topic to topic in the HTML pages displayed by the Web browser and, in the case of the IT-CP, from controller to controller or plant to plant. Basically this means that the scope of the HTML pages depends on the links that you specify when you create the pages. On the other hand, the links themselves also depend on the storage of the HTML documents.

Linking Documents (HTML Pages)

HTML pages are connected by links that are structured as follows:

- URLs with absolute address information

  Example:
  <A HREF="http://www.ad.siemens.de/net/index.htm">ID_text</A>

- URLs for simplified link information with relative addresses:
  - Use of relative URLs: Generally, these contain only the name of the folder and the file or even only the file name if the HTML page called is in the same folder as the calling HTML page.

    Example:
    <A HREF="processpic.htm">ID_text</A>

  - Use of server-related URLs: These addresses begin with "/" and indicate that the HTML page is on the same server as the calling HTML page.

    Example:
    <A HREF="/pics/processpic.htm">ID_text</A>

Designing HTML Pages

To design pages that are both practical and attractive, HTML provides you with various description elements. The literature provides suitable information usually under the following topics:

- Displaying Tables

  The table is an important tool in structuring information. In particular when designing HTML pages, tables are suitable for compensating certain weaknesses of HTML formats. Displaying HTML text in columns is, for example only possible if you use tables.

- Inserting Pictures

  The use of pictures in HTML pages is a topic dealt with extensively in the literature on HTML. The image file formats you can use are GIF and JPG.

- HTML Forms (can only be used on the IT-CP in conjunction with JavaScript)

  Forms are required when the user is intended to store information in the system in a standardized form. HTML provides various structure elements for interaction with the user. Form functions can also be useful for entering process data.

- Using Format Templates

  The use of format templates allows generally valid formats to be specified and used in the HTML documents. This procedure is known from publishing systems such as MS Word or Word Perfect.

  It is adequate to simply say that there are various ways of linking format templates with HTML documents.

- Creating Frames

  Using frames, you can divide HTML pages into several areas. This can

significantly increase the clarity. You can, for example, use frames to make sure that the navigation menu remains visible while new areas are loaded.

Note also the information on the topic ”The number of applet instances in an HTML page is limited” in Section 3.4.

- Embedding Applets

  This is the main topic of this chapter which describes the use of the special S7 applets.

- Using Java Script

  With adequate experience, using Java Script can extend the function of HTML pages and the interaction with the user.

Note

The Internet itself is a treasure trove for HTML design. You can load HTML pages in your HTML editor at any time, save them and use them as a template for your own page design.

You can, for example, call up HTML pages you find particularly attractive in the HTML editor and then save them. If you do this, all the data of the pages, including the graphics are saved.

You should, however, remember that some HTML pages are protected by Copyright. You cannot, naturally, use these pages for your own page design.

HTML Editor – Examples/Recommendations

The requirements of an HTML editor were outlined in Section LEERER MERKER. Suitable HTML editors are recommended there.

S7 Applets

With the S7 applets, you incorporate process data display and the input of process data into your HTML page. The S7 applets are described in the following section.

3.3 Creating Your Own “Home Page”

Flexible Use of the IT-CP File System

The existing Start page provides you with basic functions that are adequate for a large number of requirements.

The IT-CP file system provides a flexible instrument for the presentation of functions and data adapted to your plant. By creating your own start page, you have the tool to extend the view to cover your entire plant or even further.

You can modify the existing start page or can replace it by your own home page.

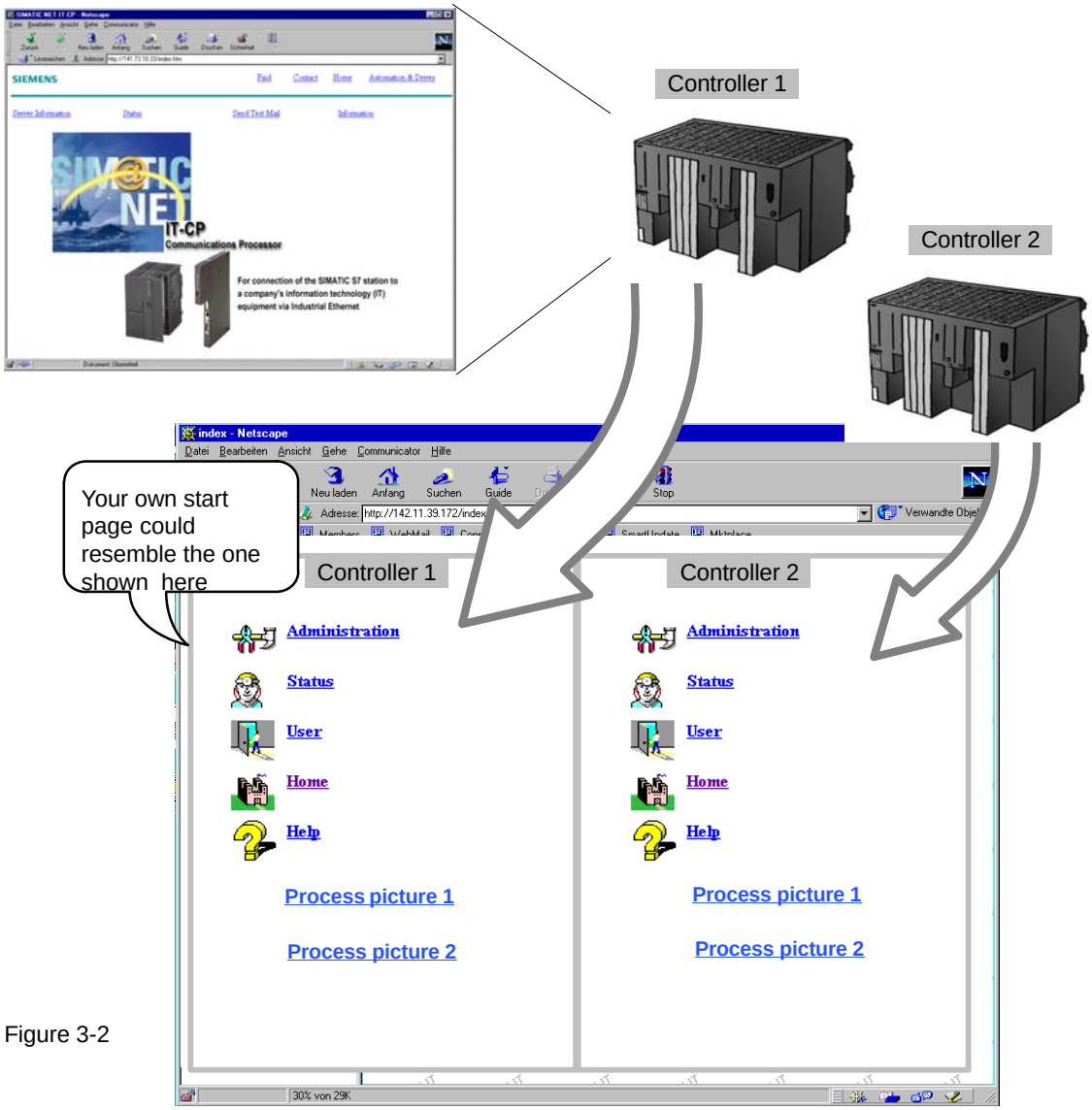


Figure 3-2

What to Do

If you want to start with the existing start page, load this in your HTML editor and add the additional instructions you require.

- The online option

  Load the HTML start page from your IT-CP in your HTML editor and save it for further editing locally on your PC.

- The offline option

You will also find the HTML start page on the Manual Collection CD. You can then adapt your start page regardless of whether you have access to the IT-CP and then download it to the IT-CP.

Points to Remember

Refer to the information in the manual of the IT-CP /1/ regarding the following points.

- The number of files that can be stored is limited by the size of the file system
- The number of characters in the URLs to be specified is limited.
- The length of the file names is limited.

Including S7 Applets

Flexible access to distributed HTML system pages is **one** aspect of designing the home page.

You have further opportunities for querying information if you include the S7 applets and S7 Beans in your HTML pages. This is explained in detail in later chapters.

Examples:

You will find examples of specific HTML pages on the Manual Collection CD.

S7-400 / S7-300

You will also find examples in the CP file system in the /examples/ folder.

Downloading HTML Pages

Use an FTP client and the file management functions as described in /3/ and /4/ to add to or replace the existing HTML pages.

3.4 S7 Applets

Meaning

S7 applets are special applets that allow read and write access to an S7 station via the IT-CP.

The Web browser in which the applet was started is responsible for execution of the applets. This activates the applet and assigns a frame to it within the current HTML page according to the parameter settings.

S7-400 / S7-300

The following example illustrates the situation where all the supplied S7 standard applets are used within one HTML page. You can see that the S7 applets in this case are embedded in an HTML table.

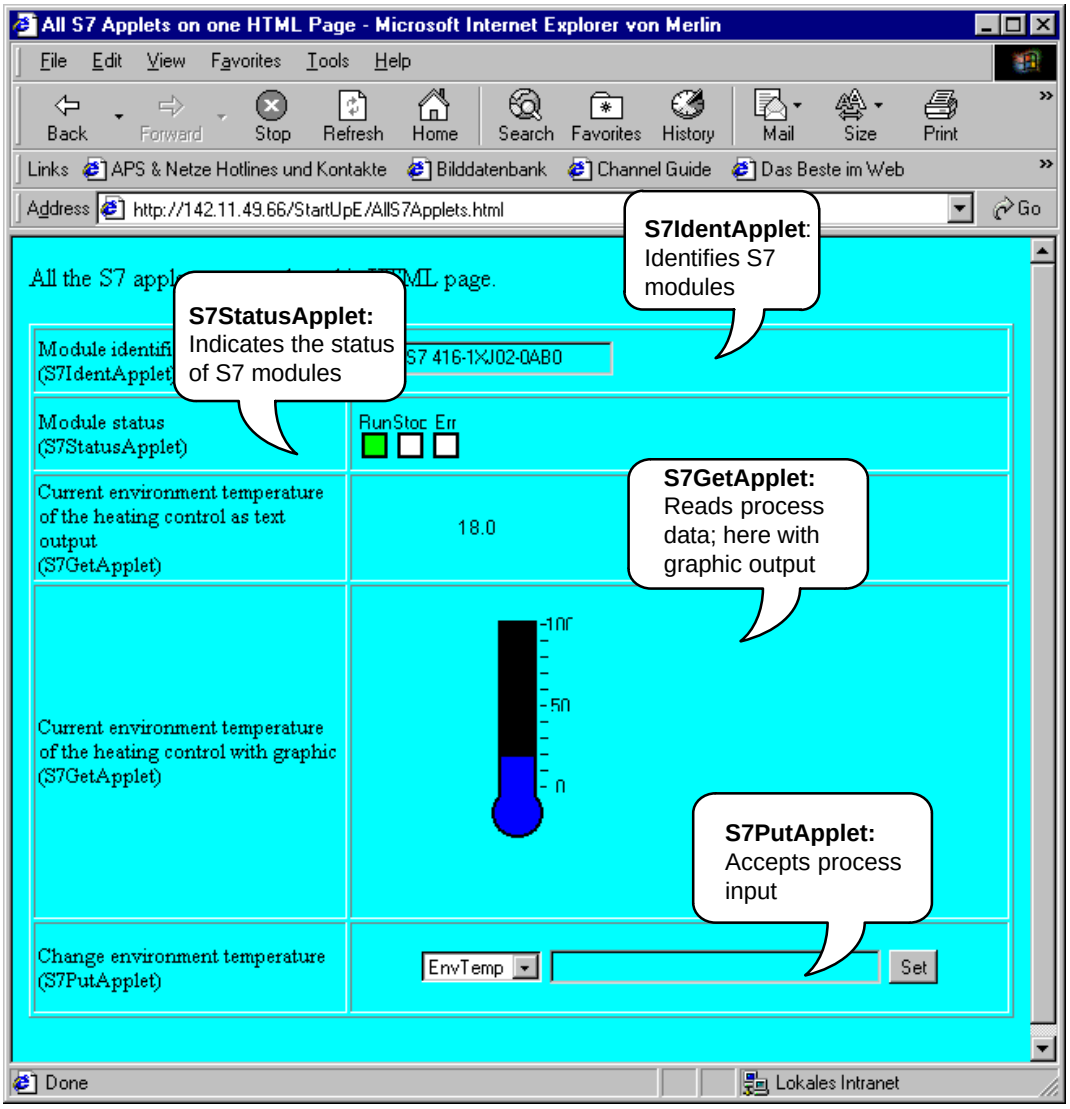


Figure 3-3

The table below contains further information. The chapters that follow describe in detail how to use and assign parameters for the S7 applets.

Table 3-1

S7 Applet	Meaning
S7IdentApplet (S7-300/400 only)	Identifies S7 modules based on the order number and version.
S7StatusApplet (S7-300/400 only)	Displays the status of S7 modules Example: Run/Stop
S7GetApplet	Reads process data cyclically, for example memory words or data in a data block. The process data • are addressed symbolically or in absolute form • can be displayed graphically. JavaBeans are used for graphic display.
S7PutApplet	Enters process data in the HTML pages and transfers/writes them to the controller, for example memory words or data in a data block. The process data are addressed in symbolic or absolute form.

The Interaction of the Components

The S7 applets transfer special read or write jobs to the IT-CP from the Web browser. When requested, the IT-CP passes on the jobs to the relevant module or CPU.

The S7 applets can display messages about actions and errors in the Java Console (see Section 3.6). These messages provide you with information about the current status of program execution.

The number of Applet instances in an HTML page is limited

Remember that the number of applets that can be called in an HTML page is limited. The possible number depends on the Web browser you are using and your system environment (for example the operating system). You will find this maximum number in the documentation of your Web browser.

Increasing complexity with S7 beans:

If you require more complex displays, you should use the options provided by the S7 beans class library. If you create your own applets, you can access a larger number of process variables using S7 beans. This will help you to avoid the restrictions relating to the number of applets.

3.4.1 Applet Call and Parameter Assignment

Calling S7 Applets in the HTML page

Like all Java programs, S7 applets have the file name extension ”class”. The applet call is embedded in the HTML page with a corresponding HTML tag (refer to the table below). The call to identify an S7 module in an S7 station is, for example, specified as follows in the applet tag:

<Applet CODE=”de.siemens.simaticnet.itcp.applets.S7IdentApplet.class” ...>

This assignment specifies the name or the address of the file with the applet. The notation used here indicates a relative address. All the applets are put together to make a jar file. If the CODEBASE attribute is also used in the call, the applet is located in the folder specified by CODEBASE.

Example:

The following call identifies an S7 module located in Rack 0 in Slot 3 of an S7 station. The information read is displayed on the HTML page in black script on a green background.

```
<APPLET CODE=“de.siemens.simaticnet.itcp.applets.S7IdentApplet.class”
CODEBASE=“/applets/”

ARCHIVE=“s7applets.jar, s7api.jar” NAME=“s7_MLFB” WIDTH=150
HEIGHT=35>

<PARAM name=”RACK” value=0>
<PARAM name=”SLOT” value=3>
<PARAM name=”BACKGROUNDCOLOR” value=”0x00FF00”>
 </APPLET>
```

You will find further examples of applications in the descriptions of the S7 applets themselves.

General Parameter Assignment

Apart from the names of the S7 applets, you must specify certain general attributes and parameters. In addition to the general attributes and parameters required for each S7 applet, there are also function-dependent attributes and parameters. These are described with the S7 applets themselves.

The following tables show you the HTML tags, the attributes and parameters that are required with most S7 applets:

Table 3-2 General HTML Tags when Using Applets

HTML Tag and Attribute	Meaning
<APPLET...>...</APPLET>	This tag embeds an applet in an HTML document. The applet is specified by attributes and parameters. Example, see above.
<PARAM name="..." value="...">	This tag identifies applet parameters. Each parameter is identified by a name and each parameter is assigned a value. Example: <PARAM name="RACK" value=0>

Table 3-3 General Attributes of the S7 Applets

Attribute Name	Type	Description
CODE	string	The attribute identifies the applet to be called. The following must always be specified: • Package path = constant • Applet type = variable Example: CODE="de.siemens.simaticnet.itcp.applets.S7IdentApplet.class"
CODEBASE	string	This attribute identifies the path under which the applet file or the applet archive (see ARCHIVE) is stored. Example: CODEBASE="/applets/"
ARCHIVE	string	This attribute names the applet archive in which the applet is contained. Example: ARCHIVE="s7applets.jar, s7api.jar"
Name	string	Unique Applet Name Using this name, displays in the Java Console, for example, can be identified as belonging to the applet. Example: NAME="s7_MLFB"
WIDTH	int	Width of the display in the HTML page This specifies the display width in the HTML page. The value selected must be high enough for the particular display. Example: WIDTH=150
HEIGHT	int	Height of the display in the HTML page. This specifies the display width in the HTML page. The value selected must be high enough for the particular display. Example: HEIGHT=35

Table 3-4 General Parameters of the S7 Applets

Parameter Name	Type	Description
BackGroundColor	string	Background color (24 bits RGB in hexadecimal format) This parameter controls the color intensity for the RGB components. For orientation, a selection of values is shown below: white: 0xFFFFFF black: 0x000000 red: 0xFF0000 green: 0x00FF00 blue: 0x0000FF Note: You can test the effects of the setting using the input tool (see Section 3.4.2).

3.4.2 Parameter Assignment Tools

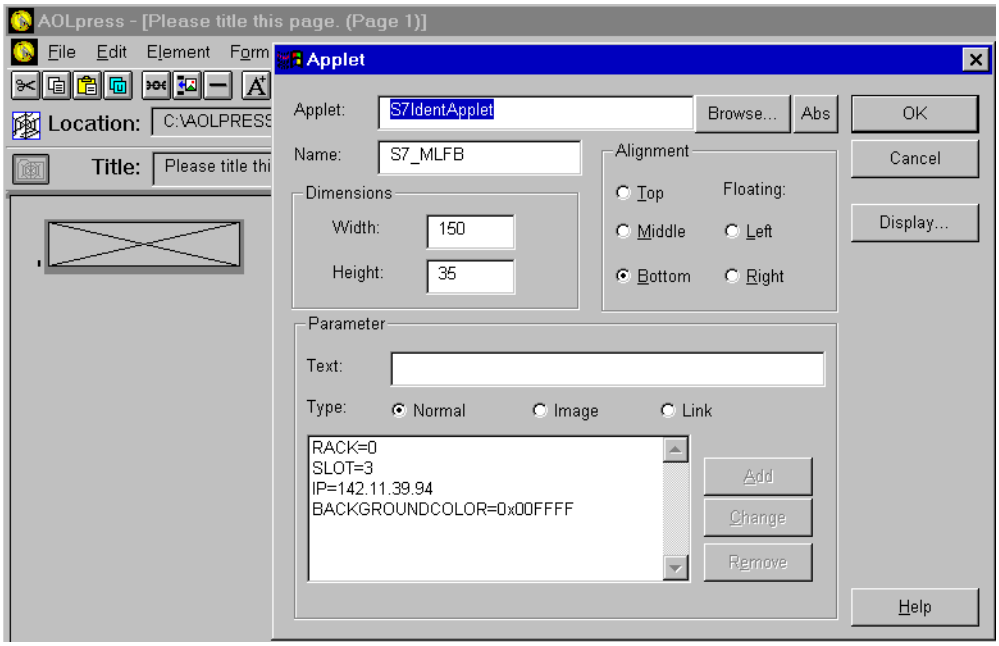
To support you when assigning parameters for the S7 applets, you can use various parameter assignment tools. These ensure that reliable, syntactically correct parameters are set for the S7 applets. The following tools are described:

- Input tool with the HTML editor
- Online parameter assignment for testing

The HTML pages available on the Manual Collection CD and on the IT-CP in the Examples folder also provide a simple, efficient way to reuse existing call parameters by simply copying and pasting them.

Input Tool of the HTML Editor

Some HTML editors, such as AOLpress provide input tools for syntactically correct calls and parameter assignment of Java applets. The following screen shot shows a dialog box in AOLpress for assigning parameters to applets.



Note: The disadvantage of this input tool is that you still need to **type in** the parameter names correctly.

Online Parameter Assignment for Testing

With the S7 applets, you can set parameters online. By double-clicking the output box on the HTML page, it is possible to modify the current parameters of the already active S7 applet in a dialog box.

Using the EDIT applet parameter you can activate or deactivate online parameter assignment. If the parameter is not used in the applet call, the default setting is online parameter assignment **deactivated**.

Example: By double-clicking the level of a tank in a plant picture, the dialog for online parameter assignment of the corresponding S7 applet is called.

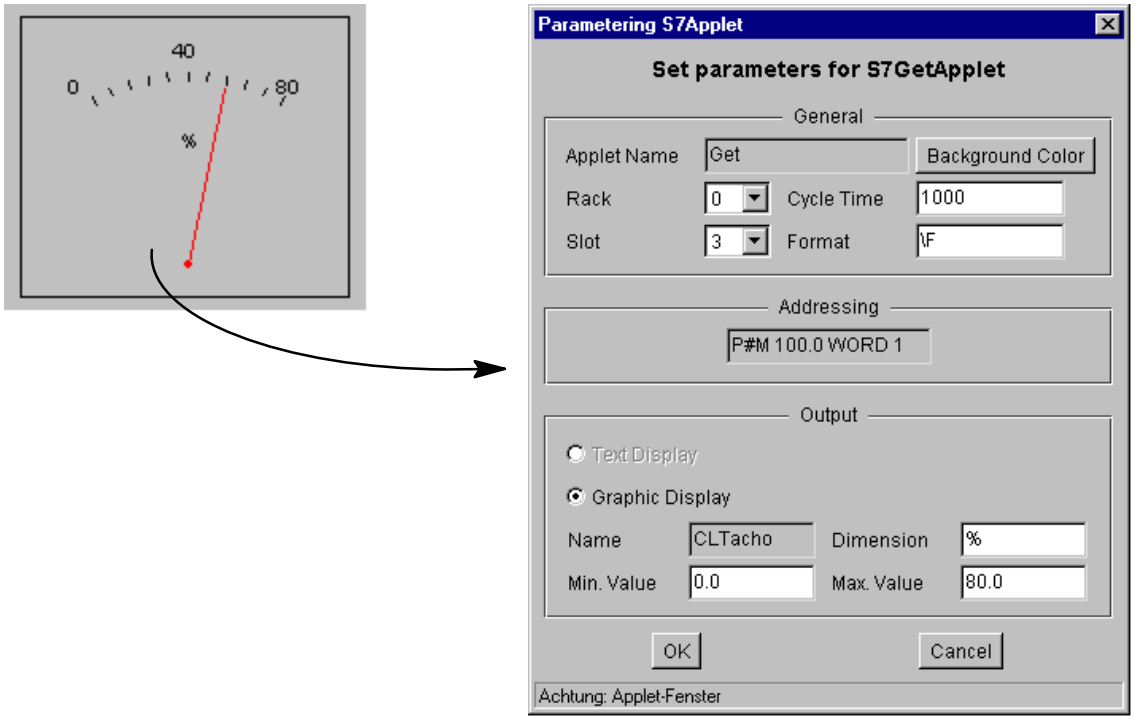


Figure 3-4

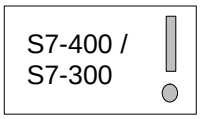
This function is intended primarily for testing. If you want to retain the modified parameters after the HTML page call, they must be entered in the HTML page using an HTML editor.

Access rights also apply during online parameter assignment.

Note

Settings remain valid only until you change to a new HTML page. (With some browsers, even changing the window size can lead to an applet being restart and reinitialization of the parameters.)

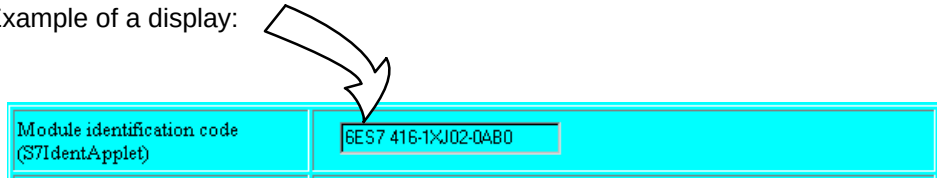
3.4.3 S7IdentApplet – Description



Meaning

This is used to identify an S7 module in an S7 station. The applet reads the order number and the version of the specified S7 module.

Example of a display:



Call Tags

CODE="de.siemens.simaticnet.itcp.applets.S7IdentApplet.class"
CODEBASE="/applets/"
ARCHIVE="s7applets.jar, s7api.jar"

Parameter Assignment

In addition to the general parameters (see Section 3.4.1), the following parameters for the specific function must have values:

Table 3-5 Parameters Specific to the Applet

Parameter Name	Type	Description
Slot	byte	Slot number of the addressed module (1 to 18)
RACK	byte	Rack number of the addressed module (0 to 7)

Table 3-6 Optional Parameters Specific to the Applet

EDIT	bool	Online parameter assignment can be activated or deactivated. Possible settings on = true off = false If the parameter in the applet call is not used, the default setting for online parameter assignment is **off** (deactivated)!
LANGUAGE	string	The language use for labels, messages, and diagnostic displays can be set permanently. Currently only German and English are supported. The two-character ISO-639 codes are used as parameters. Possible settings German = “de” English = “en” If the parameter is not used in the applet call, the language of the host system is used (or English if this language is not available). The language of the host system can be changed in Windows operating systems in the Control Panel – Regional Settings). If an unsupported language is specified, the language of the host system is used (or English if this language is not available).
DEBUGLEVEL	int	Sets the level of detail for debug display in the Java console. Possible settings 0 = no display 1 = display all messages 2 = display warning and error messages only 3 = display error messages only 4 = display only messages relating to fatal errors if the parameter is not used in the applet call, level 3 is used; in other words, only error messages are displayed.

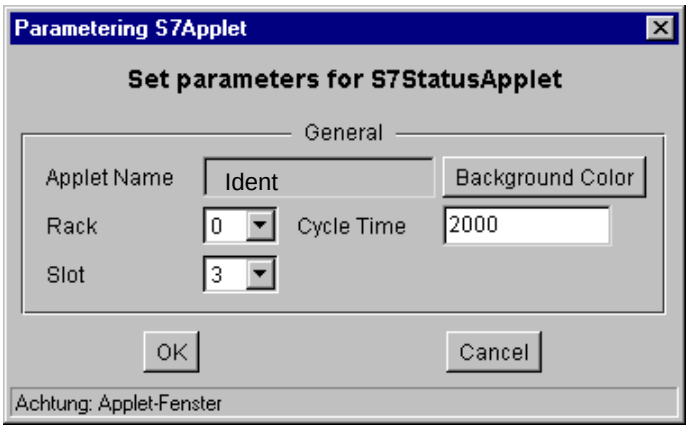
Access Rights

With the user name used for access, the following access right must be available (refer to the ”Edit User Entry” dialog in Section LEERER MERKER):

- “Query the order number of modules”

Parameter Assignment Tools (meaning and application see Section 3.4.2)

Online parameter assignment is supported for test purposes. To use this option, double-click the output field to open the parameter assignment dialog.

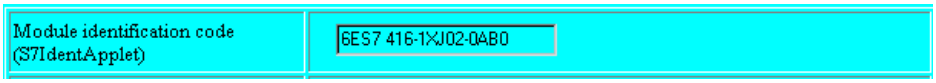


3.4.4 S7IdentApplet – Example

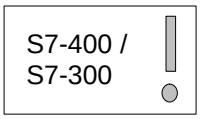
With the sample shown here, you obtain the identification code of an S7 module in rack 0, slot 3 and display it in numeric format.

```
<APPLET CODE="de.siemens.simaticnet.itcp.applets.S7IdentApplet.class"
CODEBASE="/applets/"
 ARCHIVE="s7applets.jar, s7api.jar" NAME="s7_MLFB" WIDTH=150
HEIGHT=35>
<PARAM name="RACK" value=0> <PARAM name="SLOT" value=3>
<PARAM name="BACKGROUNDCOLOR" value="0x00FFFF">
<PARAM name="EDIT" value="true">
 </APPLET>
```

Result:



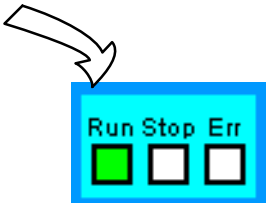
3.4.5 S7StatusApplet – Description



Meaning

This queries the status information of the specified module.

Example of a display:



Data Output / Display

The data output/display is graphic with additional text. The following table provides more information.

Table 3-7

Display / Color	Additional Text	Meaning
green	Run	The user program is running.
Yellow	Stop	The user program has been stopped.
Gray	Unknown	The connection to the CP is established; waiting for response.
red:	Error	There is a module error message.
Blue	Error	There is no connection to the addressed connection.

Access Rights

With the user name used for access, the following access right must be available (refer to the ”Edit User Entry” dialog in Section LEERER MERKER):

- “query the status of modules”;

Call Tags

CODE=“de.siemens.simaticnet.itcp.applets.S7StatusApplet.class”
CODEBASE=“/applets/”
ARCHIVE=“s7applets.jar, s7api.jar”

Parameter Assignment

In addition to the general parameters (see Section 3.4.1), the following parameters for the specific function must have values:

Table 3-8 Parameters Specific to the Applet

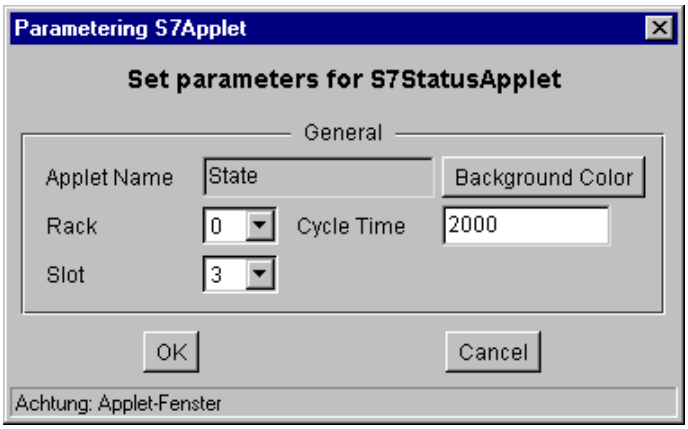
Parameter Name	Type	Description
Slot	byte	Slot number of the addressed module; (1–18)
RACK	byte	Rack number of the addressed module; (0–7)
CYCLETIME (cycle time)	int	Cycle time for the read job, specified in milliseconds. >5000 (recommended value)

Table 3-9 Optional Parameters Specific to the Applet

EDIT	bool	Online parameter assignment can be activated or deactivated. Possible settings on = true off = false If the parameter in the applet call is not used, the default setting for online parameter assignment is **off** (deactivated)!
LANGUAGE	string	The language use for labels, messages, and diagnostic displays can be set permanently. Currently only German and English are supported. The two-character ISO-639 codes are used as parameters. Possible settings German = “de” English = “en” If the parameter is not used in the applet call, the language of the host system is used (or English if this language is not available). The language of the host system can be changed in Windows operating systems in the Control Panel – Regional Settings). If an unsupported language is specified, the language of the host system is used (or English if this language is not available).
DEBUGLEVEL	int	Sets the level of detail for debug display in the Java console. Possible settings 0 = no display 1 = display all messages 2 = display warning and error messages only 3 = display error messages only 4 = display only messages relating to fatal errors if the parameter is not used in the applet call, level 3 is used; in other words, only error messages are displayed.

Parameter Assignment Tools (meaning and application see Section 3.4.2)

Online parameter assignment is supported for test purposes. To use this option, double-click the output field to open the parameter assignment dialog.



3.4.6 S7StatusApplet – Example

The example shown here displays the status of an S7 module in rack 0, slot 3 graphically.

```
<APPLET CODE="de.siemens.simaticnet.itcp.applets.S7StatusApplet.class"
ARCHIVE="s7applets.jar, s7api.jar" NAME="s7_status_3" WIDTH=80
HEIGHT=20>
<PARAM name="RACK" value=0>
 <PARAM name="SLOT" value=3>
<PARAM name="CYCLETIME" value=5000>
<PARAM name="BACKGROUNDCOLOR" value="0xFFFFFF">
<PARAM name="EDIT" value="true">
 </APPLET>
```

Result:

3.4.7 S7GetApplet – Description

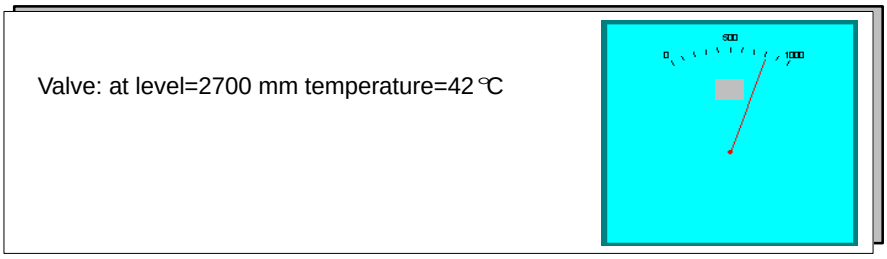
Meaning

The applet reads data or data areas cyclically according to the parameter assignment from the S7 CPU. You can name the variables symbolically (not with the S7-200) or by specifying addresses.

Using a format string that is explained later, you specify the output form of the data.

The process values can be displayed either numerically or graphically.

Example of a display (numeric and graphic):



How to use graphic display and the JavaBeans is explained in Section 4.

Requirements

The variables can be identified using symbolic names or by their addresses.

Symbolic access (not with S7-200) requires suitably configured symbols on the IT-CP. If symbolic access is used, access rights are checked according to the variable configuration (see Section 3.5).

Access Rights

S7-400 / S7-300

With the user name used for access, the following access right must be available (refer to the ”Edit User Entry” dialog in Section LEERER MERKER):

- ”Use the symbol table” (with symbolic access)
- ”Read variables using absolute addresses” (with absolute access)

Call Tags

CODE=“de.siemens.simaticnet.itcp.applets.S7GetApplet.class”
CODEBASE=“/applets/”
ARCHIVE=“s7applets.jar, s7api.jar”

Parameter Assignment

In addition to the general parameters (see Section 3.4.1), the following parameters for the specific function must have values:

Table 3-10 Parameters Specific to the Applet

Parameter Name	Type	Description
Slot	byte	Slot number of the addressed module (1 to 18) (with S7-200 always R/S=0/0)
RACK	byte	Rack number of the addressed module (0 to 7) (with S7-200 always R/S=0/0)
CYCLETIME	int	Cycle time for the read job, specified in milliseconds. Range of values: >5000 (recommended)
FORMAT	string	The character string in the format parameter specifies how the read variable values will be displayed (example and explanation see below).

Table 3-11 Optional Applet-Specific Parameters – using JavaBeans

Parameter Name	Type	Description
DISPLAY	string	Identifies a JavaBean that can be used for graphic data display. The currently possible values are as follows (see also Table 4-2, Page 77): • CLTacho • CLLevel • CLThermo
MINVAL	int, float	MINVAL and MAXVAL specify the range for formatting graphically displayed variables.
MAXVAL	int, float	
DIMENSION	string	Here, you can specify a physical unit.
EDIT	bool	Online parameter assignment can be activated or deactivated. Possible settings on = true off = false If the parameter in the applet call is not used, the default setting for online parameter assignment is **off** (deactivated)!

Table 3-11 Optional Applet-Specific Parameters, (Suite) – using JavaBeans, continued

Parameter Name	Type	Description
LANGUAGE	string	The language use for labels, messages, and diagnostic displays can be set permanently. Currently only German and English are supported. The two-character ISO-639 codes are used as parameters. Possible settings German = “de” English = “en” If the parameter is not used in the applet call, the language of the host system is used (or English if this language is not available). The language of the host system can be changed in Windows operating systems in the Control Panel – Regional Settings). If an unsupported language is specified, the language of the host system is used (or English if this language is not available).
DEBUGLEVEL	int	Sets the level of detail for debug display in the Java console. Possible settings 0 = no display 1 = display all messages 2 = display warning and error messages only 3 = display error messages only 4 = display only messages relating to fatal errors if the parameter is not used in the applet call, level 3 is used; in other words, only error messages are displayed.

Table 3-12 Parameters for Symbolic Variable Addressing (**alternative** to absolute addressing – **only with S7-300/400**)

Parameter Name	Type	Description
SYMBOL	string	Name of the S7 variable symbol The variable must be created with the STEP 7 symbol editor and configured for access via the IT-CP (see Section 3.5).

Table 3-13 Parameters for Indirect Variable Addressing using the ANY Pointer (**as an alternative** to SYMBOL)

Parameter Name	Type	Description
VARTYPE (data type)	int	This variable type coding in the ANY pointer indicates the data type of the variable to be read; possible entries are as follows: 0x02 BYTE bytes (8 bits) 0x01 BOOL Data type BOOL (1 bit) 0x03 CHAR characters (8 bits) 0x04 WORD words (16 bits) 0x05 INT integers (16 bits) 0x06 DWORD words (32 bits) 0x07 DINT integers (32 bits) 0x08 REAL real numbers (32 bits) 0x09 DATE date 0x0A TIME_OF_DAY (TOD) time of day 0x0B TIME Time 0x13 STRING Character string This can be specified in decimal (for example 10) or hexadecimal (for example 0x0A). Note: The following data types are available and can be selected in the parameter dialog. Transferring these complex data types is, however, supported only by the S7 beans (see Chapter 4). Based on the S5 or S7 format description (see STEP 7 online help), these formats can then be decoded and processed further by the program. 0x0C S5TIME S5TIME data type 0x0E DATE_AND_TIME (DT) Date and time (64 bits) 0x1C COUNTER counter 0x1D TIMER timer **Note:** The information here applies to the S7-300/400; for theS7-200 see /4/
VARCNT (repetition factor)	int	Number of variables to be read; With this information, you can specify whether a variable or contiguous variable area is transferred. STEP 7 identifies arrays and structures as a number (here using the repetition factor) of data types. Example: If 10 words are to be transferred, the value 10 must be specified for the repetition factor and the value 04 for the data type.
VARAREA (memory area)	int	Area coding for identifying the memory area; 0x81 I memory area of inputs 0x82 Q memory area of outputs 0x83 M bit memory area 0x84 DB data block. This can be specified in decimal (for example 131) or hexadecimal (for example 0x83). **Note:** The information here applies to the S7-300/400; for the S7-200 see /4/
VARSUBAREA (subarea)	int	Subarea coding; for example, by specifying the DB number. (with S7-200 always value=1)

Table 3-13 Parameters for Indirect Variable Addressing using the ANY Pointer, (Suite)(**as an alternative** to SYMBOL), Fortsetzung

Parameter Name	Type	Description
VAROFFSET (byte address)	int	Specifies a byte offset. This information can be used to address the variable within the specified memory area (VARAREA), for example, the memory bit number.
VARBITOFFSET (bit address)	int	Specifies a bit offset. This information can be used to address the bit of a variable of the type BOOL.

Notes on the Address Information VARTYPE...VAROFFSET

To read the data, the IT-CP uses the S7 function SFB14 (GET). To address variables, the data type ANY pointer must therefore be supplied to transfer parameters to the SFB.

For further information on SFB14 (GET), refer to the online help in STEP 7 under the topic ”Format of the ANY Parameter Type”; you will also find a detailed description of the ANY pointer in LEERER MERKER.

Range of Values and Using the FORMAT Parameter

The following identifiers can be used in the FORMAT parameter:

Table 3-14 Meaning of the Format Parameter

Identifier	Number of Relevant Bytes	representation
\	0	\; Indicates that the following character is an ID as listed in this table. To display ”\” the following entry is necessary: ”\\” Example of a variable of the type integer: \I
S	1	Bit string Interprets the assigned byte as a string of bits to be represented individually. Example of output: 01101110
O	1	Octal
H	1	Hexadecimal
B	1	Unsigned byte
C	1	Signed byte
D	4	Unsigned 32
L	4	Signed 32

Table 3-14 Meaning of the Format Parameter, Fortsetzung

Identifier	Number of Relevant Bytes	representation
W	2	Unsigned 16
I	2	Signed 16
F	4	Floating Point
Z	1	Character
X (n, string1, string2)	1	Binary value (n= position 0 to 7, string1 = character string for value 1; string 2 = character string for value 0). If you want to use several binary values within a byte for output, the identifier Y must be used alternately! **Note the following!** Between the opening bracket and the first comma, there must be **no** blank, otherwise the entry is not recognized.
Y (n, on, off)	0	Binary value (n= position 0 to 7, string1 = character string for value 1; string 2 = character string for value 0). Notes on the function: In contrast to ID X, the position counter is not incremented. Y is used when several binary values within a byte are to be output. The ID X is used only for the output of the last binary value within a byte. **Note the following!** Between the opening bracket and the first comma, there must be **no** blank, otherwise the entry is not recognized.
G	1	Increments the position counter by 1 byte without representation: This ID is required to skip bytes in the variable string. This is necessary when empty bytes must be taken into account due to the data structure (for example words and bytes are defined alternately).

Interpretation of the Format String

The character string in the format parameter specifies how the read variable values will be displayed (example see below). It is assumed that variable values are read in the form of byte strings.

In the representation, the character string in the format string is interpreted starting from the left and assigned to the variable string. With each value assigned and output, a position counter is incremented according to the specified ”number of relevant bytes” in Table 3-14.

Output continues until all the format IDs have been processed. If it is not possible to assign all bytes, there is no further output. If more format IDs are specified than can be assigned, output continues until the format IDs have all been processed.

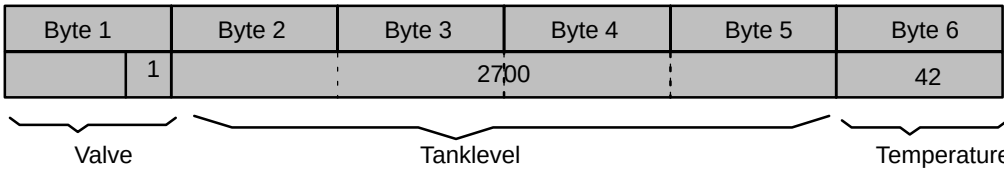
Note

Make sure that the format string exactly matches the bytes in the variable string.

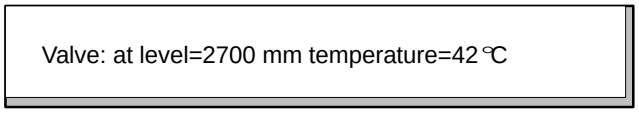
The following schematic illustrates the assignment and output based on an example of 3 variables.

The variables are read as a contiguous string of 6 bytes from the S7-CPU. Following this, the format assignment assigns the various variable types and the variables are displayed.

Format string:valve: \X(0, open, closed) tanklevel= \D mm temperature= \B
℃
Variable string read (6 bytes):



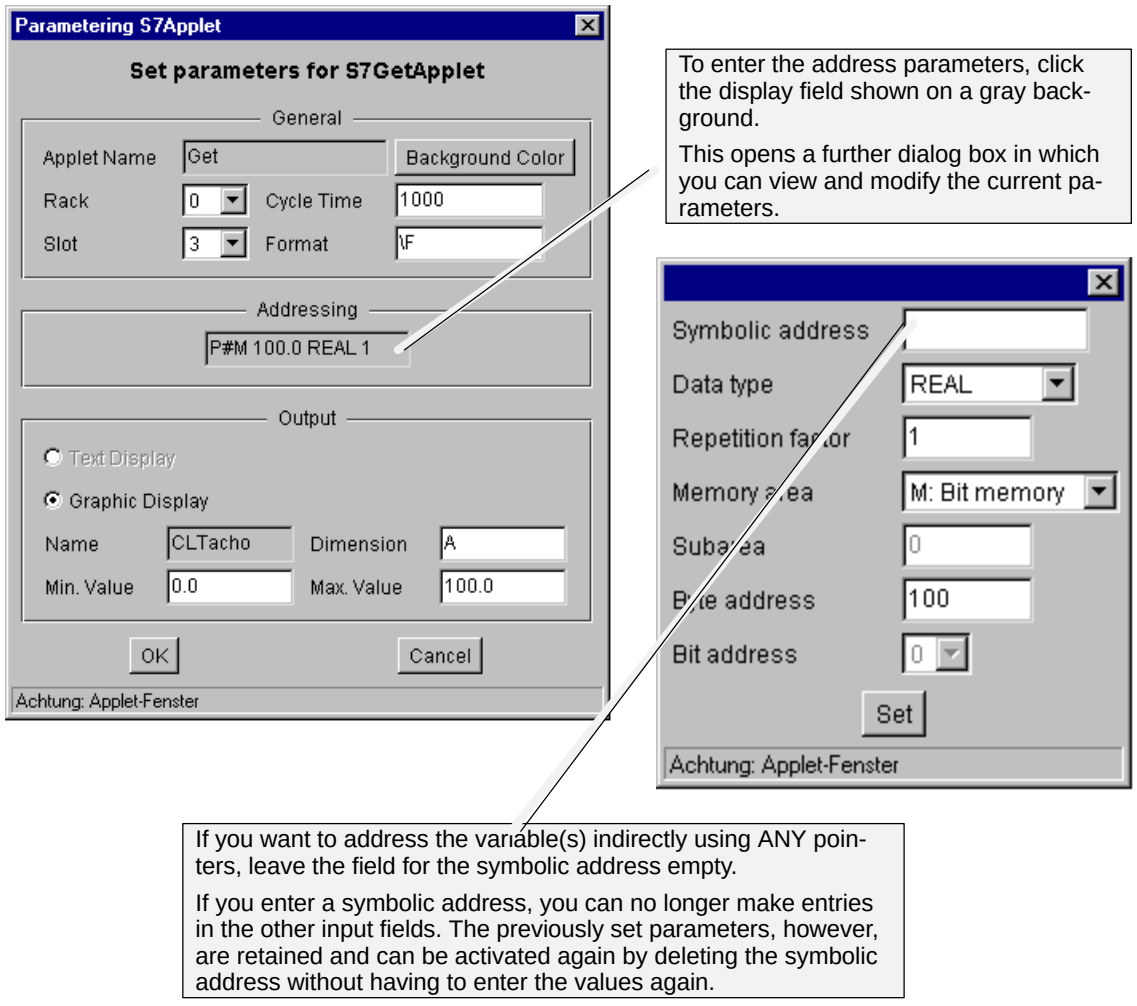
Display on the HTML Page



An **error message** (refer to output in the Java Console in Section 3.6) is only generated when there is an assignment conflict. Example: \D – a double word – is defined in the format string, the variable read, however, is only 2 bytes long.

Parameter Assignment Tools (meaning and application see Section 3.4.2)

Online parameter assignment is supported for test purposes. To use this option, double-click the output field to open the parameter assignment dialog.



To enter the address parameters, click the display field shown on a gray background.

This opens a further dialog box in which you can view and modify the current parameters.

If you want to address the variable(s) indirectly using ANY pointers, leave the field for the symbolic address empty.

If you enter a symbolic address, you can no longer make entries in the other input fields. The previously set parameters, however, are retained and can be activated again by deleting the symbolic address without having to enter the values again.

3.4.8 S7GetApplet – Examples

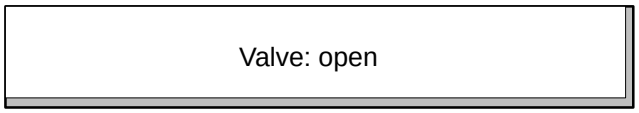
The two possible access methods using applet parameter settings are shown below based on simple examples.

Example 1: Accessing a variable in a data block

The example assumes a binary variable that contains the state of a valve – open/closed –. This variable is stored with the name “valve” in DB10. DB10 contains the name “heater1” in the symbol table of the CPU.

The variable is displayed as a character string.

Display in the HTML page



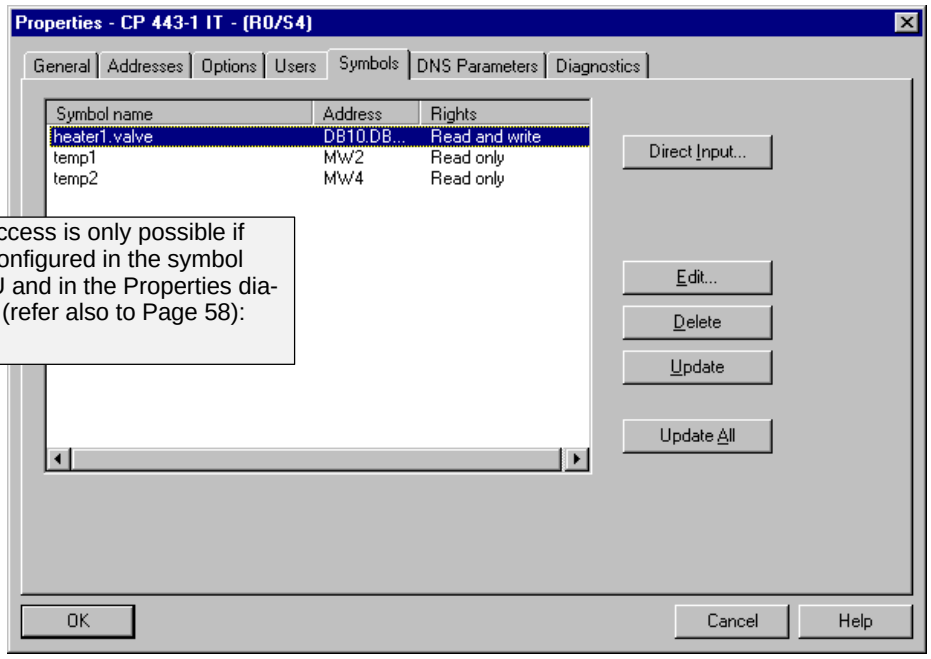
To access this variable and to display it on the HTML page, the following applet parameter settings are required.

S7-400 / S7-300

a) Access with symbolic addressing of the variable

```
<P ALIGN=Center><APPLET CODE=”de.siemens.simaticnet.itcp.
   applets.S7GetApplet.class”

CODEBASE=“/applets/” ARCHIVE=“s7applets.jar, s7api.jar”  NAME=“valve”
WIDTH=45 HEIGHT=30>

 <PARAM name=”RACK” value=0>  <PARAM name=”SLOT” value=3>
 <PARAM name=”CYCLETIME” value=5000>
<PARAM name=”SYMBOL” value=”heater1.valve”>
<PARAM name=”FORMAT”  value=”Ventil: \X(0,auf,zu)”>
<PARAM name=”BACKGROUNDCOLOR” value=”0xFFFFFF”>
<PARAM name=”EDIT” value=”true”>
</APPLET>
```

With the “Format” parameter, remember that a variable of the type byte is being accessed that contains the binary value at position ”0”.



This symbolic access is only possible if the variable is configured in the symbol table of the CPU and in the Properties dialog of the IT-CP (refer also to Page 58):

b) Access with indirect addressing of the variable

```
<P ALIGN=Center><APPLET CODE="de.siemens.simaticnet.itcp.
   applets.S7GetApplet.class"

CODEBASE="/applets/" ARCHIVE="s7applets.jar, s7api.jar"  NAME="valve"
WIDTH=45 HEIGHT=30>

 <PARAM name="RACK" value=0> <PARAM name="SLOT" value=3>
 <PARAM name="CYCLETIME" value=5000>
<PARAM name="VARTYPE" value=2>
<PARAM name="VARCNT" value =1>
<PARAM name="VARAREA" value=0x84>
<PARAM name="VARSUBAREA" value=10>
<PARAM name="VAROFFSET" value=40>
<PARAM name="FORMAT" value="Ventil: \X(0,auf,zu)">
<PARAM name="BACKGROUNDCOLOR" value="0xFFFFFF">
<PARAM name="EDIT" value="true">
</APPLET>
```

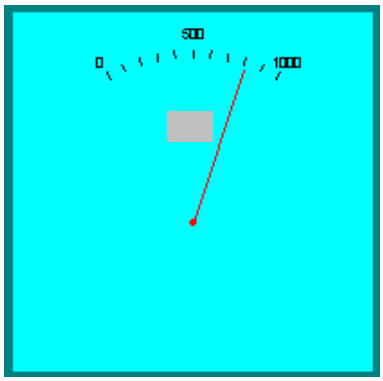
This info corresponds to ANY pointer:

P#DB10.DBX 40.0 BYTE 1

Example 2: Accessing the variable in a bit memory area

In the example, a 16-bit memory word (MW 12) will be read. Access is achieved by indirect addressing of the variable.

The variable is displayed in graphic form.



To access this variable and to display it on the HTML page, the following applet parameter settings are required.

```
<P ALIGN=Center><APPLET CODE="de.siemens.simaticnet.itcp.applets.
   S7GetApplet.class"
CODEBASE="/applets/" ARCHIVE="s7applets.jar, s7api.jar"  NAME="speed"
WIDTH=45 HEIGHT=30>
 <PARAM name="RACK" value=0> <PARAM name="SLOT" value=3>
 <PARAM name="CYCLETIME" value=5000>
<PARAM name="VARTYPE" value=5>
<PARAM name="VARCNT" value =1>
<PARAM name="VARAREA" value=0x83>
<PARAM name="VARSUBAREA" value=0>
<PARAM name="VAROFFSET" value=12>

<PARAM name="DISPLAY"  value="CLTACHO">
<PARAM name="MINVAL" value=0><PARAM name="MAXVAL" value=1000>
<PARAM name="BACKGROUNDCOLOR" value="0x00FFFF">
<PARAM name="EDIT" value="true">
</APPLET>
```

This info corresponds to ANY pointer:

P#MW12 INT 1

For the graphic display, the S7 JavaBean CLTACHO is used.

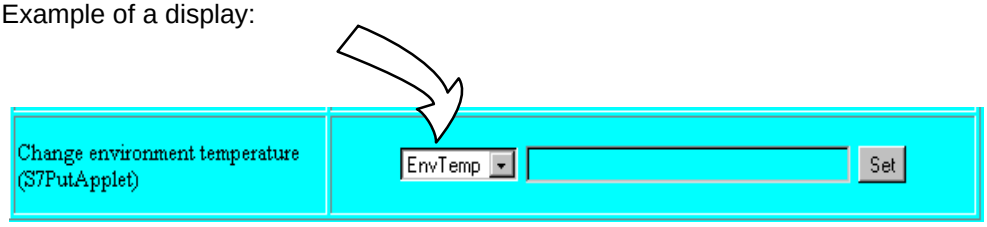
3.4.9 S7PutApplet – Description

Meaning

This applet receives variable values entered by the user and transfers them to the S7-CPU.

According to the parameter settings in a format string, two fields are displayed in the Web browser in which you select the data area and enter the variable value. An additional button “Set” is used to trigger the write job.

Example of a display:



Requirements

The variables can be identified using symbolic names or by their addresses.

S7-400 / S7-300

Symbolic access is only possible if symbols were configured on the IT-CP. If symbolic access is used, access rights are checked according to the variable configuration (see Section 3.5).

Access Rights

With the user name used for access, the following access right must be available (refer to the ”Edit User Entry” dialog in Section LEERER MERKER):

- ”Use the symbol table” (only with symbolic access)
- ”Write variables using absolute addresses” (only with absolute access)

Call Tags

CODE=“de.siemens.simaticnet.itcp.applets.S7PutApplet.class”
CODEBASE=“/applets/”
ARCHIVE=“s7applets.jar, s7api.jar”

Parameter Assignment

In addition to the general parameters (see Section 3.4.1), the following parameters for the specific function must have values:

Table 3-15 Parameters Specific to the Applet

Parameter Name	Type	Description

Table 3-15 Parameters Specific to the Applet, continued

Parameter Name	Type	Description
Slot	byte	Slot number of the addressed module (1 to 18) (with S7-200 always R/S=0/0)
RACK	byte	Rack number of the addressed module (0 to 7) (with S7-200 always R/S=0/0)

Table 3-16 Optional Parameters Specific to the Applet

EDIT	bool	Online parameter assignment can be activated or deactivated. Possible settings on = true off = false If the parameter in the applet call is not used, the default setting for online parameter assignment is **off** (deactivated)!
LANGUAGE	string	The language use for labels, messages, and diagnostic displays can be set permanently. Currently only German and English are supported. The two-character ISO-639 codes are used as parameters. Possible settings German = “de” English = “en” If the parameter is not used in the applet call, the language of the host system is used (or English if this language is not available). The language of the host system can be changed in Windows operating systems in the Control Panel – Regional Settings). If an unsupported language is specified, the language of the host system is used (or English if this language is not available).
DEBUGLEVEL	int	Sets the level of detail for debug display in the Java console. Possible settings 0 = no display 1 = display all messages 2 = display warning and error messages only 3 = display error messages only 4 = display only messages relating to fatal errors if the parameter is not used in the applet call, level 3 is used; in other words, only error messages are displayed.

Table 3-17 Parameters for Symbolic Data Addressing **(S7-300/400 only)**

Parameter Name	Type	Description
SYMBOLNUM	int	Number of variables that can be entered as symbols
SYMBOLn *)	string	Name of the S7 variable symbol. The name appears in the list box of the first text field. The variable must be created with the STEP 7 symbol editor and be configured for access via the IT-CP (see Section 3.5).
SYMFORMATn *)	string	The character string in the format parameter specifies how the entered variable values will be interpreted. The assignment of the entry to S7 variables is made according to the value n (example see below).

*) Key: “n” indicates consecutive numbering of the variables addressed as symbols within a call;

Table 3-18 Parameters for Addressing Data Using Absolute Addresses

Parameter Name	Type	Description
VARNUM	int	Number of variables to be written;
VARNAMEn *)	string	Variable name of a variable addressed indirectly. The name appears in the display in the list box of the first text field. Maximum 256 characters
VARTYPEn *) (data type)	int	Variable type coding in the ANY pointer; The variable type coding in the ANY pointer indicates the data type of the variables to be written; the following entries are possible: 0x02 BYTE bytes (8 bits) 0x01 BOOL Data type BOOL (1 bit) 0x03 CHAR characters (8 bits) 0x04 WORD words (16 bits) 0x05 INT integers (16 bits) 0x06 DWORD words (32 bits) 0x07 DINT integers (32 bits) 0x08 REAL real numbers (32 bits) 0x09 DATE date 0x0A TIME_OF_DAY (TOD) time of day 0x0B TIME Time 0x13 STRING Character string Note: The following data types are available and can be selected in the parameter dialog. Transferring these complex data types is, however, supported only by the S7 beans (see Chapter 4). Based on the S5 or S7 format description (see STEP 7 online help), these formats can then be decoded and processed further by the program. 0x0C S5TIME S5TIME data type 0x0E DATE_AND_TIME (DT) Date and time (64 bits) 0x1C COUNTER counter 0x1D TIMER timer **Note:** The information here applies to the S7-300/400; for the S7-200 see /4/

Table 3-18 Parameters for Addressing Data Using Absolute Addresses, continued

Parameter Name	Type	Description
VARAREAn *) (memory area)	int	Area coding for identifying the memory area; 0x81 I memory area of inputs 0x82 Q memory area of outputs 0x83 M bit memory area 0x84 DB data block. This can be specified in decimal (for example 131) or hexadecimal (for example 0x83). **Note:** The information here applies to the S7-300/400; for the S7-200 see /4/
VARSUBAREAn *) (subarea)	int	Subarea coding; for example, by specifying the DB number.
VAROFFSETn *) (byte address)	int	Specifies a byte offset. This information can be used to address the variable within the specified memory area (VARAREA), for example, the memory bit number.
VARBITOFFSET (bit address)	int	Specifies a bit offset. This information can be used to address the bit of a variable of the type BOOL.
VARFORMATn *)	string	The character string in the format parameter specifies how the entered variable values will be interpreted. The information is assigned to the S7 variable in the input field according to the value specified for n (example see Section 3.4.10).

*) Legend: “n” stands for a continuous numbering of the indirectly (via ANY pointer) addressed variables within a call beginning with “1”.

Range of Values for the FORMAT Parameter

The following identifiers can be used in the FORMAT parameter:

Table 3-19 Meaning of the Format Parameter

Identifier	Number of Bytes Used	The input string is interpreted as follows
S	1	Bit string
O	1	Octal
H	1	Hexadecimal
B	1	Unsigned byte
C	1	Signed byte
D	4	Unsigned 32
L	4	Signed 32
W	2	Unsigned 16

Table 3-19 Meaning of the Format Parameter, continued

Identifier	Number of Bytes Used	The input string is interpreted as follows
I	2	Signed 16
Z	n	Character string
F	4	Floating point 32

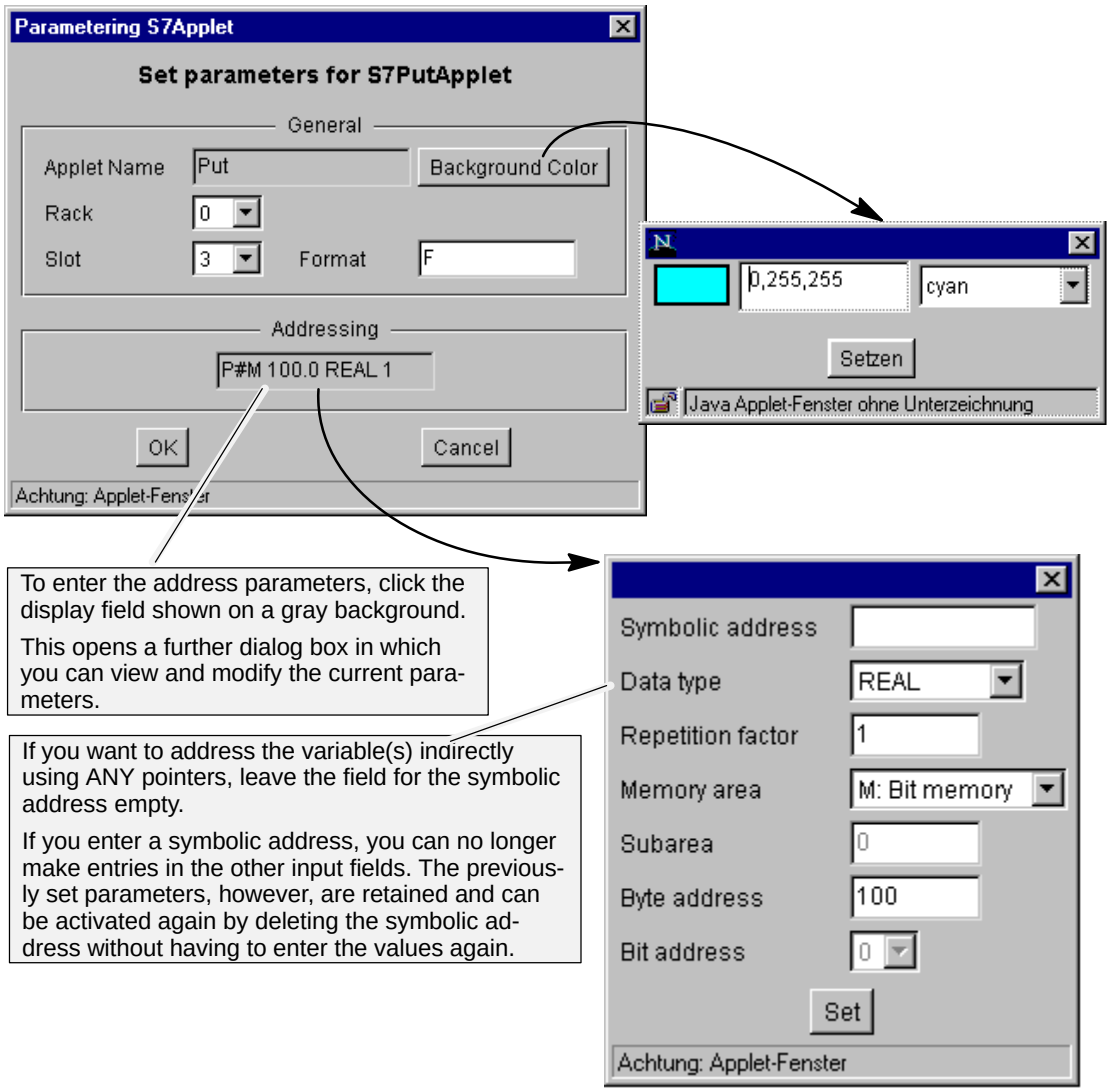
Note

In contrast to the S7GetApplet, the ”\” ID is not required with the S7PutApplet.

Parameter Assignment Tools (meaning and application see Section 3.4.2)

- **Online parameter assignment** is supported for test purposes. The EDIT applet parameter must be set to true.

  Double-click on the display area (**caution! not** in the input field for the variable value!) of the Put applet to open the following dialog box:



To enter the address parameters, click the display field shown on a gray background.

This opens a further dialog box in which you can view and modify the current parameters.

If you want to address the variable(s) indirectly using ANY pointers, leave the field for the symbolic address empty.

If you enter a symbolic address, you can no longer make entries in the other input fields. The previously set parameters, however, are retained and can be activated again by deleting the symbolic address without having to enter the values again.

3.4.10 S7PutApplet – Examples

Example 1: Entering a variable

To enter a variable that will be used, for example for a setpoint, the following applet parameter assignment is necessary:

S7-400 / S7-300

a) when addressing the variable as a symbol

```
<APPLET CODE=”de.siemens.simaticnet.itcp.applets.S7PutApplet.class”
CODEBASE=”/applets/”
ARCHIVE=”s7applets.jar, s7api.jar” NAME=”s7_client0” WIDTH=400
HEIGHT=40>
 <PARAM name=”RACK” value=0>
 <PARAM name=”SLOT” value=3>
<PARAM name=”SYMBOLNUM” value=”1”>
<PARAM name=”SYMBOL1” value=”Desired_value_tank1”>
<PARAM name=”SYMFORMAT1” value=”I”>
<PARAM name=”BACKGROUNDCOLOR” value=”0x00FFFF”>
<PARAM name=”EDIT” value=”true”>
</APPLET>
```

b) S7 applet with indirect access:

```
<APPLET CODE=“”de.siemens.simaticnet.itcp.applets.S7PutApplet.class”
CODEBASE=“/applets/”
ARCHIVE=“s7applets.jar, s7api.jar” NAME=“s7_client0” WIDTH=400
HEIGHT=40>
 <PARAM name=”RACK” value=0>
 <PARAM name=”SLOT” value=3>
<PARAM name=”VARNUM” value=”1”>
<PARAM name=”VARNAME1” value=“Desired value tank1”>
<PARAM name=”VARTYPE1” value=2>
<PARAM name=”VARAREA1” value=0x84>
<PARAM name=”VARSUBAREA1” value=0x10>
<PARAM name=”VAROFFSET1” value=40>
<PARAM name=”VARFORMAT1” value=”B”>
<PARAM name=”BACKGROUNDCOLOR” value=”0x00FFFF”>
<PARAM name=”EDIT” value=”true”>
</APPLET>
```

This info corresponds to ANY pointer:

P#DB10.DBX 40.0 BYTE 1

Example 2: Entering several variables

With an S7PutApplet, you can enter more than one variable. It is also possible to mix the type of addressing (symbolic or indirect).

In the following example, three variables are addressed using symbolic addressing and one variable using indirect addressing.

If you enter several variables with mixed address types, the parameter settings for an applet might appear as follows:

```
<APPLET CODE="de.siemens.simaticnet.itcp.applets.S7PutApplet.class"
CODEBASE="/applets/"

ARCHIVE="s7applets.jar, s7api.jar" NAME="s7_client0" WIDTH=400
HEIGHT=40>

 <PARAM name="RACK" value=0>
 <PARAM name="SLOT" value=3>

Three variables using symbolic addresses:
<PARAM name="SYMBOLNUM" value="3">
<PARAM name="SYMBOL1" value="Sollwert_Kessel1">
<PARAM name="SYMFORMAT1" value="I">
<PARAM name="SYMBOL2" value="Sollwert_Kessel2">
<PARAM name="SYMFORMAT2" value="I">
<PARAM name="SYMBOL3" value="Grenzwert_hoch">
<PARAM name="SYMFORMAT3" value="I">

One variable using indirect addressing:
<PARAM name="VARNUM" value="1">
<PARAM name="VARNAME1" value="Desired_value_tank3">
<PARAM name="VARTYPE1" value=2>
<PARAM name="VARAREA1" value=0x83>
<PARAM name="VARSUBAREA1" value=0x00>
<PARAM name="VAROFFSET1" value=40>
<PARAM name="VARFORMAT1" value="B">

<PARAM name="BACKGROUNDCOLOR" value="0x00FFFF">
<PARAM name="EDIT" value="true">
</APPLET>
```

3.5 Configuring Variables for Access Using Symbols

S7–400 / S7–300

The Use of Symbols Avoids Configuration Errors

The S7 applets S7GetApplet and S7PutApplet allow convenient access to variables using symbolic names familiar from LAD/FBD/STL programming with the symbol table. This saves you time and headache trying to work with absolute addresses.

The following example shows how a name is assigned to data block DB100 in the symbol table.

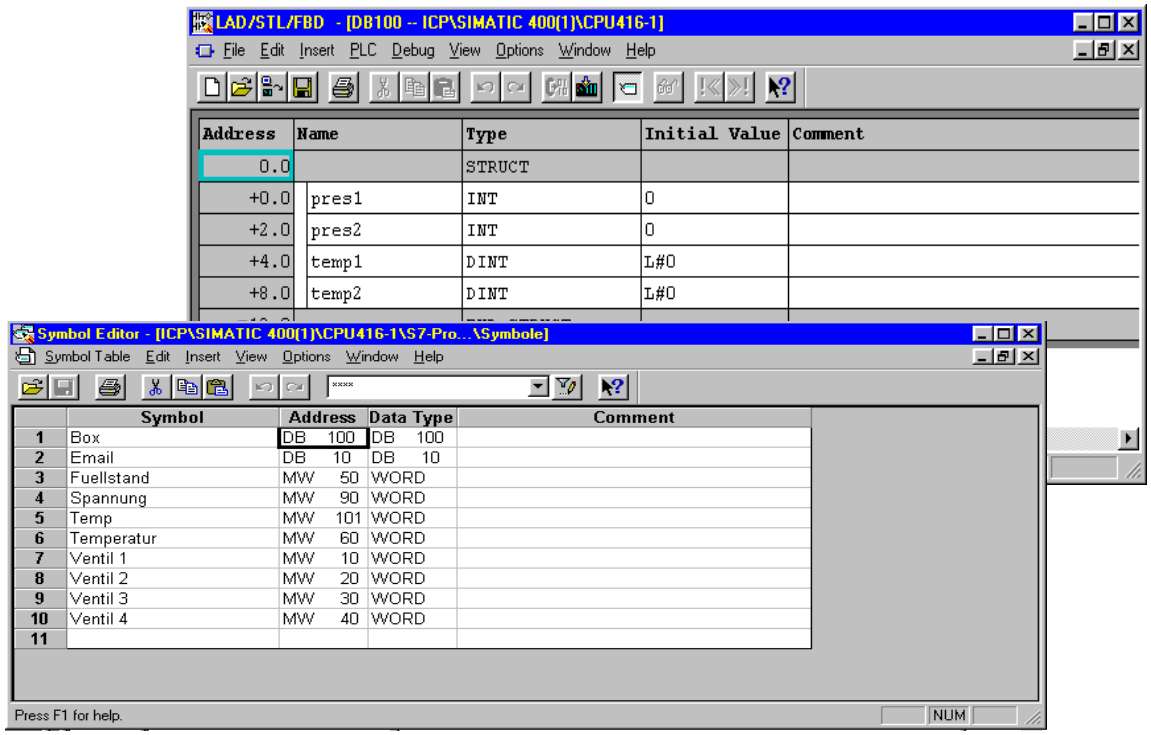


Figure 3-5

Symbols for HTML Pages

To access variables on the S7 CPU with Java applets using a Web browser, you must inform the IT-CP of the names, addresses, and access rights of the variables. There are tabs for these settings in the Properties dialog of the IT-CP.

The symbols that you specify here in the configuration must first be declared in the symbol table with the STEP 7 symbols editor. Following this, the assignments of the symbols to the variables have been established. With the configuration described here, you select the symbols you want to be able to access with the Web browser.

Procedure

Open the Properties dialog of your IT-CP in STEP 7 HW Config.

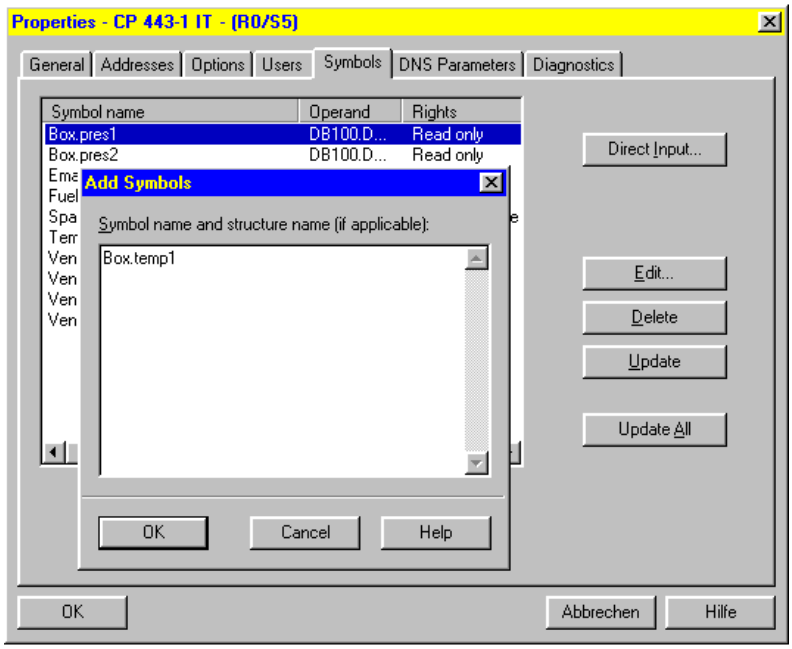


Figure 3-6

Enter the symbols of the variables or structure elements you want to be able to access with the Web browser. You can obtain detailed help on the dialogs at any time with the online help.

Examples (see also syntax rules for symbols in the online help)

- Simple variables:
  tank
  temperature
- Structure elements
  tank.pressure1
  tank.temperature

You must first enter the symbols in the symbol table using the symbols editor of STEP 7! Your entries will only be accepted if they match the entry in the symbol table.

Assigning Access Rights

It is possible to assign access rights to the variables declared as symbols and these rights will be checked when the symbol is accessed. To set access rights, select the ”Edit...” button.

Notice

Access restrictions set here cannot be overridden by the EDIT applet parameter.

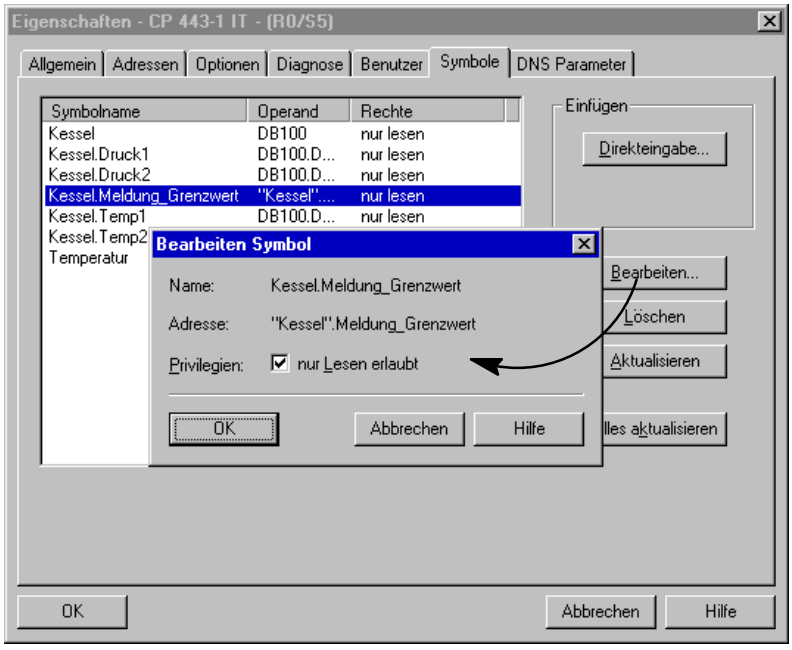


Figure 3-7

Printing the List of Variables

Along with the parameters of the IT-CP module, you can print out a list of variables in HW Config.

3.6 Testing and Using HTML Pages

Test: Java Console

With the Web browser, you can follow and log the running of Java applets in a Java Console.

The S7 applets used in your HTML pages only display error messages in the Java Console. These messages provide you with information about unexpected reactions in the display of your HTML pages.

To display warnings or even all the available messages on the console, you can set the level of detail using the DEBUGLEVEL applet parameter (see Section 4.3):

Possible settings

0 = no display

1 = display all messages

2 = display warning and error messages only

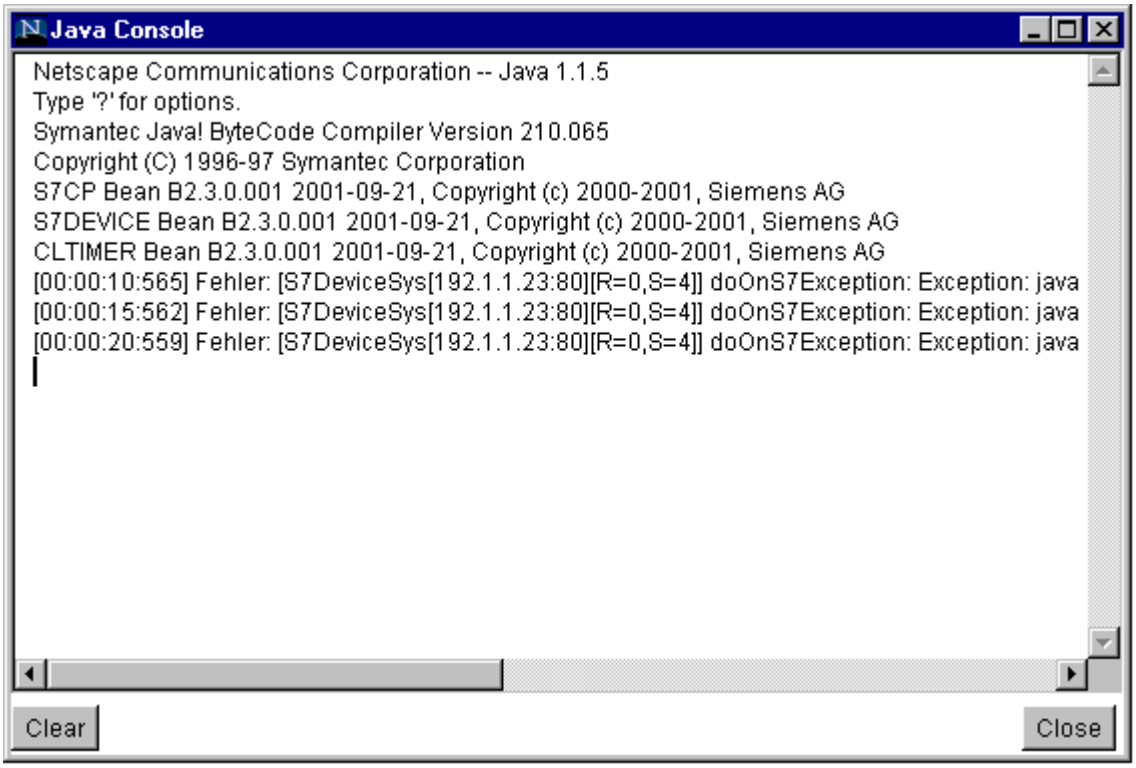
3 = display error messages only (default level)

4 = display only messages relating to fatal errors

if the parameter is not used in the applet call, level 3 is used; in other words, only error messages are displayed.

Notice

You should normally select Level 3 during normal operation! The other levels are intended only for test purposes.



Download: FTP for Transferring HTML Pages

Using FTP functions (see Section LEERER MERKER), you can download the HTML pages to the IT-CP and organize the files on the IT-CP to suit your requirements.

3.7 JavaScript Linking to the S7Applets

You can also access the values of the S7Applets with a JavaScript thanks to special methods integrated in the S7Applets. These methods are listed below:

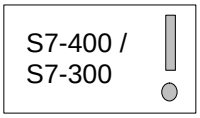
Table 3-20

S7Applet	Method	Return / Transfer Type
S7GetApplet	getValue ( )	java.lang.Object
S7PutApplet	setValue (param)	java.lang.Object
S7StatusApplet	getState ( ) *)	java.lang.String
S7IdentApplet	getIdent ( ) *)	java.lang.String
– All S7 Applets –	setRack (rack)	int
– All S7 Applets –	setSlot (slot)	int

*) Methods only with the S7-300/400

In the next three sections, you will find Web pages in HTML code in which the access to the S7Applets using JavaScript is illustrated in simple examples.

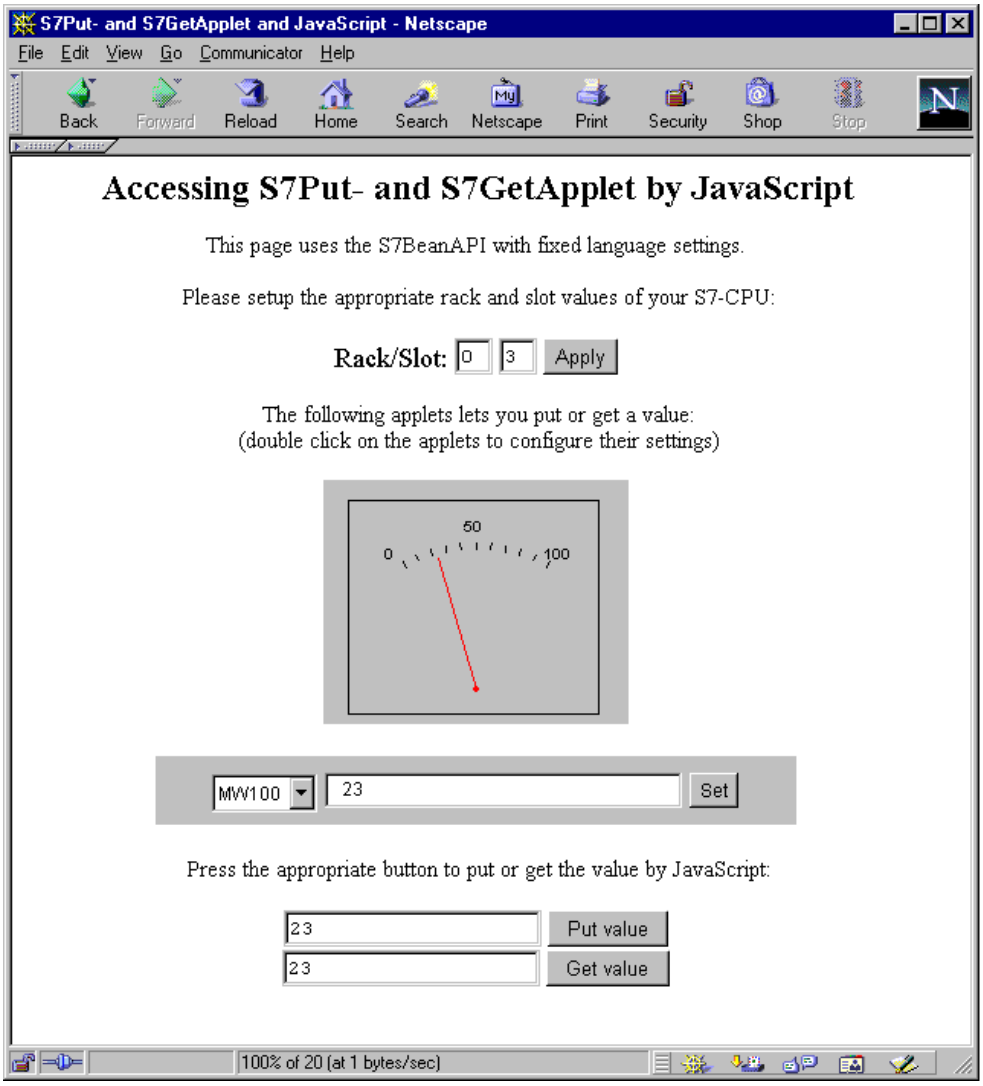
In these examples, the Event Handler onClick is used with which destination links can be defined in forms. The Event Handler is then activated when the user clicks on a form field in the HTML page.



The following HTML pages are included in the file system of the IT-CP in /examples/..

3.7.1 S7GetApplet and S7PutApplet

HTML example for writing and reading a value using JavaScript. The example includes an S7PutApplet for setting a value and an S7GetApplet allowing the written value to be verified.



The applets are set permanently to a particular language by the LANGUAGE parameter (in the example German or English). Messages in the Java console and the user interface elements of the applet are therefore also displayed in German/English.

The applets can also be edited directly. This means that if you double-click on the applet (edge), a configuration dialog opens in which you can change the most important applet parameters.

With the Rack/Slot input fields, you can adapt the configuration of the hardware. To

achieve this, the entered values are transferred to the two applets with JavaScript.

```
<HTML><HEAD>
<meta http–equiv=”Content–Type” content=”text/html; charset=iso–8859–1”>
<TITLE>S7Put and S7Get Applet with JavaScript</TITLE>

<script language=”JavaScript”>
<!—
 function setRackSlot(rack, slot) {
                document.Get.setRack(rack);
                document.Get.setSlot(slot);
                document.Put.setRack(rack);
                document.Put.setSlot(slot);

 }
//—>
</script>

</HEAD>

<BODY BGCOLOR=”#FFFFFF”>

<CENTER>
                <H2>Access to the S7Put and S7Get applets with
                   JavaScript</H2>
                <font size=+0>This page uses the S7BeanAPI with a
                  permanently set language.</font>

<FORM NAME=”config”>
                <font size=+0>Please enter values to suit your
                  S7-CPU:</font><br><br>
                <font size=+1>Rack/Slot: </font>
                <INPUT TYPE=”text” SIZE=”2” NAME=”rack” VALUE=”0”>
                <INPUT TYPE=”text” SIZE=”2” NAME=”slot” VALUE=”3”>
                <INPUT TYPE=”button” VALUE=”Apply”
                                onClick=”setRackSlot(document.config.rack.value,
                                  document.config.slot.value)”>
</FORM>

<font size=+0>With the following applets, you can write and read the values
   of the S7-CPU:</font><br>
<font size=+0>(double-click on the applets to change their
  settings)</font><br> <br>
<APPLET CODE=”de.siemens.simaticnet.itcp.applets.S7GetApplet.class”
CODEBASE=”/applets/”
                ARCHIVE=”s7applets.jar, s7api.jar” WIDTH=”200” HEIGHT=”160”
NAME=”Get”>
                <PARAM name=”RACK” value= 0>
                <PARAM name=”SLOT” value= 3>
                <PARAM name=”CYCLETIME” value= 10000>
                <PARAM name=”BACKGROUNDCOLOR” value=0xC0C0C0>
```

```
                <PARAM name="LANGUAGE" value="de">
                <PARAM name="DEBUGLEVEL" value=2>
                <PARAM name="DISPLAY" value="CLTacho">
                <PARAM name="MINVAL" value=0>
                <PARAM name="MAXVAL" value=100>
                <PARAM name="EDIT" value="true">

                <PARAM name="VARTYPE"     value=4>
                <PARAM name="VARCNT"      value=1>
                <PARAM name="VARAREA"     value=131>
                <PARAM name="VARSUBAREA"  value=0>
                <PARAM name="VAROFFSET"   value=100>
                <PARAM name="FORMAT"      value="MW100 = \H\H ">
</APPLET>

<br>
<br>

<APPLET CODE="de.siemens.simaticnet.itcp.applets.S7PutApplet.class"
CODEBASE="/applets/"
                ARCHIVE="s7applets.jar, s7api.jar" WIDTH="420" HEIGHT="45"
NAME="Put">
                <PARAM name="RACK" value= 0>
                <PARAM name="SLOT" value= 3>
                <PARAM name="BACKGROUNDCOLOR" value=0xC0C0C0>
                <PARAM name="LANGUAGE" value="en">
                <PARAM name="DEBUGLEVEL" value=2>
                <PARAM name="EDIT" value="true">

                <PARAM name="VARNUM"     value="1">
                <PARAM name="VARNAME1"    value="MW100">
                <PARAM name="VARTYPE1"    value="4">
                <PARAM name="VARAREA1"    value="131">
                <PARAM name="VARSUBAREA1" value="0">
                <PARAM name="VAROFFSET1"  value="100">
                <PARAM name="VARFORMAT1"  value="W">
</APPLET>

<br>

<FORM NAME="form1">
                <font size=+0>Click the appropriate button to set or read
the value with JavaScript:</font><br><br>
                <table width="25%" border="0" align="center">
    <tr>
      <td><INPUT TYPE="text" SIZE="20" NAME="inp"></td>
      <td align="center"><INPUT TYPE="button" VALUE=" Set Value "
        onClick="document.Put.setValue(document.form1.inp.value)"></td>
    </tr>
    <tr>
      <td><INPUT TYPE="text" SIZE="20" NAME="out"></td>
```

```
      <td align=”center”><INPUT TYPE=”button” VALUE=”Display value”
                                    onClick=”document.form1.out.value =
document.Get.getValue()”></td>
    </tr>
                </table>
</FORM>

</CENTER>
</BODY>
</HTML>
```

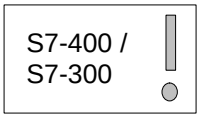
Note

Please note that the value you obtain with getValue() from the S7GetApplet is the value known to the S7GetApplet at this point in time. The value is not read directly from the PLC!

The value you transfer with the S7PutApplet using setValue() is passed on immediately to the underlying S7BeansAPI for transfer to the PLC.

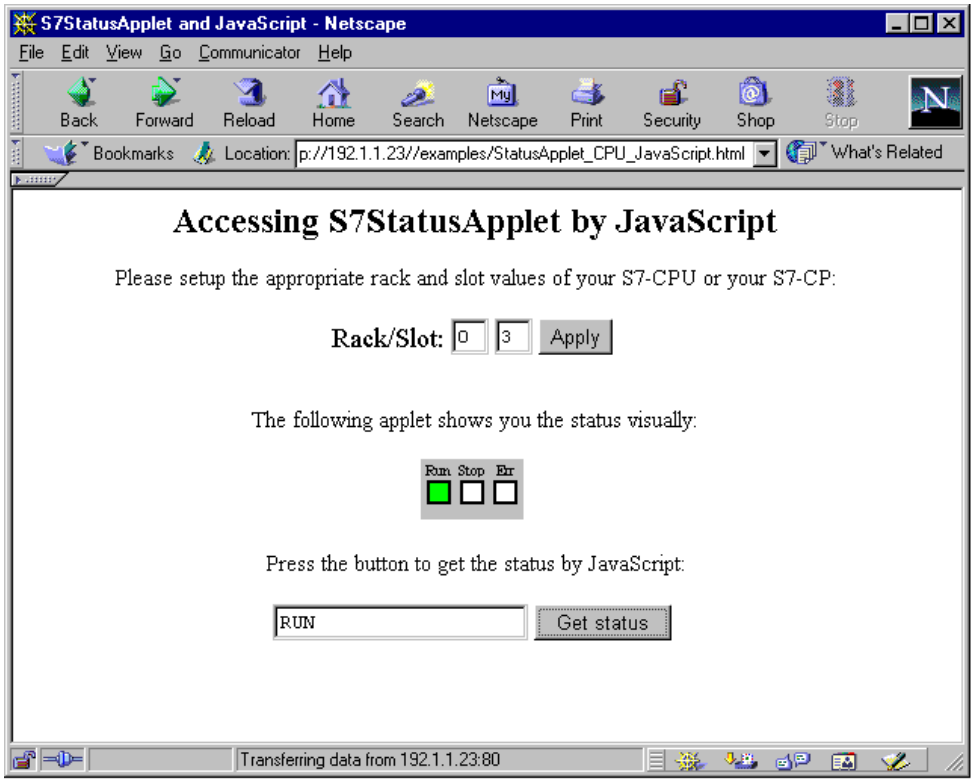
S7GetApplet and S7PutApplet are two programs that exist separately and do not communicate immediately with each other. This means that a value you wrote to the PLC with the S7PutApplet does not need to be known to the S7GetApplet directly.

3.7.2 S7StatusApplet



HTML example for access to the status of an S7 module.

The S7StatusApplet queries the status of the specified S7 module and displays it.




```
<HTML><HEAD>
<meta http–equiv=”Content–Type” content=”text/html;
charset=iso–8859–1”><TITLE>S7Status Applet and JavaScript</TITLE>
<script language=”JavaScript”>
<!—
  function setRackSlot(rack, slot) {
                    document.State.setRack(rack);
                    document.State.setSlot(slot);
 }
//—>
</script>
</HEAD>
<BODY  BGCOLOR=”#FFFFFF”>
<CENTER>
                    <H2>Accessing S7Status Applet by JavaScript</H2>
<FORM NAME=”config”>
                    <font size=+0>Please set the appropriate rack and slot values
                       of your S7–CPU or your S7–CP:</font><br> <br>
                    <font size=+1>Rack/Slot: </font>
                    <INPUT TYPE=”text” SIZE=”2” NAME=”rack” VALUE=”0”>
                    <INPUT TYPE=”text” SIZE=”2” NAME=”slot” VALUE=”3”>
                    <INPUT TYPE=”button” VALUE=”Apply”
                                        onClick=”setRackSlot(document.config.rack.value,
```

```
                                        document.config.slot.value)">
</FORM>
<br>
<font size=+0>The following applet shows you the status visually:</font><br> <br>
<APPLET CODE="de.siemens.simaticnet.itcp.applets.S7StatusApplet.class"
CODEBASE="/applets/"
                ARCHIVE="s7applets.jar, s7api.jar" WIDTH="68" HEIGHT="40"
                  NAME="State">
                <PARAM name="RACK" value= 0>
                <PARAM name="SLOT" value= 3>
                <PARAM name="CYCLETIME" value= 5000>
                <PARAM name="BACKGROUNDCOLOR" value=0xC0C0C0>
</APPLET>
<br>
<FORM NAME="form1">
                <font size=+0>Click the button to get the status by
                   JavaScript:</font><br> <br>
                <INPUT TYPE="text" SIZE="20" NAME="str">
                <INPUT TYPE="button" VALUE="Get status"
                                onClick="document.form1.str.value =
                                  document.State.getState()">
</FORM>
</CENTER>
</BODY>
</HTML>
```

Note

Please note that the status you obtain with getState() from S7StatusApplet is the status known to the S7StatusApplet at this point in time. The value is not read directly from the S7 module!

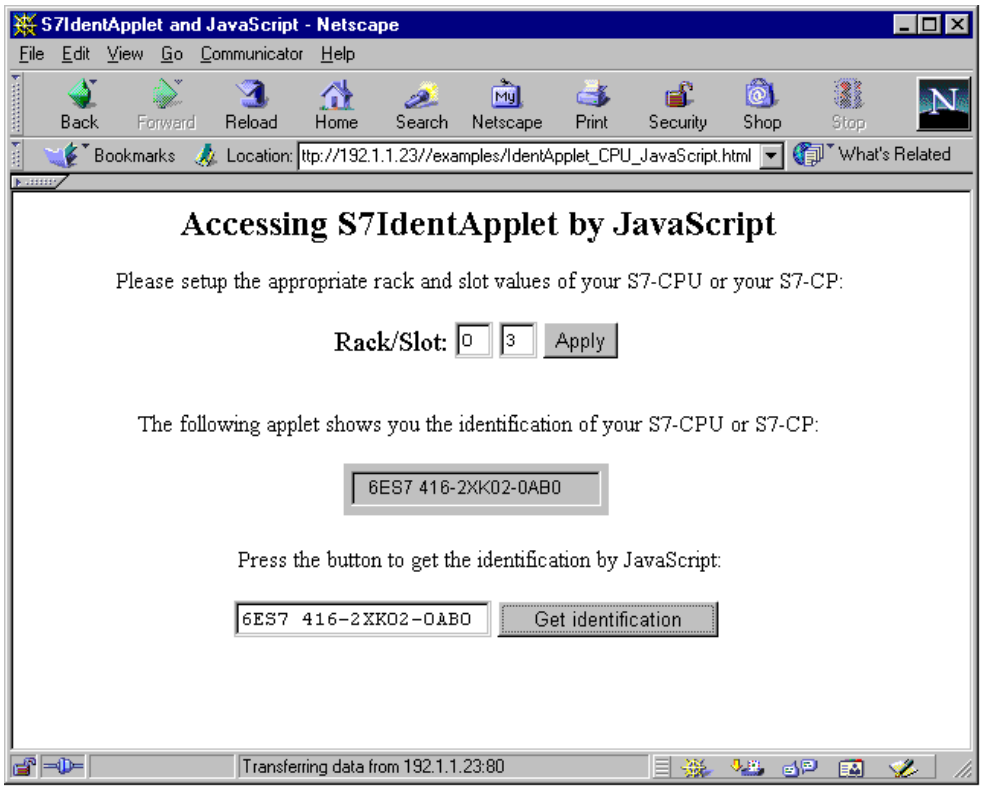
Remember also that not all S7 modules have the same status repertoire. The CP 443-1, for example, distinguishes between several causes of STOP (for example, STOP switch, internal STOP) while the CPU 416-1 outputs internal STOP in both cases (since it does not distinguish the STOP switch).

3.7.3 S7IdentApplet

S7-400 / S7-300

HTML example for querying the order (MLFB) number of an S7 module.

The S7IdentApplet reads the order number (MLFB) of the specified S7 module and displays it.




```
<HTML><HEAD>
<meta http–equiv=”Content–Type” content=”text/html;
charset=iso–8859–1”><TITLE>
  S7IdentApplet and JavaScript</TITLE>

<script language=”JavaScript”>
<!—
 function setRackSlot(rack, slot) {
                    document.Ident.setRack(rack);
                    document.Ident.setSlot(slot);

 }
//—>
</script>

</HEAD>
```

```
<BODY BGCOLOR="#FFFFFF">

<CENTER>
        <H2>Accessing S7Ident Applet by JavaScript</H2>

<FORM NAME="config">
        <font size=+0>Please set the appropriate rack and slot values of
your S7–CPU
          or your S7–CP:</font><br> <br>
        <font size=+1>Rack/Slot: </font>
        <INPUT TYPE="text" SIZE="2" NAME="rack" VALUE="0">
        <INPUT TYPE="text" SIZE="2" NAME="slot" VALUE="3">
        <INPUT TYPE="button" VALUE="Apply"
          onClick="setRackSlot(document.config.rack.value,
document.config.slot.
            value)">

</FORM>

<br>
<font size=+0>The following applet shows you the identification of your S7–CPU or
  ,S7–CP:</font><br> <br>
<APPLET CODE="de.siemens.simaticnet.itcp.applets.S7IdentApplet.class"
CODEBASE=
  "/applets/"
        ARCHIVE="s7applets.jar, s7api.jar" WIDTH="174"
          HEIGHT="34" NAME="Ident">
        <PARAM name="RACK" value= 0>
        <PARAM name="SLOT" value= 3>
        <PARAM name="BACKGROUNDCOLOR" value=0xC0C0C0>

</APPLET>

<br>

<FORM NAME="form1">
        <font size=+0>Click the button to get the identification by
JavaScript:</font><br>
          <br>
        <INPUT TYPE="text" SIZE="20" NAME="str">
        <INPUT TYPE="button" VALUE="Get identification"
                  onClick="document.form1.str.value =
                    document.Ident.getIdent()">
</FORM>

</CENTER>
</BODY>
</HTML>
```

3.8 Help on the S7 Applets

This appendix explains how errors can occur when using the S7 applets.

If errors occur during operation, you will see messages displayed in the Java console (see Section 2.2 and 3.6).

All S7 Applets

- The name of the applet class, the CODEBASE parameter, or the ARCHIVE parameter were not or incorrectly specified (check upper– /lowercase).
- The width and/or height of the applet was not specified or the value was too high or low.
- The syntax of the parameter tag <PARAM NAME="..." VALUE="..."> is wrong.
- You have forgotten a parameter or written it incorrectly.
- The BACKGROUND parameter for the background color of the applet is missing or is not in the valid range from 0x000000 to 0xFFFFFF.
- The user does not have rights to run the applet.
- The RACK and SLOT parameters for the rack number and the slot number do not match the rack and/or slot in which the module is actually located.

Only S7StatusApplet and S7IdentApplet

- The addressed module is not a CPU / CP

S7GETApplet and S7StatusApplet Only

- No CYCLETIME parameter for the cycle time or is not an integer.

S7GETApplet and S7PUTApplet Only

- Symbol table not found
- A specified symbol was not found in the symbol table.
- The ANY pointer contains invalid values.
- The type of the specified parameter does not match the type expected (for example, integer, floating–point, string).

S7GETApplet Only

- The format string has a syntax error (for example, an unknown formatting character).
- The format string does not match the length of the data fetched from the CPU.
- The maximum and/or minimum value is not specified by the parameters MINVAL and/or MAXVAL if an S7 bean is used.
- The value of the MAXVAL parameter is less than or equal to the value in the MINVAL parameter.

S7PUTApplet Only

- The information in the SYMBOLNUM parameter and/or VARNUM does not match the actual symbol specified or the ANY pointer.
- A specified symbol has the read–only attribute in the symbol table

S7StatusApplet Only

- The addressed module is not capable of supplying its module status.
- If you want to display the status of more than one module and the cycle time selected is too short, you will often experience problems establishing the connection.

❑

4 Individual Solutions with JavaBeans

In many situations, you will probably prefer to see a graphic visualization of the process values rather than a simple display of the numbers. In the plant pictures on operator control and monitoring systems, such display forms, for example level displays or temperature scales are normal.

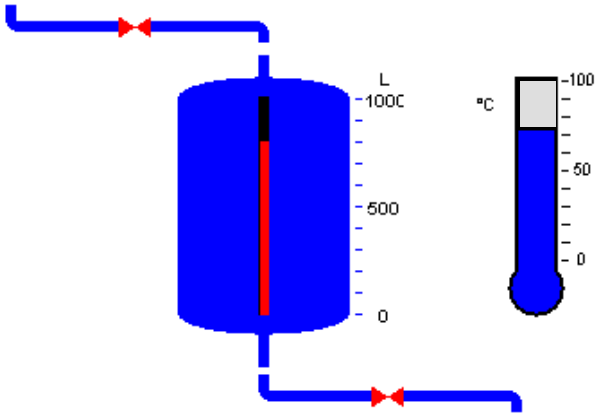


Figure 4-1

The JavaBeans concept in Java allows objects (Java components) to be created and linked simply to executable programs. JavaBeans also allow extremely flexible options for accessing process data on a SIMATIC S7 CPU.

As the IT-CP user, you have an S7 beans class library (S7BeansAPI) available for Java programming. The object classes contained in this S7 beans class library can be used for object-oriented access to a variety of information on the SIMATIC S7 and can be used for graphic display of process variables.

4.1 JavaBeans Concept and Possible Applications

Standard Application: You can use S7 beans along with the S7GetApplet

Simply by using the “S7GetApplet” of the IT-CP, you can use a range of graphic elements from the S7 beans class library for display on your HTML page using the DISPLAY parameter (refer to the DISPLAY parameter in Section 3.4.7). The S7 beans to be used are marked in Table 4-2 in the following section.

Extended Applications: Configure S7 beans using “Builder Tools”

To be able to use the full range of S7 beans in the S7 bean class library, use builder tools in your Java programming environment.

With these builder tools you can create an object-oriented configuration for your application. In simplified form, this involves the following steps:

- Select the required S7 beans
- Link the S7 beans together to specify the data flow
- Specify the parameter settings for the S7 beans

Using this technique, you can configure and program complex process displays in your IDE with little effort.

For the experienced Java user, this provides almost unlimited options for further processing of the data acquired by the IT-CP, for example, in databases or management information systems.

Note

When developing new applets and your own beans, check the run–time version in the file system and the development version on the Manual Collection CD.

4.2 The S7 Beans Class Library (S7BeansAPI)

Application: Manual Collection CD

The S7 beans class library is on the Manual Collection CD.

The Available S7 Beans

The following tables provide you with an overview of the S7 beans supplied at the current time. Based on this table, you will get an impression of the configuration options provided by these S7 beans.

A distinction must be made between the following:

- S7 Beans for Devices

  These JavaBeans in the S7BeansAPI are provided for the modules and software objects that can be addressed the SIMATIC S7 rack. These provide the link in the program to the S7 beans for input and output (S7 beans for the client).

- S7 Beans for the Client

  These Java beans are available in the S7BeansAPI for graphic output of process data in process pictures on the client

- S7 Utility Beans for the Client

  Utility beans are available for additional preparation of data. Since these are simply conversion functions, there is no direct graphic display by these beans.

  These S7 beans are in the JAR file s7util.jar.

Table 4-1 S7 Beans for Devices / Objects Package = API (included in the JAR file s7api.jar)

S7 Bean	Function
S7CP	This bean represents the IT–CP serving as the host. Any other IT–CPs that exist must be addressed using S7Device. This bean must be used with each applet for addressing and for saving the host address.
S7 Device	S7Device represents any intelligent S7 module such as a CPU, PROFIBUS CP, Ethernet CP, other IT–CPs (however, under no circumstances the IT–CP serving as host for the applets, to be addressed using the browser!)
S7 Variable	This bean represents variables in the S7 CPU.
CLTimer	CLTimer is required for cyclic calling of methods of other beans. Whenever you want to monitor the status of an S7 module or a process variable continuously (cyclically), you require this bean. Note: CLTimer has **no** graphic representation

Table 4-2 S7 Beans for the Client – Package = GUI (included in the JAR file s7gui.jar)

S7 Bean	Function	Can be used in S7GetApplet	representation
CLTextIn	CLTextIn is a bean for entering text. This text can be passed on to the S7 variable bean.	no	– Input Field –
CLTextOut	CLTextOut is a bean for the textual output of values of process variables of the S7 variable bean	no	
CLIdentOut (S7-300/400 only)	CLIdentOut is a bean required for the textual display of an identification number of an IT-CP or module using the S7 CP bean or s7 device bean. Note: Not all modules support this service.	no	
CLStateLED (S7-300/400 only)	CLStateLED is a bean for graphic representation of the status of an IT-CP or a module. This is represented by the color of the LED: • green: RUN • yellow: STOP • red: Error message from the module • blue: Connection error • gray: Status unknown Note: Not all modules support this service.	no	
CLState3LED (S7-300/400 only)	CLState3LED is a bean for graphic representation of the status of an IT-CP or a module. The representation is in the form of three LEDs: • green: RUN • yellow: STOP • red: Error message from the module • blue: Connection error • gray: Status unknown Note: Only intelligent modules support this service; I/O modules, for example, do not.	no	

Table 4-2 S7 Beans for the Client, Fortsetzung – Package = GUI (included in the JAR file s7gui.jar)

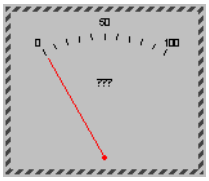
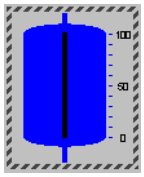
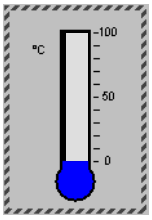
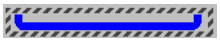
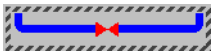
S7 Bean	Function	Can be used in S7GetApplet	representation
CLTacho	CLTacho is a bean for graphic representation of a pointer instrument. The pointer represents the value of a process variable. It is also possible to visualize a value reaching an upper or lower limit value with an LED display. The size of the bean is scalable.	yes	
CLLevel	CLLevel is a bean for graphic representation of the level of a process variable. It is also possible to visualize a value reaching an upper or lower limit value with an LED display. The size of the bean is scalable.	yes	
CLThermo	CLThermo is a bean for graphic representation of a process variable as a temperature value. It is also possible to visualize a value reaching an upper or lower limit value with an LED display. The size of the bean is scalable.	yes	
CLPipe	CLPipe is a bean for graphic display or a horizontal or vertical pipe. The color of the pipe changes with a Boolean value. The size of the bean is scalable.	no	
CLValve	CLValve is a bean for graphic representation of a valve The valve with its inlet can be displayed horizontally or vertically. The valve is opened and closed by setting a Boolean value.	no	

Table 4-3 S7 Utility Beans Package = UTIL (included in the JAR file s7util.jar)

S7 Utility Bean	Function
COUNTER	Supplies the counter reading (for example, C1) as a string in the format C#347.
TIMER	Supplies the value of timers (for example, T1) as a string in the format S5T#1h3m2s.

Table 4-3 S7 Utility Beans Package = UTIL (included in the JAR file s7util.jar), continued

S7 Utility Bean	Function
DATE	Supplies the S7 type DATE as a string in the format D#2000–12–31.
TIME	Supplies the S7 type TIME as a string in the format T#9h6m6s.
DATEandTIME	Supplies the S7 type DATE_AND_TIME as a string in the format DT#00–12–31–12:31:47.487.
TIMEofDAY	Supplies the S7 type Time Of Day as a string in the format TOD#9:6:6.127.
S5TIME	Supplies the S7 type S5TIME as a string in the format S5T#1h3m2s.
ConvertNumberSystem	Supplies a decimal number in either hexadecimal, octal, or binary format (as a string). Decimal numbers can also be converted to BCD numbers and vice versa.

4.3 Linking S7 Beans

Before your S7 beans can interact in your application, they must be connected together. Using tools such as Visual Age that is introduced in greater detail in Chapter 5, you can make these connections with graphic support.

As a general introduction, we will first show you an overview of useful connections between the beans.

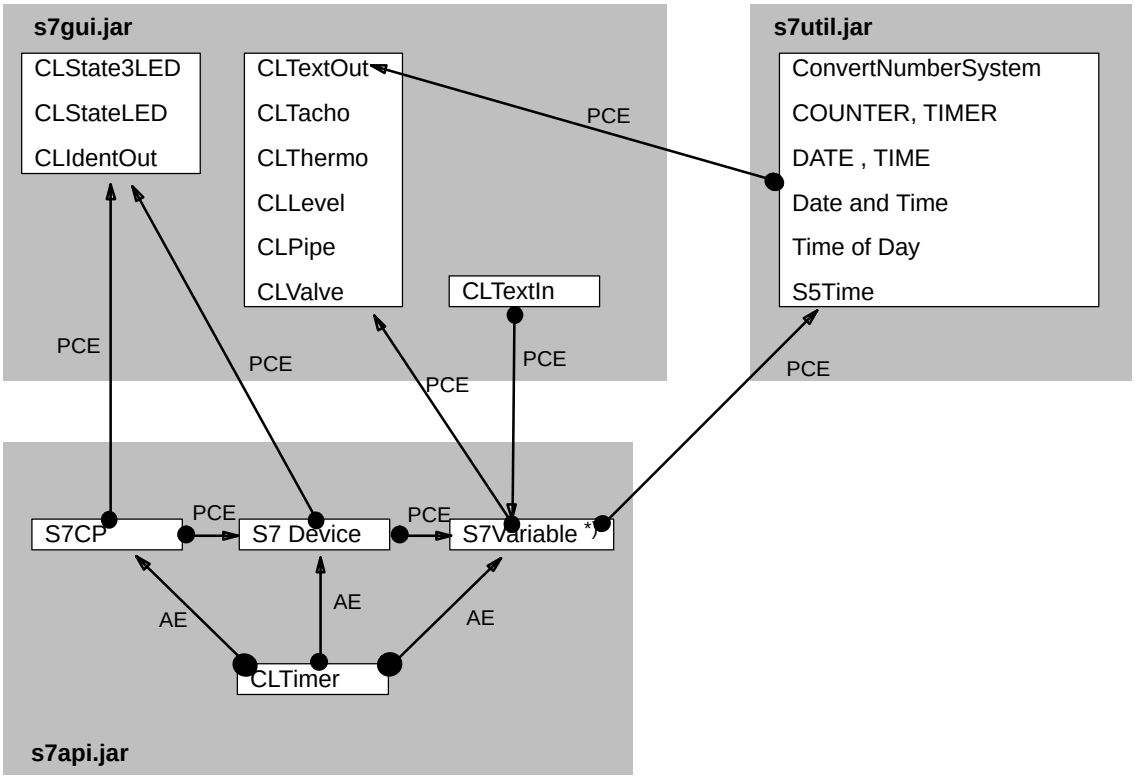
The following graphic shows which connections are possible; the abbreviations used are

AE: ActionEvent

PCE: PropertyChangeEvent

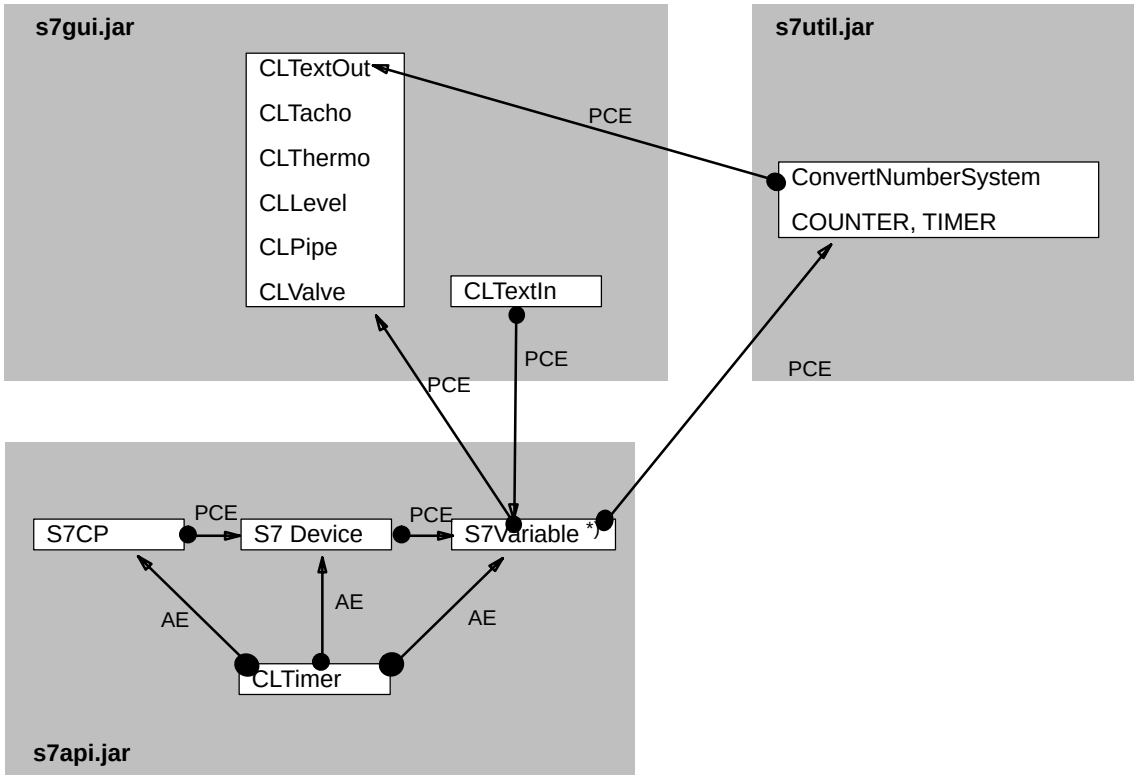
The producer of the event is shown at the end of the arrow with the dot and the tip of the arrow points to the listener.

Linking S7 Beans on the CP 343–1 IT and CP 443–1 IT



*) For more information on the variable types with S7–300 / S7–400, please refer to tables in Chapter C

Linking S7 Beans on the CP 243–1 IT



*) For more information on the variable types with S7–200, please refer to tables in Chapter C

❑

5 S7BeansAPI – Interface Description

This chapter describes the interfaces of the ”invisible” S7API and S7Beans for devices. You will learn how to create instances of the beans (Constructor calls) and which access methods exist.

The S7 beans for input and output of values pass or receive their values using a PropertyChangeEvent (see Section 4.3: Connecting S7 Beans).

For more information, you should also read the JavaDoc for the S7BeansAPI supplied on the Manual Collection CD.

5.1 S7API Methods

As of S7BeansAPI version V2.3, the **`de.siemens.simaticnet.itcp.api`** package contains the new class **S7Api**. This class contains a series of useful auxiliary functions that are introduced below.

5.1.1 Controlling the Language Used

With the **S7Api.setLocale(String newLanguage, String newCountry)** method, the language used for messages and labels can be set at run time. The **newLanguage** parameter is a language code defined according to ISO–639 and **newCountry** is the country code according to ISO–3166. The S7BeanApi ships with English and German resource files. These language files are located in the relevant JAR archives and have the extension *.properties. Based on these resource files, you can also create your own translations so that you can use languages other than English and German. Below there is an example of a **setLocale()** method call as you could use it in the **init()** method of one of your applets:

```
S7Api.setLocale(”de”, ””);
```

This call sets the language of the S7BeansAPI to neutral German, since no country code is specified.

Note

If no language is set, the S7BeansAPI uses the IDE language of the host if this exists just as in the development phase of the IDE.

5.1.2 Setting the Level of Detail for Debug Output

To avoid resource problems, as of S7BeansAPI version V2.3, the default setting is that debug messages in the Java console are displayed only for errors. With the new S7Api method **`setDebugLevel(int level)`**, you can specify the level of detail at run time. The following values can be passed for the **level** integer parameter:

0 = VOID_LEVEL: Display no messages

1 = LOG_LEVEL: All possible messages

2 = WARN_LEVEL: Display warnings and messages only

3 = ERR_LEVEL: Display error messages only (default)

4 = FATAL_LEVEL: Display fatal errors only

With the following call, for example, only warnings and error messages are displayed:

```
S7Api.setDebugLevel(2);
```

Note

The messages (warnings, errors etc.) are displayed with time stamp and object information. This makes it easier to analyze problems and the run time response.

5.1.3 Terminating the API Mechanisms

To avoid resource problems when using the applets developed with the beans, all resources should be released when the relevant HTML page is exited and all threads stopped. Since the S7BeansAPI uses static resources and threads internally for communication with the IT–CP, the **terminate()** method was created in the **S7Api** class to release these resources and to stop all threads.

Terminate() can normally be called after releasing all its own resources in the destroy() method of the applet. Since some browsers (for example, Netscape) do not call the destroy() method immediately when the HTML page is exited, but only when the applet is removed from the History cache of the browser, it is advisable to release the resources in the stop() method and to call terminate() from the S7Api class immediately. This does, however, require reinitializing all resources in the start() method of the applet because the applet, for example under Netscape, is also stopped and started again by changing the browser window size.

5.2 S7 Beans

5.2.1 S7CP

Description

The S7 bean S7CP is the component for connecting to the CP 343–1 IT / CP 443–1 IT via which you can access the PLC.

(Note: From one applet you can only ever access **the** IT–CP from which the applet was loaded (Sandbox); if there are other IT–CPs in the PLC, you can only access these using the S7Device S7 bean).

Creating an Instance – Constructor Call:

To create an instance of the S7CP, start the Standard Constructor and then set the properties **host** and **modulName**.

Constructor Call:

As an alternative, you can also call the following Constructor and set the parameters directly. Please note, that the parameters rack and slot must both be set to 0 (zero) in this case.

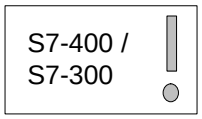
S7CP(String modulName, String host, int rack, int slot)

Setting Properties:

If you used the Standard Constructor, set the following properties:

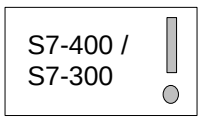
- setModulName(String modulName): Unique name for the instance.
- setHost(String host): IP address of the IT–CP.

Calling up the Identification of the Module (MLFB) (S7–300 / S7–400 only):



processIdent(): Initiates the MLFB number query. The method terminates immediately. As soon as the required information is available, it is sent to the listener using a PropertyChangeEvent (propertyName = identification).

Calling up the Status Information of the Module (S7–300 / S7–400 only):



processState(): Initiates the status information query. The method terminates immediately. As soon as the required information is available, it is sent to the listener using a PropertyChangeEvent (propertyName = state).

5.2.2 S7Device

Description

The S7 bean S7Device represents an (intelligent) module, such as a CPU or a CP. An S7Device can also be a further IT–CP via which you do not want to access the PLC from within the applet.

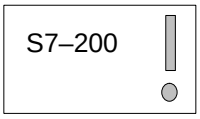
Each S7Device must be logged on with S7CP as PropertyChangeListener! You can log on several S7Devices at one S7CP.

Creating an Instance – Constructor Call:

To create an instance of the S7Device, start the Standard Constructor and then set the properties *rack, slot* and *modulName.*

Setting Properties:

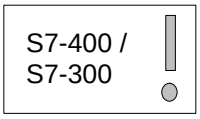
- setModulName(String modulName): Unique name for the instance.
- setRack(int rack): Number of the rack.
- setSlot(int slot): Number of the slot.



Notice

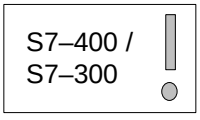
For an S7–200, please specify: rack =0; slot =0

Calling up the Identification of the Module (MLFB) (S7–300 / S7–400 only):



processIdent(): Initiates the MLFB number query. The method terminates immediately. As soon as the required information is available, it is sent to the listener using a PropertyChangeEvent (propertyName = identification).

Querying Status Information of the Module:



processState(): Initiates the status information query. The method terminates immediately. As soon as the required information is available, it is sent to the listener using a PropertyChangeEvent (propertyName = state).

5.2.3 S7Variable

Description

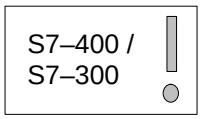
The S7 bean S7Variable represents a memory area on the CPU, that you either address using an ANY pointer or a symbolic address (the latter must be entered in the hardware configuration of the IT–CP).

Each S7Variable must be logged on at the relevant S7Device as PropertyChangeListener. You can log on several S7Variables at one S7Device.

Constructor Call:

To create an instance of the S7Variable, start the Standard Constructor and then set the properties *symbolName* or *s7AnyPointer* and *variableName.*

Setting Properties:



- setSymbolName(String symbolName): Symbolic address.
- setS7Anypointer(S7Anypointer anypointer): ANY pointer address.
- setVariableName(String variableName): Unique name for this instance.

The symbolic addressing and the ANY pointer addressing are alternatives. If both are specified, the symbolic address has priority.

Reading a Memory Area:

processGet(): Initiates the request to read a memory area. The method terminates immediately, the read job runs asynchronously. As soon as the requested information is available, this is sent to the listener using a PropertyChangeEvent (propertyName: the string entered in the *variableName* property).

getValue() object: Delivers the content of the memory area stored in the S7Variable. Remember, however, that this method does not necessarily supply the current content of the addressed memory area of the PLC! If no value was read previously with *processGet()*, no value will be returned here!

Writing a Memory Areas:

setValue (newValue object): Initiates writing a value to the memory area of the CPU. The method terminates immediately, the write job is asynchronous.

5.2.4 CLTimer

Description

The CLTimer S7 bean creates an action event cyclically and can therefore be used as a trigger for the periodic reading of status information and memory areas.

Each S7 bean that executes an action periodically must be logged on at the CLTimer as an ActionListener. You must specify which method will be executed when the event arrives (for S7CP and S7Device, this is **processState()** a for S7Variable, this is **processGet()**). Several beans can be logged on with the CLTimer as ActionListeners. It is also possible to use several instances of CLTimer (for example, with different times).

Constructor Call:

To create an instance of CLTimer, call the Standard Constructor and then, if necessary, the *delay* property.

Setting the Property:

setDelay(int delay): Cycle time in milliseconds (default value is 5,000 ms).

Controlling CLTimer:

start(): Restarts the CLTimer after a stop() call. Following instantiation, the CLTimer starts automatically and does not need to be triggered with start().

stop(): Stops the CLTimer.

6 Examples:

6.1 Examples with S7Beans and AWT Components

The following sections contain examples that illustrate the use of the S7Beans in your own applets.

The examples are printed with their full source text and can be run. Only the IP address of the IT–CP and the rack/slot number of the S7–CPU must be adapted in the source text.

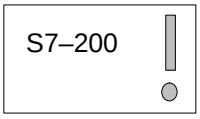
You can import the samples in a Java development environment such as VisualAge from IBM or JBuilder from Borland or compile them directly with the Java Development Kit (JDK). You then transfer the finished applet with to your IT–CP with FTP. You also require a suitable HTML page that will serve as container for the applet and allows the sample applet to be called.

Below you will see a simple HTML page that represents only a title page and starts the applet 1 example:

```
<HTML>
<HEAD>
<TITLE>Example 1</TITLE>
</HEAD>
<BODY>
<H1>Example 1</H1>
<APPLET CODE=de.siemens.simaticnet.itcp.example.Example1.class
ARCHIVE=Examples.jar WIDTH=200 HEIGHT=100>
</APPLET>
</BODY>
</HTML>
```

For the other examples, you only need to adapt the number after ”Example”. The ”ARCHIVE” entry assumes that the HTML page and the corresponding sample applet are in the same path on the IT–CP.

Using the Examples with an S7–200 / CP 243–1 IT



Notice

The examples described here can be used for the IT–CPs documented here. At some points, however, the parameter settings are suitable only for the S7–300 and S7–400.

If you want to use them with a CP 243–1 IT, note the following:

- When specifying bit memory areas, adapt the entries to the valid bit memory areas of the S7–200!
- Note the points at which specific entries are necessary, for example identifying slots.
- Rack/Slot number of the addressed module with S7-200 is always R/S=0/0

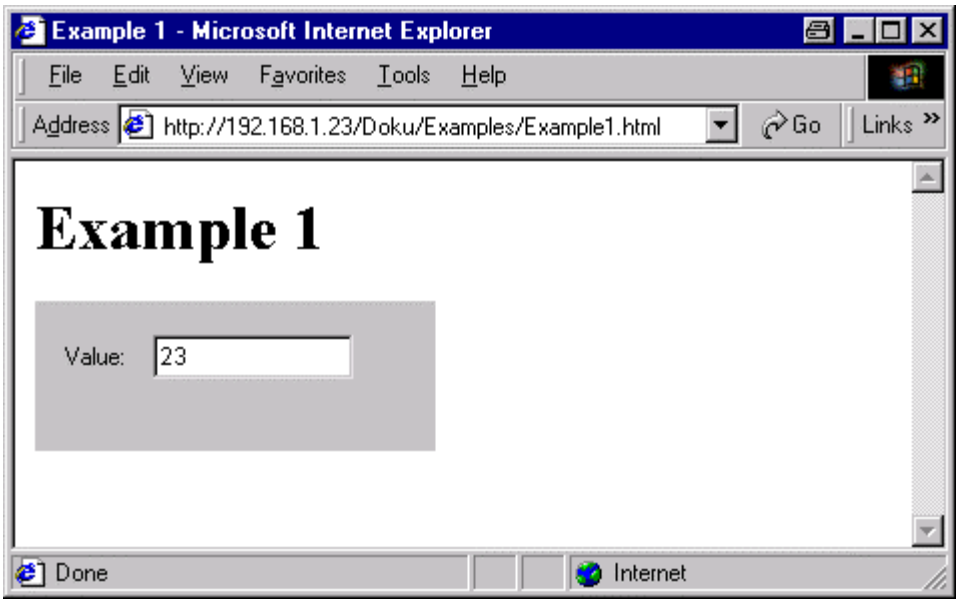
6.1.1 Example 1 – Reading and Displaying a Variable from the S7 CPU with S7Beans

How the Function Works

A variable will be read from the SIMATIC S7 CPU and displayed.

The memory word MW 10 will be read as INT.

representation



Source Text

```
// This statement assigns all classes whose source text contains
// this statement to a package.
package de.siemens.simaticnet.itcp.example;
// By importing a package or a class, all declarations
// are made visible that may be made visible in other packages
// based on their access class.
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.beans.*;
import de.siemens.simaticnet.itcp.api.*;
import de.siemens.simaticnet.itcp.gui.*;
/**
 * Example1.java
 * <p>Title: Example 1 - Using the ITCP Beans.</p>
 * <p>Description: Outputting a value using the CLTextOut Bean with an S7Variable.</p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Organization: Siemens AG SIMATIC NET</p>
 *
 * Outputting a value using the CLTextOut Bean with an S7Variable.
```

```
 * The memory word MW10 is read.
 *
 * Components used:
 * S7CP
 * S7Device
 * S7Variable
 * CLTimer
 * CLTextOut
 *
 * @author ITCP Team
 * @version 1.0
 *
 */
public class Example1 extends Applet implements PropertyChangeListener, ActionListener {
   /*-----------------------------------Implemented Interfaces*/
   /*----------------------------Basic Class Applet*/
   // Declaration of the required components
   private CLTextOut cLTextOut1 = null;
   private CLTimer cLTimer1 = null;
   private S7CP s7CP1 = null;
   private S7Device s7Device1 = null;
   private S7Variable s7Variable1 = null;
   /**
    * Is always called when the applet is initialized.
    * This occurs immediately after it is loaded.
    *
    * @see #start
    * @see #stop
    * @see #destroy
    */
   public void init() {
      super.init();
      // Sets the name of the component to the specified character string.
      setName("Example1");
      // Sets the layout manager for this component.
      setLayout(null);
      // Sets the size of the applet width / height
      setSize(426, 240);
      /*-----------Height*/
      /*------Width*/
      // Create an instance for the S7CP bean.
      // S7CP is the Ethernet access point to the station
      s7CP1 = new S7CP();
      // Assign the IP address
      // ######## Project-specific adaptation of the IP address necessary          ########
      s7CP1.setHostString(new HostString ("192.168.1.1:80"));
      /*-------------------------------------------------Specify port number
                                                        normally :80*/
      /*-----------------------------------IP address as string*/
      // Create an instance for the S7Device bean.
      // S7Device is used to address the communication partner in the station.
      s7Device1 = new S7Device();
      // The address is made up of the rack number and slot number of the
      // module. The default of both methods for addressing is '0'.
      // This means that the rack number is unnecessary (.setRack(0)).
      // As we want to communicate with the CPU, the slot of the CPU must
      // be entered.
      // ######## Project-specific adaptation of the rack and slot number  ########
      // ######## necessary                                                ########
      s7Device1.setSlot(2);
      /*----------------Slot number 2 (int)*/
      // Create an instance for the S7Variable bean.
      // The S7Variable bean represents the variable to be read
      // or written.
```

```
    s7Variable1 = new S7Variable();
    // The variable is described by an S7 ANY pointer
    s7Variable1.setS7Anypointer(
       new S7Anypointer((int)5, (int)1, (int)131, (int)0, (int)10, (int)0));
    /*-------------------------------------------------------------Bit number 0 ..7*/
    /*-----------------------------------------------------Memory area offset*/
    /*-------------------------------------------DB number or '0'*/
    /*-----------------------------------Memory area 131 == M*/
    /*--------------------------Repetition factor 1 .. n*/
    /*------------------Data type 5 == INT*/
    // Sets the name of the component to the specified character string.
    s7Variable1.setVariableName("s7Variable1");
    // Create an instance for the CLTimer bean.
    // The CLTimer bean triggers a PropertyChangeEvent when
    // the time elapses. This event implements a cyclic data
    // update.
    cLTimer1 = new CLTimer();
    // The setDelay() method sets the time interval.
    cLTimer1.setDelay(2000);
    /*----------------Time interval in msec. 2000 == 2 sec.*/
    // Create an instance for the CLTextOut bean.
    // Using the CLTextOut bean, you as user can enter elementary
    // variables.
    cLTextOut1 = new CLTextOut();
    // Sets the name of the component to the specified character string.
    cLTextOut1.setName("cLTextOut1");
    // Specify the start position and size of the component.
    cLTextOut1.setBounds(0, 0, 200, 45);
    /*----------------------------Component height*/
    /*-------------------------Component width*/
    /*----------------------Start position Y*/
    /*-------------------Start position X*/
    // Specify the size of the output box.
    cLTextOut1.setOutFieldSize(100);
    /*-------------------------Length of the output box*/
    // Sets the descriptive text to the specified character string.
    cLTextOut1.setLabel("Value:");
    // Sets the dimension text to the specified character string.
    cLTextOut1.setUnit("");
    // Insert the component in the applet.
    add(cLTextOut1, cLTextOut1.getName());
    // Apart from the definition of the methods to be executed when an event
    // occurs, an object must register with the corresponding
    // event source.
    // This is done by calling the addXXXListener method of the event source,
    // where »XXX« stands for the corresponding event type.
    // The addXXXListener methods all expect a reference to the relevant
    // interface:
    s7CP1.addPropertyChangeListener(this);
    s7Device1.addPropertyChangeListener(this);
    cLTimer1.addActionListener(this);
    s7Variable1.addPropertyChangeListener(this);
}
/**
 * Executes after initialization of an applet.
 * With browsers, start() is then also called when a page containing
 * an applet is reloaded.
 *
 * @see #init
 * @see #stop
 * @see #destroy
 */
public void start() {
   super.start();
```

```
}
/**
 * There is a call when the browser or the applet viewer is minimized
 * to an icon or an HTML page containing an applet is exited in
 * a browser.
 *
 * @see #init
 * @see #start
 * @see #destroy
 */
public void stop() {
   super.stop();
}
/**
 * Is always called when the applet is destroyed.
 *
 * @see #init
 * @see #start
 * @see #stop
 */
public void destroy() {
   super.destroy();
   // This method deletes all S7Bean instances and discards all threads.
   // After calling this method, a reinitialization is necessary.
   S7Api.terminate();
}
/**
 * Method for handling events for the PropertyChangeListener interface.
 *
 * @param evt PropertyChangeEvent
 */
public void propertyChange(PropertyChangeEvent evt) {
   // Query whether event was triggered by S7CP.
   if (evt.getSource() == s7CP1)
      // If YES
      // Pass event to the S7Device instance
      s7Device1.propertyChange(evt);
   // Query whether or not event was triggered by S7Device.
   if (evt.getSource() == s7Device1)
      // If YES
      // Pass event to the S7Variable instance
      s7Variable1.propertyChange(evt);
   // Query whether or not event was triggered by S7Variable.
   if (evt.getSource() == s7Variable1)
      // If YES
      // Then transfer output value to the CLTextOut instance.
      cLTextOut1.propertyChange(evt);
}
/**
 * Method for handling events for the ActionListener interface.
 *
 * @param e java.awt.event.ActionEvent
 */
public void actionPerformed(ActionEvent e) {
   // Query whether CLTimer triggered.
   if (e.getSource() == cLTimer1) {
      // If YES, read values from the PLC.
      // Reading is triggered by the processGet() method
      // of S7Variable bean.
      // If the new values exist, the S7Variable bean triggers a
      // PropertyChangeEvent.
      s7Variable1.processGet();
   }
}
```

```
}
```

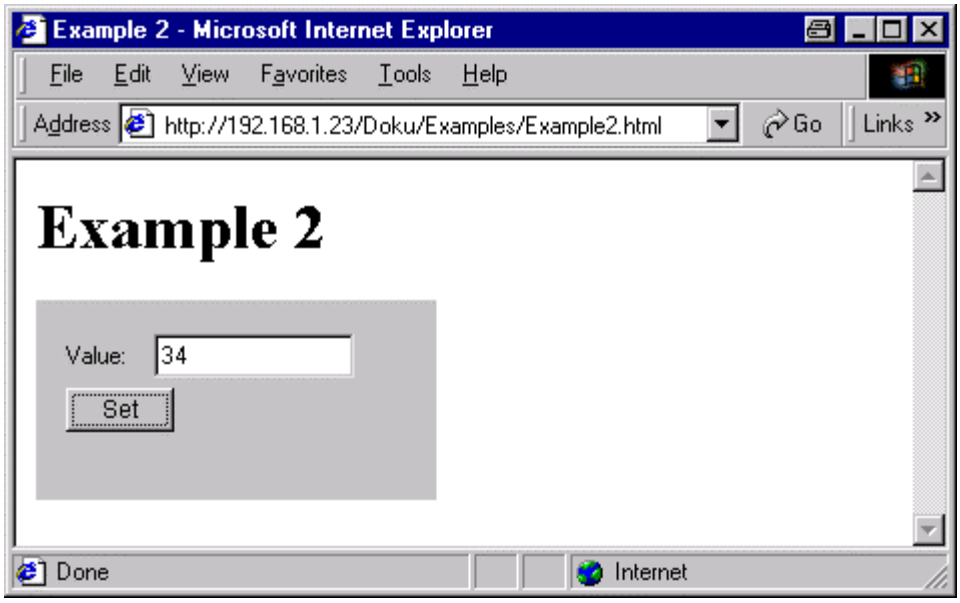
6.1.2 Example 2 – Writing a Variable to the SIMATIC S7 with S7Beans

How the Function Works

A variable will be written to the SIMATIC S7.

The memory word MW 10 will be written as INT.

representation



Source Text

```
// This statement assigns all classes whose source text contains
// this statement to a package.
package de.siemens.simaticnet.itcp.example;
// By importing a package or a class, all declarations
// are made visible that may be made visible in other packages
// based on their access class.
import java.applet.*;
import java.awt.*;
import java.beans.*;
import de.siemens.simaticnet.itcp.api.*;
import de.siemens.simaticnet.itcp.gui.*;
/**
 * Example2.java
 * <p>Title: Example 2 - Using the ITCP Beans.</p>
 * <p>Description: Inputting a value using the CLTextIn Bean with an S7Variable.</p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Organization: Siemens AG SIMATIC NET</p>
 *
 * Inputting a value using the CLTextIn Bean with an S7Variable.
 * The memory word MW10 is written.
 *
 * Components used:
```

```
 * S7CP
 * S7Device
 * S7Variable
 * CLTextIn
 *
 * @author ITCP Team
 * @version 1.0
 *
 */
public class Example2 extends Applet implements PropertyChangeListener {
   /*-----------------------------------Implemented Interfaces*/
   /*----------------------------Basic Class Applet*/
   // Declaration of the required components
   private CLTextIn cLTextIn1 = null;
   private S7CP s7CP1 = null;
   private S7Device s7Device1 = null;
   private S7Variable s7Variable1 = null;
   /**
    * Is always called when the applet is initialized.
    * This occurs immediately after it is loaded.
    *
    * @see #start
    * @see #stop
    * @see #destroy
    */
   public void init() {
      super.init();
      // Sets the name of the component to the specified character string.
      setName("Example2");
      // Sets the layout manager for this component.
      setLayout(null);
      // Sets the size of the applet width / height
      setSize(426, 240);
      /*-----------Height*/
      /*------Width*/
      // Create an instance for the S7CP bean.
      // S7CP is the Ethernet access point to the station
      s7CP1 = new S7CP();
      // Assign the IP address
      // ######## Project-specific adaptation of the IP address necessary          ########
      s7CP1.setHostString(new HostString ("192.168.1.1:80"));
      /*-------------------------------------------------Specify port number
                                                         normally :80*/
      /*-----------------------------------IP address as string*/
      // Create an instance for the S7Device bean.
      // S7Device is used to address the communication partner in the station.
      s7Device1 = new S7Device();
      // The address is made up of the rack number and slot number of the
      // module. The default of both methods for addressing is '0'.
      // This means that the rack number is unnecessary (.setRack(0)).
      // As we want to communicate with the CPU, the slot of the CPU must
      // be entered.
      // ######## Project-specific adaptation of the rack and slot number  ########
      // ######## necessary                                                ########
      s7Device1.setSlot(2);
      /*----------------Slot number 2 (int)*/
      // Create an instance for the S7Variable bean.
      // The S7Variable bean represents the variable to be read
      // or written.
      s7Variable1 = new S7Variable();
      // The variable is described by an S7 ANY pointer
      s7Variable1.setS7Anypointer(
         new S7Anypointer((int)5, (int)1, (int)131, (int)0, (int)10, (int)0));
      /*-------------------------------------------------------------Bit number 0 ..7*/
```

```
    /*-----------------------------------------------------Memory area offset*/
    /*-------------------------------------------DB number or '0'*/
    /*-----------------------------------Memory area 131 == M*/
    /*--------------------------Repetition factor 1 .. n*/
    /*-------------------Data type 5 == INT*/
    // Sets the name of the component to the specified character string.
    s7Variable1.setVariableName("s7Variable1");
    // Create an instance for the CLTextIn bean.
    // Using the CLTextIn bean, you as user can enter elementary
    // variables.
    cLTextIn1 = new CLTextIn();
    // Sets the name of the component to the specified character string.
    cLTextIn1.setName("CLTextIn1");
    // Specify the start position and size of the component.
    cLTextIn1.setBounds(0, 0, 200, 45);
    /*-----------------------------Component height*/
    /*-------------------------Component width*/
    /*----------------------Start position Y*/
    /*-------------------Start position X*/
    // Specify the size of the input box.
    cLTextIn1.setInFieldSize(100);
    /*-------------------------Length of the input box*/
    // Sets the descriptive text to the specified character string.
    cLTextIn1.setLabel("Value:");
    // Sets the dimension text to the specified character string.
    cLTextIn1.setUnit("");
    // Insert the component in the applet.
    add(cLTextIn1, cLTextIn1.getName());
    // Apart from the definition of the methods to be executed when an event
    // occurs, an object must register with the corresponding
    // event source.
    // This is done by calling the addXXXListener method of the event source,
    // where »XXX« stands for the corresponding event type.
    // The addXXXListener methods all expect a reference to the relevant
    // interface:
    s7CP1.addPropertyChangeListener(this);
    s7Device1.addPropertyChangeListener(this);
    s7Variable1.addPropertyChangeListener(this);
    cLTextIn1.addPropertyChangeListener(this);
}
/**
 * Executes after initialization of an applet.
 * With browsers, start() is then also called when a page containing
 * an applet is reloaded.
 *
 * @see #init
 * @see #stop
 * @see #destroy
 */
public void start() {
    super.start();
}
/**
 * There is a call when the browser or the applet viewer is minimized
 * to an icon or an HTML page containing an applet is exited in
 * a browser.
 *
 * @see #init
 * @see #start
 * @see #destroy
 */
public void stop() {
    super.stop();
}
```

```
   /**
    * Is always called when the applet is destroyed.
    *
    * @see #init
    * @see #start
    * @see #stop
    */
   public void destroy() {
      super.destroy();
      // This method deletes all S7Bean instances and discards all threads.
      // After calling this method, a reinitialization is necessary.
      S7Api.terminate();
   }
   /**
    * Method for handling events for the PropertyChangeListener interface.
    *
    * @param evt PropertyChangeEvent
    */
   public void propertyChange(PropertyChangeEvent evt) {
      // Query whether event was triggered by S7CP.
      if (evt.getSource() == s7CP1)
         // If YES
         // Pass event to the S7Device instance
         s7Device1.propertyChange(evt);
      // Query whether or not event was triggered by S7Device.
      if (evt.getSource() == s7Device1)
         // If YES
         // Pass event to the S7Variable instance
         s7Variable1.propertyChange(evt);
      // Query whether event was triggered by CLTextIn.
      if (evt.getSource() == cLTextIn1)
         // If YES, write values to PLC.
         // Then pass input value to the S7Variable instance.
         s7Variable1.propertyChange(evt);
   }
}
```

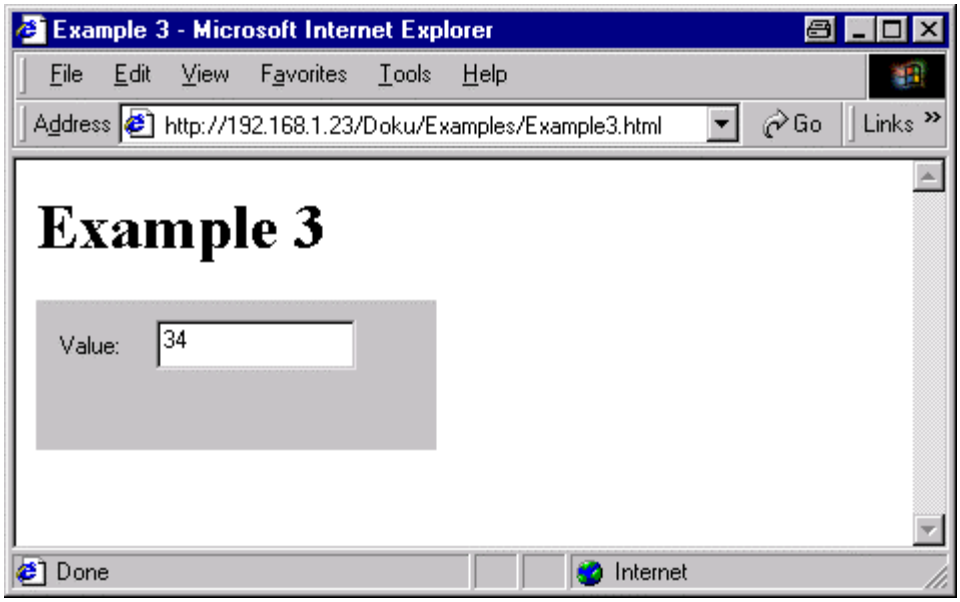
6.1.3 Example 3 – Read and Display a variable from the SIMATIC S7 PLC with AWT components

How the Function Works

You want to read and display a variable from the SIMATIC S7 PLC with AWT components (Abstract Windowing Toolkit).

The memory word MW 10 will be read as INT.

representation



Source Text

```
// This statement assigns all classes whose source text contains
// this statement to a package.
package de.siemens.simaticnet.itcp.example;
// By importing a package or a class, all declarations
// are made visible that may be made visible in other packages
// based on their access class.
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.beans.*;
import de.siemens.simaticnet.itcp.api.*;
/**
 * Example3.java
 * <p>Title: Example 3 - Using the ITCP Beans.</p>
 * <p>Description: Outputting a value using AWT components with an S7Variable.</p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Organization: Siemens AG SIMATIC NET</p>
 *
 * Outputting a value using AWT components with an S7Variable.
```

```
 * The memory word MW10 is read.
 *
 * Components used:
 * S7CP
 * S7Device
 * S7Variable
 * CLTimer
 * AWT Label to describe the input field
 * AWT TextField for the values
 *
 * @author ITCP Team
 * @version 1.0
 *
 */
public class Example3 extends Applet implements PropertyChangeListener, ActionListener {
   /*-----------------------------------Implemented Interfaces*/
   /*----------------------------Basic Class Applet*/
   // Declaration of the required components
   private CLTimer cLTimer1 = null;
   private S7CP s7CP1 = null;
   private S7Device s7Device1 = null;
   private S7Variable s7Variable1 = null;
   private TextField textField1 = null;
   private Label label1 = null;
   /**
    * Is always called when the applet is initialized.
    * This occurs immediately after it is loaded.
    *
    * @see #start
    * @see #stop
    * @see #destroy
    */
   public void init() {
      super.init();
      // Sets the name of the component to the specified character string.
      setName("Example3");
      // Sets the layout manager for this component.
      setLayout(null);
      // Sets the size of the applet width / height
      setSize(426, 240);
      /*-----------Height*/
      /*------Width*/
      // Create an instance for the S7CP bean.
      // S7CP is the Ethernet access point to the station
      s7CP1 = new S7CP();
      // Assign the IP address
      // ######## Project-specific adaptation of the IP address necessary          ########
      s7CP1.setHostString(new HostString ("192.168.1.1:80"));
      /*-------------------------------------------------Specify port number
                                                        normally :80*/
      /*-----------------------------------IP address as string*/
      // Create an instance for the S7Device bean.
      // S7Device is used to address the communication partner in the station.
      s7Device1 = new S7Device();
      // The address is made up of the rack number and slot number of the
      // module. The default of both methods for addressing is '0'.
      // This means that the rack number is unnecessary (.setRack(0)).
      // As we want to communicate with the CPU, the slot of the CPU must
      // be entered.
      // ######## Project-specific adaptation of the rack and slot number  ########
      // ######## necessary                                                ########
      s7Device1.setSlot(2);
      /*----------------Slot number 2 (int)*/
      // Create an instance for the S7Variable bean.
```

```
    // The S7Variable bean represents the variable to be read
    // or written.
    s7Variable1 = new S7Variable();
    // The variable is described by an S7 ANY pointer
    s7Variable1.setS7Anypointer(
       new S7Anypointer((int)5, (int)1, (int)131, (int)0, (int)10, (int)0));
    /*-------------------------------------------------------------Bit number 0 ..7*/
    /*-----------------------------------------------------Memory area offset*/
    /*-------------------------------------------DB number or '0'*/
    /*-----------------------------------Memory area 131 == M*/
    /*--------------------------Repetition factor 1 .. n*/
    /*--------------------Data type 5 == INT*/
    // Sets the name of the component to the specified character string.
    s7Variable1.setVariableName("s7Variable1");
    // Create an instance for the CLTimer bean.
    // The CLTimer bean triggers a PropertyChangeEvent when
    // the time elapses. This event implements a cyclic data
    // update.
    cLTimer1 = new CLTimer();
    // The setDelay() method sets the time interval.
    cLTimer1.setDelay(2000);
    /*----------------Time interval in msec. 2000 == 2 sec.*/
    // Create an instance for a label.
    label1 = new Label();
    // Sets the output text to the specified character string.
    label1.setText("Value:");
    // Sets the name of the component to the specified character string.
    label1.setName("Label1");
    // Specify the start position and size of the component.
    label1.setBounds(10, 10, 50, 25);
    /*---------------------------Component height*/
    /*-----------------------Component width*/
    /*-------------------Start position Y*/
    /*---------------Start position X*/
    // Insert the component in the applet.
    add(label1, label1.getName());
    // Create an instance for a TextField.
    textField1 = new TextField();
    // Sets the output text to the specified character string.
    textField1.setText("");
    // Sets the name of the component to the specified character string.
    textField1.setName("TextField1");
    // Specify the start position and size of the component.
    textField1.setBounds(60, 10, 100, 25);
    /*-------------------------------Component height*/
    /*---------------------------Component width*/
    /*-----------------------Start position Y*/
    /*-------------------Start position X*/
    // Insert the component in the applet.
    add(textField1, textField1.getName());
    // Apart from the definition of the methods to be executed when an event
    // occurs, an object must register with the corresponding
    // event source.
    // This is done by calling the addXXXListener method of the event source,
    // where »XXX« stands for the corresponding event type.
    // The addXXXListener methods all expect a reference to the relevant
    // interface:
    s7CP1.addPropertyChangeListener(this);
    s7Device1.addPropertyChangeListener(this);
    cLTimer1.addActionListener(this);
    s7Variable1.addPropertyChangeListener(this);
}
/**
 * Executes after initialization of an applet.
```

```
 * With browsers, start() is then also called when a page containing
 * an applet is reloaded.
 *
 * @see #init
 * @see #stop
 * @see #destroy
 */
public void start() {
   super.start();
}
/**
 * There is a call when the browser or the applet viewer is minimized
 * to an icon or an HTML page containing an applet is exited in
 * a browser.
 *
 * @see #init
 * @see #start
 * @see #destroy
 */
public void stop() {
   super.stop();
}
/**
 * Is always called when the applet is destroyed.
 *
 * @see #init
 * @see #start
 * @see #stop
 */
public void destroy() {
   super.destroy();
   // This method deletes all S7Bean instances and discards all threads.
   // After calling this method, a reinitialization is necessary.
   S7Api.terminate();
}
/**
 * Method for handling events for the PropertyChangeListener interface.
 *
 * @param evt PropertyChangeEvent
 */
public void propertyChange(PropertyChangeEvent evt) {
   // Query whether event was triggered by S7CP.
   if (evt.getSource() == s7CP1)
      // If YES
      // Pass event to the S7Device instance
      s7Device1.propertyChange(evt);
   // Query whether or not event was triggered by S7Device.
   if (evt.getSource() == s7Device1)
      // If YES
      // Pass event to the S7Variable instance
      s7Variable1.propertyChange(evt);
   // Query whether or not event was triggered by S7Variable.
   if (evt.getSource() == s7Variable1)
      // If YES, the new data from the station is included in the event.
      // The new data is made available as integers by the getNewValue()
      // method.
      // Output of the values in the applet
      textField1.setText(evt.getNewValue().toString());
   /*-----------------------------------Convert integer value to 'String' */
   /*--------------------Read new value*/
   /*---------Display read value in the applet*/
}
/**
 * Method for handling events for the ActionListener interface.
```

```
     *
     * @param e java.awt.event.ActionEvent
     */
    public void actionPerformed(ActionEvent e) {
       // Query whether CLTimer triggered.
       if (e.getSource() == cLTimer1) {
          // If YES, read values from the PLC.
          // Reading is triggered by the processGet() method
          // of S7Variable bean.
          // If the new values exist, the S7Variable bean triggers a
          // PropertyChangeEvent.
          s7Variable1.processGet();
       }
    }
}
```

6.1.4 Example 4 – Writing a Variable to the SIMATIC S7 PLC with AWT Components

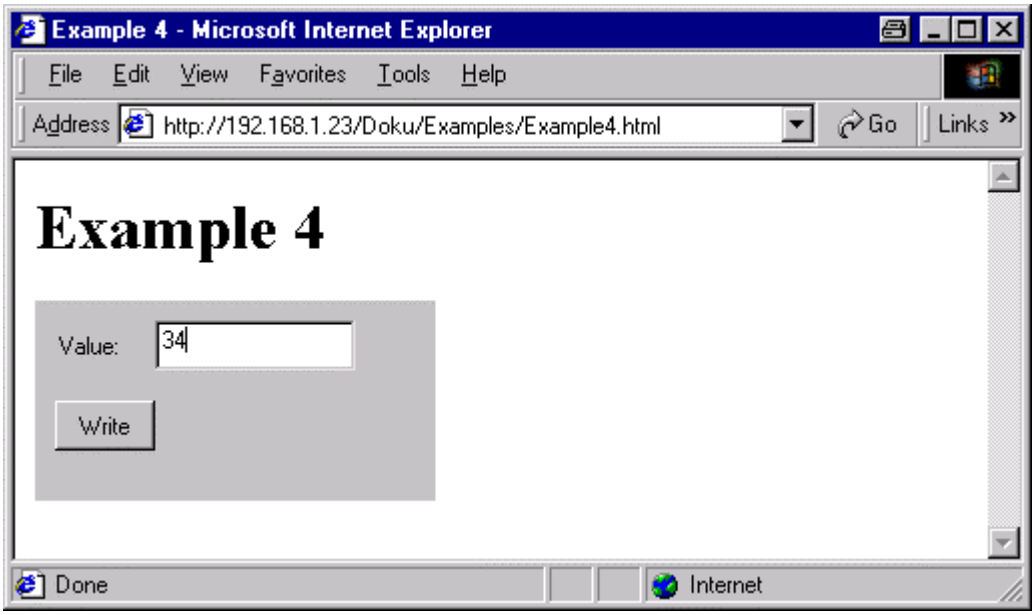
How the Function Works

You want to write a variable to the SIMATIC S7 PLC with AWT components (Abstract Windowing Toolkit).

The memory word MW 10 will be written as INT.

Writing is triggered by clicking a button.

representation



Source Text

```
// This statement assigns all classes whose source text contains
// this statement to a package.
package de.siemens.simaticnet.itcp.example;
// By importing a package or a class, all declarations
// are made visible that may be made visible in other packages
// based on their access class.
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.beans.*;
import de.siemens.simaticnet.itcp.api.*;
/**
 * Example4.java
 * <p>Title: Example 4 - Using the ITCP Beans.</p>
 * <p>Description: Inputting a value using AWT components with an S7Variable.</p>
 * <p>Copyright: Copyright (c) 2003</p>
```

```
 * <p>Organization: Siemens AG SIMATIC NET</p>
 *
 * Inputting a value using AWT components with an S7Variable.
 * The memory word MW10 is written.
 *
 * Components used:
 * S7CP
 * S7Device
 * S7Variable
 * AWT Label to describe the input field
 * AWT TextField for the values
 * AWT Button for writing
 *
 * @author ITCP Team
 * @version 1.0
 *
 */
public class Example4 extends Applet implements PropertyChangeListener, ActionListener {
   /*-----------------------------------Implemented Interfaces*/
   /*----------------------------Basic Class Applet*/
   // Declaration of the required components
   private S7CP s7CP1 = null;
   private S7Device s7Device1 = null;
   private S7Variable s7Variable1 = null;
   private TextField textField1 = null;
   private Label label1 = null;
   private Button button1 = null;
   /**
    * Is always called when the applet is initialized.
    * This occurs immediately after it is loaded.
    *
    * @see #start
    * @see #stop
    * @see #destroy
    */
   public void init() {
      super.init();
      // Sets the name of the component to the specified character string.
      setName("Example4");
      // Sets the layout manager for this component.
      setLayout(null);
      // Sets the size of the applet width / height
      setSize(426, 240);
      /*-----------Height*/
      /*------Width*/
      // Create an instance for the S7CP bean.
      // S7CP is the Ethernet access point to the station
      s7CP1 = new S7CP();
      // Assign the IP address
      // ######## Project-specific adaptation of the IP address necessary          ########
      s7CP1.setHostString(new HostString ("192.168.1.1:80"));
      /*-------------------------------------------------Specify port number
                                                         normally :80*/
      /*-----------------------------------IP address as string*/
      // Create an instance for the S7Device bean.
      // S7Device is used to address the communication partner in the station.
      s7Device1 = new S7Device();
      // The address is made up of the rack number and slot number of the
      // module. The default of both methods for addressing is '0'.
      // This means that the rack number is unnecessary (.setRack(0)).
      // As we want to communicate with the CPU, the slot of the CPU must
      // be entered.
      // ######## Project-specific adaptation of the rack and slot number  ########
      // ######## necessary                                                ########
```

```
s7Device1.setSlot(2);
/*----------------Slot number 2 (int)*/
// Create an instance for the S7Variable bean.
// The S7Variable bean represents the variable to be read
// or written.
s7Variable1 = new S7Variable();
// The variable is described by an S7 ANY pointer
s7Variable1.setS7Anypointer(
   new S7Anypointer((int)5, (int)1, (int)131, (int)0, (int)10, (int)0));
/*-----------------------------------------------------------------Bit number 0 ..7*/
/*-------------------------------------------------------Memory area offset*/
/*-----------------------------------------------DB number or '0'*/
/*-----------------------------------Memory area 131 == M*/
/*---------------------------Repetition factor 1 .. n*/
/*-------------------Data type 5 == INT*/
// Sets the name of the component to the specified character string.
s7Variable1.setVariableName("s7Variable1");
// Create an instance for a label.
label1 = new Label();
// Sets the output text to the specified character string.
label1.setText("Value:");
// Sets the name of the component to the specified character string.
label1.setName("Label1");
// Specify the start position and size of the component.
label1.setBounds(10, 10, 50, 25);
/*---------------------------Component height*/
/*-----------------------Component width*/
/*-------------------Start position Y*/
/*---------------Start position X*/
// Insert the component in the applet.
add(label1, label1.getName());
// Create an instance for a TextField.
textField1 = new TextField();
// Sets the output text to the specified character string.
textField1.setText("");
// Sets the name of the component to the specified character string.
textField1.setName("TextField1");
// Specify the start position and size of the component.
textField1.setBounds(60, 10, 100, 25);
/*-------------------------------Component height*/
/*---------------------------Component width*/
/*-----------------------Start position Y*/
/*-------------------Start position X*/
// Insert the component in the applet.
add(textField1, textField1.getName());
// Create an instance for a button.
button1 = new Button();
// Sets the output text to the specified character string.
button1.setLabel("Write");
// Sets the name of the component to the specified character string.
button1.setName("Button1");
// Specify the start position and size of the component.
button1.setBounds(10, 50, 50, 25);
/*---------------------------Component height*/
/*-----------------------Component width*/
/*-------------------Start position Y*/
/*---------------Start position X*/
// Insert the component in the applet.
add(button1, button1.getName());
// Apart from the definition of the methods to be executed when an event
// occurs, an object must register with the corresponding
// event source.
// This is done by calling the addXXXListener method of the event source,
// where »XXX« stands for the corresponding event type.
```

```
      // The addXXXListener methods all expect a reference to the relevant
      // interface:
      s7CP1.addPropertyChangeListener(this);
      s7Device1.addPropertyChangeListener(this);
      s7Variable1.addPropertyChangeListener(this);
      button1.addActionListener(this);
      textField1.addActionListener(this);
   }
   /**
    * Executes after initialization of an applet.
    * With browsers, start() is then also called when a page containing
    * an applet is reloaded.
    *
    * @see #init
    * @see #stop
    * @see #destroy
    */
   public void start() {
      super.start();
   }
   /**
    * There is a call when the browser or the applet viewer is minimized
    * to an icon or an HTML page containing an applet is exited in
    * a browser.
    *
    * @see #init
    * @see #start
    * @see #destroy
    */
   public void stop() {
      super.stop();
   }
   /**
    * Is always called when the applet is destroyed.
    *
    * @see #init
    * @see #start
    * @see #stop
    */
   public void destroy() {
      super.destroy();
      // This method deletes all S7Bean instances and discards all threads.
      // After calling this method, a reinitialization is necessary.
      S7Api.terminate();
   }
   /**
    * Method for handling events for the PropertyChangeListener interface.
    *
    * @param evt PropertyChangeEvent
    */
   public void propertyChange(PropertyChangeEvent evt) {
      // Query whether event was triggered by S7CP.
      if (evt.getSource() == s7CP1)
         // If YES
         // Pass event to the S7Device instance
         s7Device1.propertyChange(evt);
      // Query whether or not event was triggered by S7Device.
      if (evt.getSource() == s7Device1)
         // If YES
         // Pass event to the S7Variable instance
         s7Variable1.propertyChange(evt);
      // Query whether or not event was triggered by S7Variable.
      if (evt.getSource() == s7Variable1) {
         // If YES, the new data from the station is included in the event.
```

```
            // The new data is made available as integers by the getNewValue()
            // method.
            // Output of the values in the applet
            textField1.setText(evt.getNewValue().toString());
            /*-----------------------------------Convert integer value to 'String' */
            /*--------------------Read new value*/
            /*---------Display read value in the applet*/
         }
      }
      /**
       * Method for handling events for the ActionListener interface.
       *
       * @param e java.awt.event.ActionEvent
       */
      public void actionPerformed(ActionEvent e) {
         // Query whether or not write button was triggered.
         if (e.getSource() == button1) {
            // If YES, write values to PLC.
            // The new data is transferred to the setValue() method as integer.
            // The setValue() method then writes the value to the station.
            // Calling setValue() automatically retriggers the processGet() method to
            // read the area.
            s7Variable1.setValue(textField1.getText());
            /*------------------------------Read input value from TextField*/
         }
         // Query whether or not Return was triggered in the TextField.
         if (e.getSource() == textField1) {
            // If YES, write values to PLC.
            // The new data is transferred to the setValue() method as integer.
            // The setValue() method then writes the value to the station.
            // Calling setValue() automatically retriggers the processGet() method to
            // read the area.
            s7Variable1.setValue(textField1.getText());
            /*------------------------------Read input value from TextField*/
         }
      }
}
```

6.1.5 Example 5 – A button with the ”Pushbutton” Function

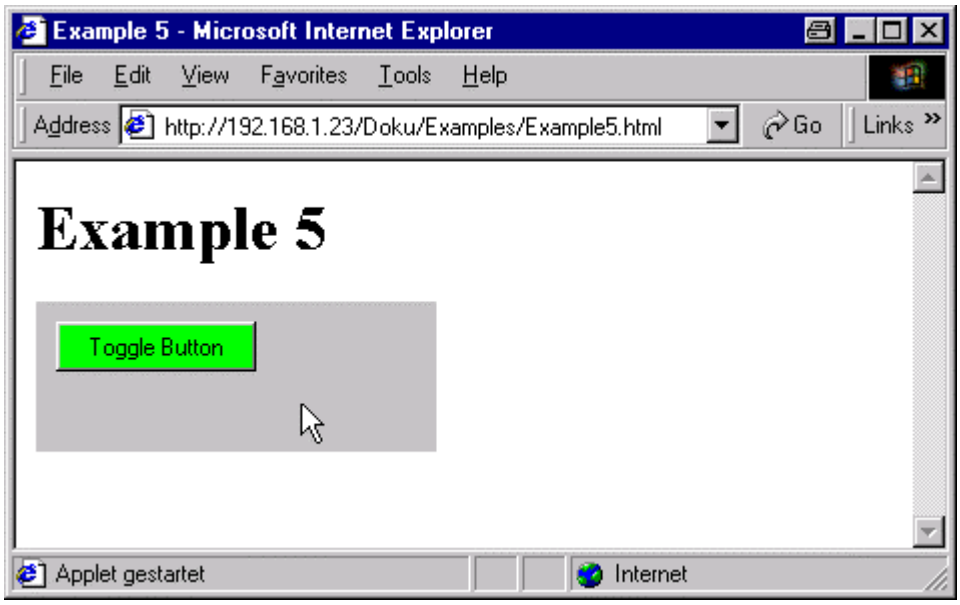
How the Function Works

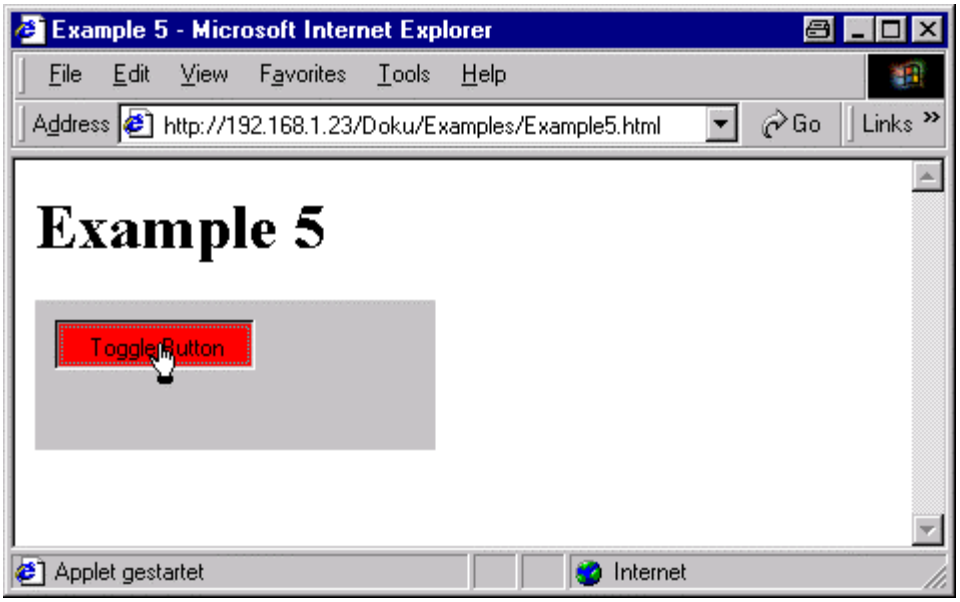
You want to set up a button with a ”pushbutton” function.

M 10.0 is written as BOOL.

The value ”true” is passed to the memory bit when the button is pressed. The value ”false” is passed to the memory bit when the button is released.

representation





Source Text

```
// This statement assigns all classes whose source text contains
// this statement to a package.
package de.siemens.simaticnet.itcp.example;
// By importing a package or a class, all declarations
// are made visible that may be made visible in other packages
// based on their access class.
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.beans.*;
import de.siemens.simaticnet.itcp.api.*;
/**
 * Example5.java
 * <p>Title: Example 5 - Using the ITCP Beans.</p>
 * <p>Description: A button with a "pushbutton" function.</p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Organization: Siemens AG SIMATIC NET</p>
 *
 * A button with the "pushbutton" function
 * Memory bit M10.0 is read and written.
 *
 * Components used:
 * S7CP
 * S7Device
 * S7Variable
 * AWT Button
 *
 * @author ITCP Team
 * @version 1.0
 *
 */
public class Example5 extends Applet implements PropertyChangeListener, MouseListener {
   /*---------------------------------Implemented Interfaces*/
   /*---------------------------Basic Class Applet*/
   // Declaration of the required components
   private S7CP s7CP1 = null;
```

```
private S7Device s7Device1 = null;
private S7Variable s7Variable1 = null;
private Button button1 = null;
/**
 * Is always called when the applet is initialized.
 * This occurs immediately after it is loaded.
 *
 * @see #start
 * @see #stop
 * @see #destroy
 */
public void init() {
   super.init();
   // Sets the name of the component to the specified character string.
   setName("Example5");
   // Sets the layout manager for this component.
   setLayout(null);
   // Sets the size of the applet width / height
   setSize(426, 240);
   /*-----------Height*/
   /*------Width*/
   // Create an instance for the S7CP bean.
   // S7CP is the Ethernet access point to the station
   s7CP1 = new S7CP();
   // Assign the IP address
   // ######## Project-specific adaptation of the IP address necessary          ########
   s7CP1.setHostString(new HostString ("192.168.1.1:80"));
   /*-------------------------------------------------Specify port number
                                                      normally :80*/
   /*----------------------------------IP address as string*/
   // Create an instance for the S7Device bean.
   // S7Device is used to address the communication partner in the station.
   s7Device1 = new S7Device();
   // The address is made up of the rack number and slot number of the
   // module. The default of both methods for addressing is '0'.
   // This means that the rack number is unnecessary (.setRack(0)).
   // As we want to communicate with the CPU, the slot of the CPU must
   // be entered.
   // ######## Project-specific adaptation of the rack and slot number  ########
   // ######## necessary                                                ########
   s7Device1.setSlot(2);
   /*----------------Slot number 2 (int)*/
   // Create an instance for the S7Variable bean.
   // The S7Variable bean represents the variable to be read
   // or written.
   s7Variable1 = new S7Variable();
   // The variable is described by an S7 ANY pointer
   s7Variable1.setS7Anypointer(
      new S7Anypointer((int)1, (int)1, (int)131, (int)0, (int)10, (int)0));
   /*-----------------------------------------------------------------Bit number 0 ..7*/
   /*-------------------------------------------------------Memory area offset*/
   /*-----------------------------------------------DB number or '0'*/
   /*----------------------------------Memory area 131 == M*/
   /*---------------------------Repetition factor 1 .. n*/
   /*-------------------Data type 1 == BOOL*/
   // Sets the name of the component to the specified character string.
   s7Variable1.setVariableName("s7Variable1");
   // Create an instance for a button.
   button1 = new Button();
   // Sets the output text to the specified character string.
   button1.setLabel("Toggle Button");
   // Sets the name of the component to the specified character string.
   button1.setName("ToggleButton");
   // Specify the start position and size of the component.
```

```
        button1.setBounds(10, 10, 100, 25);
        /*-----------------------------Component height*/
        /*-------------------------Component width*/
        /*--------------------Start position Y*/
        /*----------------Start position X*/
        // Sets the background color to green.
        button1.setBackground(Color.green);
        // Insert the component in the applet.
        add(button1, button1.getName());
        // Apart from the definition of the methods to be executed when an event
        // occurs, an object must register with the corresponding
        // event source.
        // This is done by calling the addXXXListener method of the event source,
        // where »XXX« stands for the corresponding event type.
        // The addXXXListener methods all expect a reference to the relevant
        // interface:
        s7CP1.addPropertyChangeListener(this);
        s7Device1.addPropertyChangeListener(this);
        s7Variable1.addPropertyChangeListener(this);
        button1.addMouseListener(this);
    }
    /**
     * Executes after initialization of an applet.
     * With browsers, start() is then also called when a page containing
     * an applet is reloaded.
     *
     * @see #init
     * @see #stop
     * @see #destroy
     */
    public void start() {
        super.start();
    }
    /**
     * There is a call when the browser or the applet viewer is minimized
     * to an icon or an HTML page containing an applet is exited in
     * a browser.
     *
     * @see #init
     * @see #start
     * @see #destroy
     */
    public void stop() {
        super.stop();
    }
    /**
     * Is always called when the applet is destroyed.
     *
     * @see #init
     * @see #start
     * @see #stop
     */
    public void destroy() {
        super.destroy();
        // This method deletes all S7Bean instances and discards all threads.
        // After calling this method, a reinitialization is necessary.
        S7Api.terminate();
    }
    /**
     * Method for handling events for the PropertyChangeListener interface.
     *
     * @param evt PropertyChangeEvent
     */
    public void propertyChange(PropertyChangeEvent evt) {
```

```
    // Query whether event was triggered by S7CP.
    if (evt.getSource() == s7CP1)
       // If YES
       // Pass event to the S7Device instance
       s7Device1.propertyChange(evt);
    // Query whether or not event was triggered by S7Device.
    if (evt.getSource() == s7Device1)
       // If YES
       // Pass event to the S7Variable instance
       s7Variable1.propertyChange(evt);
    // Query whether or not event was triggered by S7Variable.
    if (evt.getSource() == s7Variable1) {
       // In this example, there is no evaluation of the station values
       // read!
    }
}
/**
 * Method for handling events for the MouseListener interface.
 *
 * mousePressed is called when a mouse button is pressed.
 *
 * @param e java.awt.event.ActionEvent
 */
public void mousePressed(MouseEvent e) {
    // Query whether or not button was triggered.
    if (e.getSource() == button1) {
       // If YES, write new value (true) to the PLC.
       // The new data is transferred to the setValue() method as string.
       // The setValue() method then writes the value to the station.
       // Calling setValue() automatically retriggers the processGet() method to
       // read the area.
       s7Variable1.setValue(String.valueOf("true"));
       // Sets the background color to red.
       button1.setBackground(Color.red);
       // The waitOnNewData method only allows setValue() to be called again
       // only after a wait time has elapsed or after the arrival of new data.
       // This prevents fast button clicks.
       s7Variable1.waitOnNewData(2000);
       /*------------------------Wait time in msec. 2000 == 2 sec.*/
    }
}
/**
 * Method for handling events for the MouseListener interface.
 *
 * mouseReleased is called when a mouse button is released.
 *
 * @param e java.awt.event.ActionEvent
 */
public void mouseReleased(MouseEvent e) {
    // Query whether or not button was triggered.
    if (e.getSource() == button1) {
       // If YES, write new value (false) to the PLC.
       // The new data is transferred to the setValue() method as string.
       // The setValue() method then writes the value to the station.
       // Calling setValue() automatically retriggers the processGet() method to
       // read the area.
       s7Variable1.setValue(String.valueOf("false"));
       // Sets the background color to green.
       button1.setBackground(Color.green);
       // The waitOnNewData method only allows setValue() to be called again
       // only after a wait time has elapsed or after the arrival of new data.
       // This prevents fast button clicks.
       s7Variable1.waitOnNewData(2000);
       /*------------------------Wait time in msec. 2000 == 2 sec.*/
```

```
        }
    }
    // The MouseListener interface has more than one method.
    // If you require only one of these methods to process the event,
    // the methods you do not require must be made available as dummy
    // implementations.
    public void mouseClicked(MouseEvent e) {
    }
    public void mouseEntered(MouseEvent e) {
    }
    public void mouseExited(MouseEvent e) {
    }
}
```

6.1.6 Example 6 – A button with the ”Switch” Function

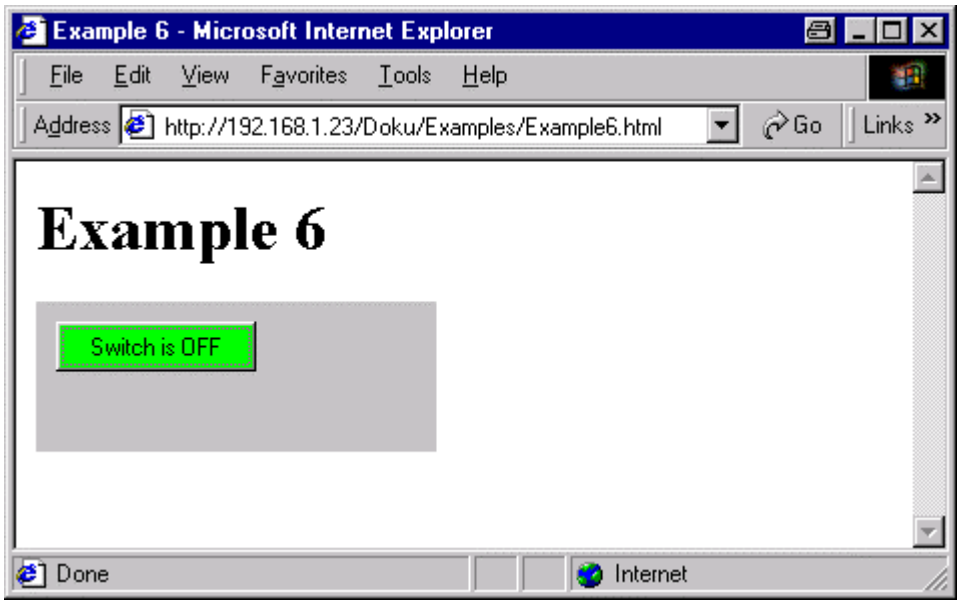
How the Function Works

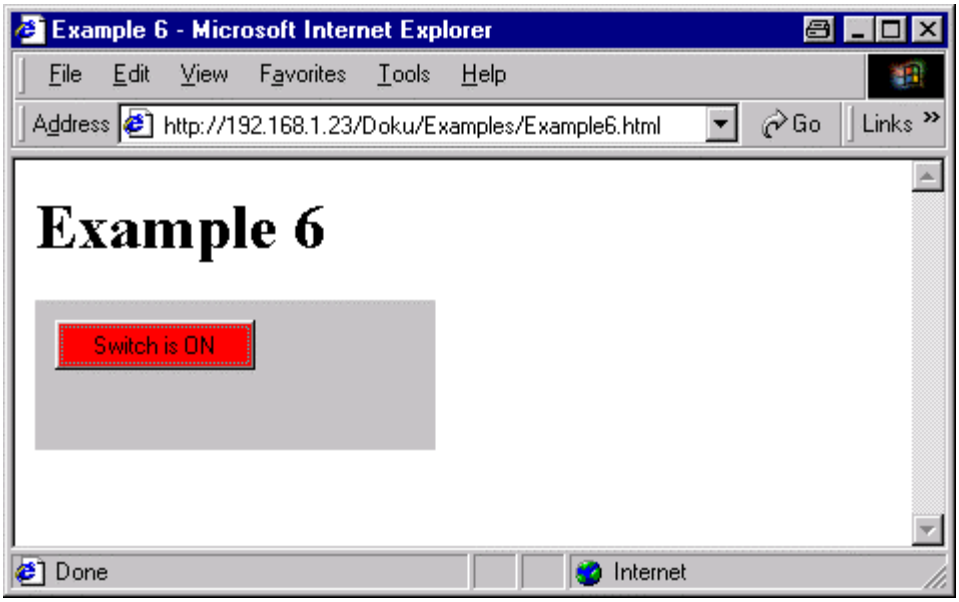
You want to set up a button with a ”switch” function.

M 10.0 is written as BOOL.

When the applet starts, the status of the memory bit is read. If the button is pressed, the status inverts and is written to the memory bit.

representation





Source Text

```
// This statement assigns all classes whose source text contains
// this statement to a package.
package de.siemens.simaticnet.itcp.example;
// By importing a package or a class, all declarations
// are made visible that may be made visible in other packages
// based on their access class.
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.beans.*;
import de.siemens.simaticnet.itcp.api.*;
/**
 * Example6.java
 * <p>Title: Example 6 - Using the ITCP Beans.</p>
 * <p>Description: A button with a "switch" function.</p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Organization: Siemens AG SIMATIC NET</p>
 *
 * A button with the "switch" function
 * Memory bit M10.0 is read and written.
 *
 * Components used:
 * S7CP
 * S7Device
 * S7Variable
 * AWT Button
 *
 * @author ITCP Team
 * @version 1.0
 *
 */
public class Example6 extends Applet implements PropertyChangeListener, MouseListener {
   /*---------------------------------Implemented Interfaces*/
   /*---------------------------Basic Class Applet*/
   // Declaration of the required components
   private S7CP s7CP1 = null;
```

```
private S7Device s7Device1 = null;
private S7Variable s7Variable1 = null;
private Button button1 = null;
/**
 * Is always called when the applet is initialized.
 * This occurs immediately after it is loaded.
 *
 * @see #start
 * @see #stop
 * @see #destroy
 */
public void init() {
   super.init();
   // Sets the name of the component to the specified character string.
   setName("Example6");
   // Sets the layout manager for this component.
   setLayout(null);
   // Sets the size of the applet width / height
   setSize(426, 240);
   /*-----------Height*/
   /*------Width*/
   // Create an instance for the S7CP bean.
   // S7CP is the Ethernet access point to the station
   s7CP1 = new S7CP();
   // Assign the IP address
   // ######## Project-specific adaptation of the IP address necessary          ########
   s7CP1.setHostString(new HostString ("192.168.1.1:80"));
   /*-------------------------------------------------Specify port number
                                                      normally :80*/
   /*----------------------------------IP address as string*/
   // Create an instance for the S7Device bean.
   // S7Device is used to address the communication partner in the station.
   s7Device1 = new S7Device();
   // The address is made up of the rack number and slot number of the
   // module. The default of both methods for addressing is '0'.
   // This means that the rack number is unnecessary (.setRack(0)).
   // As we want to communicate with the CPU, the slot of the CPU must
   // be entered.
   // ######## Project-specific adaptation of the rack and slot number  ########
   // ######## necessary                                                ########
   s7Device1.setSlot(2);
   /*----------------Slot number 2 (int)*/
   // Create an instance for the S7Variable bean.
   // The S7Variable bean represents the variable to be read
   // or written.
   s7Variable1 = new S7Variable();
   // The variable is described by an S7 ANY pointer
   s7Variable1.setS7Anypointer(
      new S7Anypointer((int)1, (int)1, (int)131, (int)0, (int)10, (int)0));
   /*-----------------------------------------------------------------Bit number 0 ..7*/
   /*-------------------------------------------------------Memory area offset*/
   /*-----------------------------------------------DB number or '0'*/
   /*----------------------------------Memory area 131 == M*/
   /*---------------------------Repetition factor 1 .. n*/
   /*-------------------Data type 1 == BOOL*/
   // Sets the name of the component to the specified character string.
   s7Variable1.setVariableName("s7Variable1");
   // Create an instance for a button.
   button1 = new Button();
   // Sets the output text to the specified character string.
   button1.setLabel("Switch");
   // Sets the name of the component to the specified character string.
   button1.setName("Switch");
   // Specify the start position and size of the component.
```

```
    button1.setBounds(10, 10, 100, 25);
    /*-----------------------------Component height*/
    /*-------------------------Component width*/
    /*--------------------Start position Y*/
    /*----------------Start position X*/
    // Insert the component in the applet.
    add(button1, button1.getName());
    // Apart from the definition of the methods to be executed when an event
    // occurs, an object must register with the corresponding
    // event source.
    // This is done by calling the addXXXListener method of the event source,
    // where »XXX« stands for the corresponding event type.
    // The addXXXListener methods all expect a reference to the relevant
    // interface:
    s7CP1.addPropertyChangeListener(this);
    s7Device1.addPropertyChangeListener(this);
    s7Variable1.addPropertyChangeListener(this);
    button1.addMouseListener(this);
}
/**
 * Executes after initialization of an applet.
 * With browsers, start() is then also called when a page containing
 * an applet is reloaded.
 *
 * @see #init
 * @see #stop
 * @see #destroy
 */
public void start() {
    super.start();
    // Read the defined S7Variable to display the current status.
    // Reading is triggered by the processGet() method
    // of S7Variable bean.
    // If the new values exist, the S7Variable bean triggers a
    // PropertyChangeEvent.
    s7Variable1.processGet();
}
/**
 * There is a call when the browser or the applet viewer is minimized
 * to an icon or an HTML page containing an applet is exited in
 * a browser.
 *
 * @see #init
 * @see #start
 * @see #destroy
 */
public void stop() {
    super.stop();
}
/**
 * Is always called when the applet is destroyed.
 *
 * @see #init
 * @see #start
 * @see #stop
 */
public void destroy() {
    super.destroy();
    // This method deletes all S7Bean instances and discards all threads.
    // After calling this method, a reinitialization is necessary.
    S7Api.terminate();
}
/**
 * Method for handling events for the PropertyChangeListener interface.
```

```
 *
 * @param evt PropertyChangeEvent
 */
public void propertyChange(PropertyChangeEvent evt) {
   // Query whether event was triggered by S7CP.
   if (evt.getSource() == s7CP1)
      // If YES
      // Pass event to the S7Device instance
      s7Device1.propertyChange(evt);
   // Query whether or not event was triggered by S7Device.
   if (evt.getSource() == s7Device1)
      // If YES
      // Pass event to the S7Variable instance
      s7Variable1.propertyChange(evt);
   // Query whether or not event was triggered by S7Variable.
   if (evt.getSource() == s7Variable1) {
      // If YES, the new data from the station is included in the event.
      // The new data is made available as Boolean by the getNewValue()
      // method.
      // Query the signal state true or false (RLO1 / RLO0) of the
      // value read.
      if (((Boolean)evt.getNewValue()).booleanValue()) {
         /*------------------------------Convert Boolean value to 'boolean' */
         /*---------------Read new value*/
         /*---Converts the new value to Boolean*/
         // If true (RLO1), then set text and color of the button.
         // Sets the output text to the specified character string.
         button1.setLabel("Switch is ON");
         // Sets the background color to red.
         button1.setBackground(Color.red);
      } else {
         // If false (RLO0), then set text and color of the button.
         // Sets the output text to the specified character string.
         button1.setLabel("Switch is OFF");
         // Sets the background color to green.
         button1.setBackground(Color.green);
      }
   }
}
/**
 * Method for handling events for the MouseListener interface.
 *
 * mousePressed is called when a mouse button is pressed.
 *
 * @param e java.awt.event.ActionEvent
 */
public void mousePressed(MouseEvent e) {
   // Query whether or not button was triggered.
   if (e.getSource() == button1) {
      // If YES, write new value to the PLC.
      // Query the last status of the station value and write back
      // inverted.
      if (((Boolean)s7Variable1.getValue()).booleanValue()) {
         /*-----------------------------------Convert Boolean value to 'boolean' */
         /*----------------------Read last status*/
         /*---Converts the last status to Boolean*/
         // If the last status was true (RLO1), then new value false (RLO0).
         // The new data is transferred to the setValue() method as string.
         // The setValue() method then writes the value to the station.
         // Calling setValue() automatically retriggers the processGet() method to
         // read the area.
         s7Variable1.setValue(String.valueOf(false));
         // The waitOnNewData method only allows setValue() to be called again
         // only after a wait time has elapsed or after the arrival of new data.
```

```
            // This prevents fast button clicks.
            s7Variable1.waitOnNewData(2000);
            /*------------------------Wait time in msec. 2000 == 2 sec.*/
         } else {
            // If the last status was false (RLO0), then new value true (RLO1).
            s7Variable1.setValue(String.valueOf(true));
            s7Variable1.waitOnNewData(2000);
         }
      }
   }
   // The MouseListener interface has more than one method.
   // If you require only one of these methods to process the event,
   // the methods you do not require must be made available as dummy
   // implementations.
   public void mouseReleased(MouseEvent e) {
   }
   public void mouseClicked(MouseEvent e) {
   }
   public void mouseEntered(MouseEvent e) {
   }
   public void mouseExited(MouseEvent e) {
   }
}
```

6.1.7 Example 7 – Outputting Several Values Using one Variable

How the Function Works

You want to output several values using one variable.

32 bytes of data are read from DB4 starting at byte 0.

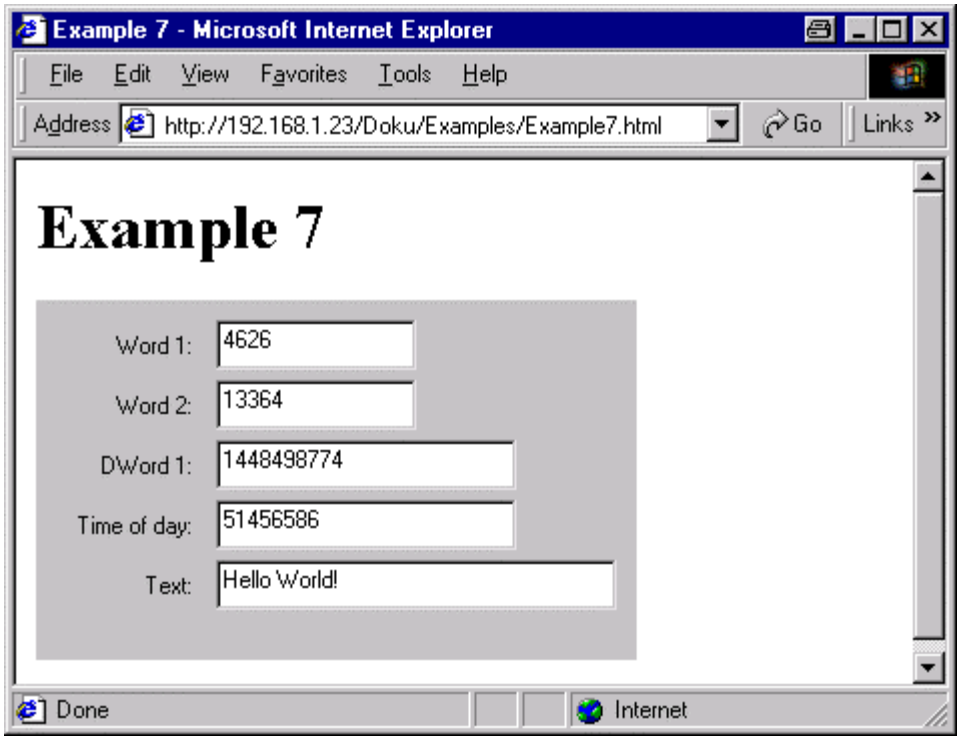
Output is cyclic.

Structure of the DB

DB4 -- ITCP400\SIMATIC 400\CPU 416-2 DP

Address	Name	Type	Initial value	Actual value	Comment
0.0	Word1	WORD	W#16#1212	W#16#1212	
2.0	Word2	WORD	W#16#3434	W#16#3434	
4.0	DWord1	DWORD	DW#16#56565656	DW#16#56565656	
8.0	TimeOfDay	TIME_OF_DAY	TOD#14:17:36.586	TOD#14:17:36.586	
12.0	Text	STRING [12]	'Hello World!'	'Hello World!'	

representation



Source Text

```
// This statement assigns all classes whose source text contains
// this statement to a package.
package de.siemens.simaticnet.itcp.example;
// By importing a package or a class, all declarations
// are made visible that may be made visible in other packages
// based on their access class.
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.beans.*;
import de.siemens.simaticnet.itcp.api.*;
/**
 * Example7.java
 * <p>Title: Example 7 - Using the ITCP Beans.</p>
 * <p>Description: Outputting several values using AWT components with an
 * S7Variable.</p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Organization: Siemens AG SIMATIC NET</p>
 *
 * Outputting several values using one variable.
 * 32 bytes of data are read from DB4 starting at byte 0.
 *
 * Components used:
 * S7CP
 * S7Device
 * S7Variable
 * CLTimer
 * AWT Label to describe the output fields
 * AWT TextField for each of the 5 values
 *
 * Structure of the data block DB4 as STL source for STEP7
 * DATA_BLOCK DB 4
 * STRUCT
 * Word1 : WORD  := W#16#1212;
 * Word2 : WORD  := W#16#3434;
 * DWord1 : DWORD  := DW#16#56565656;
 * TimeOfDay : TIME_OF_DAY  := TOD#14:17:36.586;
 * Text : STRING  [12 ] := 'Hello World!';
 * END_STRUCT ;
 * BEGIN
 * END_DATA_BLOCK
 *
 *
 * @author ITCP Team
 * @version 1.0
 *
 */
public class Example7 extends Applet implements PropertyChangeListener, ActionListener {
   /*-----------------------------------Implemented Interfaces*/
   /*----------------------------Basic Class Applet*/
   // Declaration of the required components
   private CLTimer cLTimer1 = null;
   private S7CP s7CP1 = null;
   private S7Device s7Device1 = null;
   private S7Variable s7Variable1 = null;
   private TextField tFWord1 = null;
   private Label lWord1 = null;
   private TextField tFWord2 = null;
   private Label lWord2 = null;
   private TextField tFDWord1 = null;
   private Label lDWord1 = null;
   private TextField tFTimeOfDay1 = null;
```

```
private Label lTimeOfDay1 = null;
private TextField tFText1 = null;
private Label lText1 = null;
/**
 * Is always called when the applet is initialized.
 * This occurs immediately after it is loaded.
 *
 * @see #start
 * @see #stop
 * @see #destroy
 */
public void init() {
   super.init();
   // Sets the name of the component to the specified character string.
   setName("Example7");
   // Sets the layout manager for this component.
   setLayout(null);
   // Sets the size of the applet width / height
   setSize(426, 240);
   /*-----------Height*/
   /*------Width*/
   // Create an instance for the S7CP bean.
   // S7CP is the Ethernet access point to the station
   s7CP1 = new S7CP();
   // Assign the IP address
   // ######## Project-specific adaptation of the IP address necessary          ########
   s7CP1.setHostString(new HostString ("192.168.1.1:80"));
   /*-------------------------------------------------Specify port number
                                                      normally :80*/
   /*----------------------------------IP address as string*/
   // Create an instance for the S7Device bean.
   // S7Device is used to address the communication partner in the station.
   s7Device1 = new S7Device();
   // The address is made up of the rack number and slot number of the
   // module. The default of both methods for addressing is '0'.
   // This means that the rack number is unnecessary (.setRack(0)).
   // As we want to communicate with the CPU, the slot of the CPU must
   // be entered.
   // ######## Project-specific adaptation of the rack and slot number  ########
   // ######## necessary                                               ########
   s7Device1.setSlot(2);
   /*----------------Slot number 2 (int)*/
   // Create an instance for the S7Variable bean.
   // The S7Variable bean represents the variable to be read
   // or written.
   s7Variable1 = new S7Variable();
   // The variable is described by an S7 ANY pointer
   s7Variable1.setS7Anypointer(
      new S7Anypointer((int)2, (int)26, (int)132, (int)4, (int)0, (int)0));
   /*-------------------------------------------------------------Bit number 0 ..7*/
   /*-----------------------------------------------------Memory area offset*/
   /*-------------------------------------------DB number or '0'*/
   /*----------------------------------Memory area 132 == DB*/
   /*--------------------------Repetition factor 1 .. n*/
   /*------------------Data type 2 == Byte*/
   // Sets the name of the component to the specified character string.
   s7Variable1.setVariableName("s7Variable1");
   // Create an instance for the CLTimer bean.
   // The CLTimer bean triggers a PropertyChangeEvent when
   // the time elapses. This event implements a cyclic data update.
   cLTimer1 = new CLTimer();
   // The setDelay() method sets the time interval.
   cLTimer1.setDelay(2000);
   /*----------------Time interval in msec. 2000 == 2 sec.*/
```

```
// Create an instance for a label.
lWord1 = new Label();
// Sets the output text to the specified character string.
lWord1.setText("Word 1:");
// Specify the alignment of the display text.
lWord1.setAlignment(Label.RIGHT);
// Sets the name of the component to the specified character string.
lWord1.setName("Text1");
// Specify the start position and size of the component.
lWord1.setBounds(10, 10, 70, 25);
/*---------------------------Component height*/
/*-----------------------Component width*/
/*-------------------Start position Y*/
/*---------------Start position X*/
// Insert the component in the applet.
add(lWord1, lWord1.getName());
// Create an instance for a TextField.
tFWord1 = new TextField();
// Sets the output text to the specified character string.
tFWord1.setText("");
// Sets the name of the component to the specified character string.
tFWord1.setName("TextField1");
// Specify the start position and size of the component.
tFWord1.setBounds(90, 10, 100, 25);
/*-----------------------------Component height*/
/*-------------------------Component width*/
/*--------------------Start position Y*/
/*----------------Start position X*/
// Insert the component in the applet.
add(tFWord1, tFWord1.getName());
lWord2 = new Label();
lWord2.setText("Word 2:");
lWord2.setAlignment(Label.RIGHT);
lWord2.setName("Text2");
lWord2.setBounds(10, 40, 70, 25);
add(lWord2, lWord2.getName());
tFWord2 = new TextField();
tFWord2.setText("");
tFWord2.setName("TextField2");
tFWord2.setBounds(90, 40, 100, 25);
add(tFWord2, tFWord2.getName());
lDWord1 = new Label();
lDWord1.setText("DWord 1:");
lDWord1.setAlignment(Label.RIGHT);
lDWord1.setName("Text3");
lDWord1.setBounds(10, 70, 70, 25);
add(lDWord1, lDWord1.getName());
tFDWord1 = new TextField();
tFDWord1.setText("");
tFDWord1.setName("TextField3");
tFDWord1.setBounds(90, 70, 150, 25);
add(tFDWord1, tFDWord1.getName());
lTimeOfDay1 = new Label();
lTimeOfDay1.setText("Time of day:");
lTimeOfDay1.setAlignment(Label.RIGHT);
lTimeOfDay1.setName("Text4");
lTimeOfDay1.setBounds(10, 100, 70, 25);
add(lTimeOfDay1, lTimeOfDay1.getName());
tFTimeOfDay1 = new TextField();
tFTimeOfDay1.setText("");
tFTimeOfDay1.setName("TextField4");
tFTimeOfDay1.setBounds(90, 100, 150, 25);
add(tFTimeOfDay1, tFTimeOfDay1.getName());
lText1 = new Label();
```

```
    lText1.setText("Text:");
    lText1.setAlignment(Label.RIGHT);
    lText1.setName("Text5");
    lText1.setBounds(10, 130, 70, 25);
    add(lText1, lText1.getName());
    tFText1 = new TextField();
    tFText1.setText("");
    tFText1.setName("TextField5");
    tFText1.setBounds(90, 130, 200, 25);
    add(tFText1, tFText1.getName());
    // Apart from the definition of the methods to be executed when an event
    // occurs, an object must register with the corresponding
    // event source.
    // This is done by calling the addXXXListener method of the event source,
    // where »XXX« stands for the corresponding event type.
    // The addXXXListener methods all expect a reference to the relevant
    // interface:
    s7CP1.addPropertyChangeListener(this);
    s7Device1.addPropertyChangeListener(this);
    cLTimer1.addActionListener(this);
    s7Variable1.addPropertyChangeListener(this);
}
/**
 * Executes after initialization of an applet.
 * With browsers, start() is then also called when a page containing
 * an applet is reloaded.
 *
 * @see #init
 * @see #stop
 * @see #destroy
 */
public void start() {
    super.start();
}
/**
 * There is a call when the browser or the applet viewer is minimized
 * to an icon or an HTML page containing an applet is exited in
 * a browser.
 *
 * @see #init
 * @see #start
 * @see #destroy
 */
public void stop() {
    super.stop();
}
/**
 * Is always called when the applet is destroyed.
 *
 * @see #init
 * @see #start
 * @see #stop
 */
public void destroy() {
    super.destroy();
    // This method deletes all S7Bean instances and discards all threads.
    // After calling this method, a reinitialization is necessary.
    S7Api.terminate();
}
/**
 * Method for handling events for the PropertyChangeListener interface.
 *
 * @param evt PropertyChangeEvent
 */
```

```
public void propertyChange(PropertyChangeEvent evt) {
   // Query whether event was triggered by S7CP.
   if (evt.getSource() == s7CP1)
      // If YES
      // Pass event to the S7Device instance
      s7Device1.propertyChange(evt);
   // Query whether or not event was triggered by S7Device.
   if (evt.getSource() == s7Device1)
      // If YES
      // Pass event to the S7Variable instance
      s7Variable1.propertyChange(evt);
   // Query whether or not event was triggered by S7Variable.
   if (evt.getSource() == s7Variable1) {
      // If YES, the new data from the station is included in the event.
      // The new data is made available in an int[] (IntegerArray) by the
      // getNewValue() method.
      // Reload the data into a local auxiliary variable (myArray)
      int[] myArray = (int[])evt.getNewValue();
      // Declaration of local auxiliary variables for reloading individual
      // output values.
      int word1, word2;
      long dword1, tod;
      char[] text = new char[myArray[13]];
      /*----------------------------Length of the current string*/
      // Please refer to the manual for the structure of an S7 string.
      // Since the byte data type was selected in the example, the station values are
      // returned as individual bytes.
      // Content of the first output word.
      word1 = (myArray[0]<<8) + (myArray[1]);
      /*---------------------------------Load second byte*/
      /*-------------------Shift left 8 places*/
      /*---------------Load first byte*/
      // Content of the second output word.
      word2 = (myArray[2]<<8) + (myArray[3]);
      /*---------------------------------Load second byte*/
      /*-------------------Shift left 8 places*/
      /*---------------Load first byte*/
      // Content of the first output double word.
      dword1 = (myArray[4]<<24) + (myArray[5]<<16) +
      /*-------------------Shift left 16 places*/
      /*-----------------------------------Load second byte*/
      /*--------------------Shift left 24 places*/
      /*----------------Load first byte*/
         (myArray[6]<<8) + (myArray[7]);
      /*----------------------------Load fourth byte*/
      /*---------------Shift left 8 places*/
      /*-----------Load third byte*/
      // Content of the second output double word.
      tod = (myArray[8]<<24) + (myArray[9]<<16) +
      /*-------------------------------------Shift left 16 places*/
      /*---------------------------------Load second byte*/
      /*-----------------Shift left 24 places*/
      /*-------------Load first byte*/
         (myArray[10]<<8) + (myArray[11]);
      /*----------------------------Load fourth byte*/
      /*---------------Shift left 8 places*/
      /*-----------Load third byte*/
      // Loading the PLC text string.
      // Each character is converted individually from the station byte stream.
      for (int i = 0; i < myArray[13]; i++) {
         /*--------------------------Current number of characters*/
         // Reloading the individual character
         text[i] = (char)myArray[14 + i];
         /*---------------------------nth character*/
```

```
            /*----------------------Starting point in byte stream for first character*/
            /*---------Converts PLC value to char*/
         }
         // Output of the values in the applet
         tFWord1.setText("" + word1);
         tFWord2.setText("" + word2);
         tFDWord1.setText("" + dword1);
         tFTimeOfDay1.setText("" + tod);
         tFText1.setText(new String(text));
      }
   }
   /**
    * Method for handling events for the ActionListener interface.
    *
    * @param e java.awt.event.ActionEvent
    */
   public void actionPerformed(ActionEvent e) {
      // Query whether CLTimer triggered.
      if (e.getSource() == cLTimer1) {
         // If YES, read values from the PLC.
         // Reading is triggered by the processGet() method
         // of S7Variable bean.
         // If the new values exist, the S7Variable bean triggers a
         // PropertyChangeEvent.
         s7Variable1.processGet();
      }
   }
}
```

6.1.8 Example 8 – Inputting Several Values Using one Variable

How the Function Works

You want to output and input several values using one variable.

32 bytes of data are read from and written to DB4 starting at byte 0.

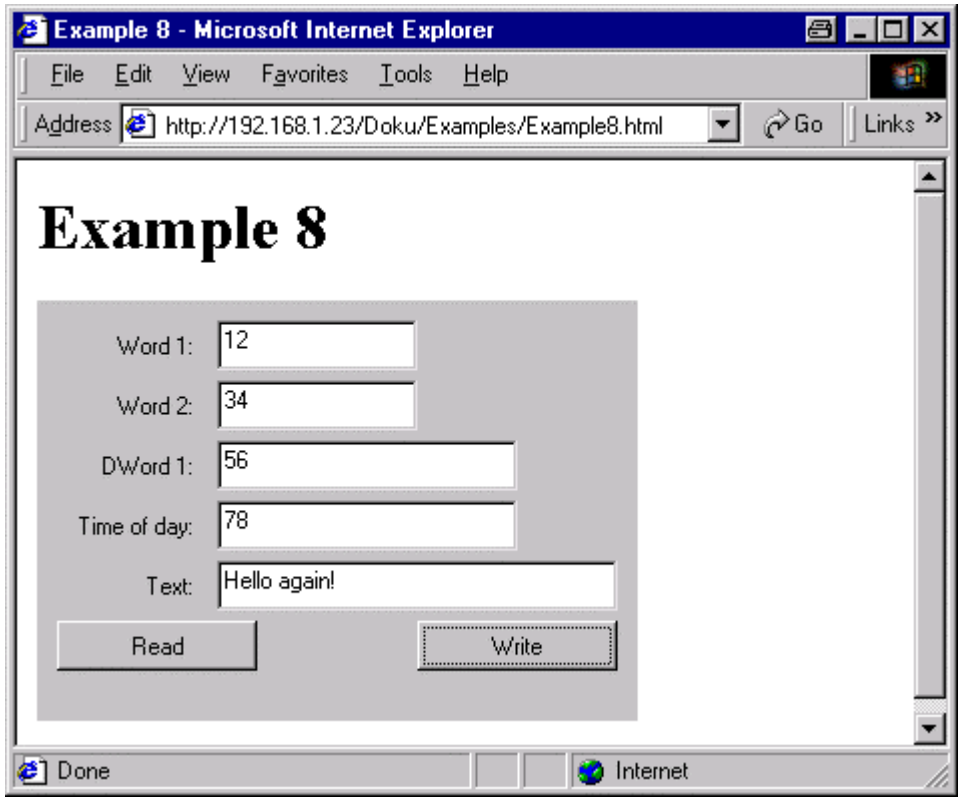
The data is read and written when a button is clicked.

Structure of the DB

DB4 -- ITCP400\SIMATIC 400\CPU 416-2 DP

Address	Name	Type	Initial value	Actual value	Comment
0.0	Word1	WORD	W#16#1212	W#16#1212	
2.0	Word2	WORD	W#16#3434	W#16#3434	
4.0	DWord1	DWORD	DW#16#56565656	DW#16#56565656	
8.0	TimeOfDay	TIME_OF_DAY	TOD#14:17:36.586	TOD#14:17:36.586	
12.0	Text	STRING [12]	'Hello World!'	'Hello World!'	

representation



Source Text

```
// This statement assigns all classes whose source text contains
// this statement to a package.
package de.siemens.simaticnet.itcp.example;
// By importing a package or a class, all declarations
// are made visible that may be made visible in other packages
// based on their access class.
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.beans.*;
import de.siemens.simaticnet.itcp.api.*;
/**
 * Example8.java
 * <p>Title: Example 8 - Using the ITCP Beans.</p>
 * <p>Description: Outputting and inputting several values using AWT components with an
 * S7Variable.</p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Organization: Siemens AG SIMATIC NET</p>
 *
 * Outputting and Inputting Several Values Using one Variable.
 * 32 bytes of data are read from and written to DB4 starting at byte 0.
 *
 * Components used:
 * S7CP
 * S7Device
 * S7Variable
 * AWT Label to describe the output fields
 * AWT TextField for each of the 5 values
 * AWT Button for reading and writing
 *
 * Structure of the data block DB4 as STL source for STEP7
 * DATA_BLOCK DB 4
 * STRUCT
 * Word1 : WORD  := W#16#1212;
 * Word2 : WORD  := W#16#3434;
 * DWord1 : DWORD  := DW#16#56565656;
 * TimeOfDay : TIME_OF_DAY  := TOD#14:17:36.586;
 * Text : STRING  [12 ] := 'Hello World!';
 * END_STRUCT ;
 * BEGIN
 * END_DATA_BLOCK
 *
 *
 * @author ITCP Team
 * @version 1.0
 *
 */
public class Example8 extends Applet implements PropertyChangeListener, ActionListener {
   /*-----------------------------------Implemented Interfaces*/
   /*----------------------------Basic Class Applet*/
   // Declaration of the required components
   private S7CP s7CP1 = null;
   private S7Device s7Device1 = null;
   private S7Variable s7Variable1 = null;
   private TextField tFWord1 = null;
   private Label lWord1 = null;
   private TextField tFWord2 = null;
   private Label lWord2 = null;
   private TextField tFDWord1 = null;
   private Label lDWord1 = null;
   private TextField tFTimeOfDay1 = null;
   private Label lTimeOfDay1 = null;
   private TextField tFText1 = null;
```

```
private Label lText1 = null;
private Button button1 = null;
private Button button2 = null;
/**
 * Is always called when the applet is initialized.
 * This occurs immediately after it is loaded.
 *
 * @see #start
 * @see #stop
 * @see #destroy
 */
public void init() {
   super.init();
   // Sets the name of the component to the specified character string.
   setName("Example8");
   // Sets the layout manager for this component.
   setLayout(null);
   // Sets the size of the applet width / height
   setSize(426, 240);
   /*-----------Height*/
   /*------Width*/
   // Create an instance for the S7CP bean.
   // S7CP is the Ethernet access point to the station
   s7CP1 = new S7CP();
   // Assign the IP address
   // ######## Project-specific adaptation of the IP address necessary          ########
   s7CP1.setHostString(new HostString ("192.168.1.1:80"));
   /*-------------------------------------------------Specify port number
                                                      normally :80*/
   /*-----------------------------------IP address as string*/
   // Create an instance for the S7Device bean.
   // S7Device is used to address the communication partner in the station.
   s7Device1 = new S7Device();
   // The address is made up of the rack number and slot number of the
   // module. The default of both methods for addressing is '0'.
   // This means that the rack number is unnecessary (.setRack(0)).
   // As we want to communicate with the CPU, the slot of the CPU must
   // be entered.
   // ######## Project-specific adaptation of the rack and slot number  ########
   // ######## necessary                                               ########
   s7Device1.setSlot(2);
   /*----------------Slot number 2 (int)*/
   // Create an instance for the S7Variable bean.
   // The S7Variable bean represents the variable to be read
   // or written.
   s7Variable1 = new S7Variable();
   // The variable is described by an S7 ANY pointer
   s7Variable1.setS7Anypointer(
      new S7Anypointer((int)2, (int)26, (int)132, (int)4, (int)0, (int)0));
   /*-----------------------------------------------------------------Bit number 0 ..7*/
   /*-------------------------------------------------------Memory area offset*/
   /*-----------------------------------------------DB number or '0'*/
   /*-----------------------------------Memory area 132 == DB*/
   /*--------------------------Repetition factor 1 .. n*/
   /*-------------------Data type 2 == Byte*/
   // Sets the name of the component to the specified character string.
   s7Variable1.setVariableName("s7Variable1");
   // Create an instance for a label.
   lWord1 = new Label();
   // Sets the output text to the specified character string.
   lWord1.setText("Word 1:");
   // Specify the alignment of the display text.
   lWord1.setAlignment(Label.RIGHT);
   // Sets the name of the component to the specified character string.
```

```
lWord1.setName("Text1");
// Specify the start position and size of the component.
lWord1.setBounds(10, 10, 70, 25);
/*--------------------------Component height*/
/*-----------------------Component width*/
/*-------------------Start position Y*/
/*---------------Start position X*/
// Insert the component in the applet.
add(lWord1, lWord1.getName());
// Create an instance for a TextField.
tFWord1 = new TextField();
// Sets the output text to the specified character string.
tFWord1.setText("");
// Sets the name of the component to the specified character string.
tFWord1.setName("TextField1");
// Specify the start position and size of the component.
tFWord1.setBounds(90, 10, 100, 25);
/*----------------------------Component height*/
/*-------------------------Component width*/
/*--------------------Start position Y*/
/*----------------Start position X*/
// Insert the component in the applet.
add(tFWord1, tFWord1.getName());
lWord2 = new Label();
lWord2.setText("Word 2:");
lWord2.setAlignment(Label.RIGHT);
lWord2.setName("Text2");
lWord2.setBounds(10, 40, 70, 25);
add(lWord2, lWord2.getName());
tFWord2 = new TextField();
tFWord2.setText("");
tFWord2.setName("TextField2");
tFWord2.setBounds(90, 40, 100, 25);
add(tFWord2, tFWord2.getName());
lDWord1 = new Label();
lDWord1.setText("DWord 1:");
lDWord1.setAlignment(Label.RIGHT);
lDWord1.setName("Text3");
lDWord1.setBounds(10, 70, 70, 25);
add(lDWord1, lDWord1.getName());
tFDWord1 = new TextField();
tFDWord1.setText("");
tFDWord1.setName("TextField3");
tFDWord1.setBounds(90, 70, 150, 25);
add(tFDWord1, tFDWord1.getName());
lTimeOfDay1 = new Label();
lTimeOfDay1.setText("Time of day:");
lTimeOfDay1.setAlignment(Label.RIGHT);
lTimeOfDay1.setName("Text4");
lTimeOfDay1.setBounds(10, 100, 70, 25);
add(lTimeOfDay1, lTimeOfDay1.getName());
tFTimeOfDay1 = new TextField();
tFTimeOfDay1.setText("");
tFTimeOfDay1.setName("TextField4");
tFTimeOfDay1.setBounds(90, 100, 150, 25);
add(tFTimeOfDay1, tFTimeOfDay1.getName());
lText1 = new Label();
lText1.setText("Text:");
lText1.setAlignment(Label.RIGHT);
lText1.setName("Text5");
lText1.setBounds(10, 130, 70, 25);
add(lText1, lText1.getName());
tFText1 = new TextField();
tFText1.setText("");
```

```
    tFText1.setName("TextField5");
    tFText1.setBounds(90, 130, 200, 25);
    add(tFText1, tFText1.getName());
    button1 = new Button();
    button1.setLabel("Read");
    button1.setName("Button1");
    button1.setBounds(10, 160, 100, 25);
    add(button1, button1.getName());
    button2 = new Button();
    button2.setLabel("Write");
    button2.setName("Button2");
    button2.setBounds(190, 160, 100, 25);
    add(button2, button2.getName());
    // Apart from the definition of the methods to be executed when an event
    // occurs, an object must register with the corresponding
    // event source.
    // This is done by calling the addXXXListener method of the event source,
    // where »XXX« stands for the corresponding event type.
    // The addXXXListener methods all expect a reference to the relevant
    // interface:
    s7CP1.addPropertyChangeListener(this);
    s7Device1.addPropertyChangeListener(this);
    s7Variable1.addPropertyChangeListener(this);
    button1.addActionListener(this);
    button2.addActionListener(this);
}
/**
 * Executes after initialization of an applet.
 * With browsers, start() is then also called when a page containing
 * an applet is reloaded.
 *
 * @see #init
 * @see #stop
 * @see #destroy
 */
public void start() {
    super.start();
}
/**
 * There is a call when the browser or the applet viewer is minimized
 * to an icon or an HTML page containing an applet is exited in
 * a browser.
 *
 * @see #init
 * @see #start
 * @see #destroy
 */
public void stop() {
    super.stop();
}
/**
 * Is always called when the applet is destroyed.
 *
 * @see #init
 * @see #start
 * @see #stop
 */
public void destroy() {
    super.destroy();
    // This method deletes all S7Bean instances and discards all threads.
    // After calling this method, a reinitialization is necessary.
    S7Api.terminate();
}
/**
```

```
 * Method for handling events for the PropertyChangeListener interface.
 *
 * @param evt PropertyChangeEvent
 */
public void propertyChange(PropertyChangeEvent evt) {
   // Query whether event was triggered by S7CP.
   if (evt.getSource() == s7CP1)
      // If YES
      // Pass event to the S7Device instance
      s7Device1.propertyChange(evt);
   // Query whether or not event was triggered by S7Device.
   if (evt.getSource() == s7Device1)
      // If YES
      // Pass event to the S7Variable instance
      s7Variable1.propertyChange(evt);
   // Query whether or not event was triggered by S7Variable.
   if (evt.getSource() == s7Variable1) {
      // If YES, the new data from the station is included in the event.
      // The new data is made available in an int[] (IntegerArray) by the
      // getNewValue() method.
      // Reload the data into a local auxiliary variable (myArray)
      int[] myArray = (int[])evt.getNewValue();
      // Declaration of local auxiliary variables for reloading individual
      // output values.
      int word1, word2;
      long dword1, tod;
      char[] text = new char[myArray[13]];
      /*-----------------------------Length of the current string*/
      // Please refer to the manual for the structure of an S7 string.
      // Since the byte data type was selected in the example, the station values are
      // returned as individual bytes.
      // Content of the first output word.
      word1 = (myArray[0]<<8) + (myArray[1]);
      /*---------------------------------Load second byte*/
      /*-------------------Shift left 8 places*/
      /*---------------Load first byte*/
      // Content of the second output word.
      word2 = (myArray[2]<<8) + (myArray[3]);
      /*---------------------------------Load second byte*/
      /*-------------------Shift left 8 places*/
      /*---------------Load first byte*/
      // Content of the first output double word.
      dword1 = (myArray[4]<<24) + (myArray[5]<<16) +
      /*-----------------------------------------Shift left 16 places*/
      /*---------------------------------Load second byte*/
      /*--------------------Shift left 24 places*/
      /*----------------Load first byte*/
         (myArray[6]<<8) + (myArray[7]);
      /*-----------------------------Load fourth byte*/
      /*---------------Shift left 8 places*/
      /*-----------Load third byte*/
      // Content of the second output double word.
      tod = (myArray[8]<<24) + (myArray[9]<<16) +
      /*-----------------------------------Shift left 16 places*/
      /*-------------------------------Load second byte*/
      /*-----------------Shift left 24 places*/
      /*-------------Load first byte*/
         (myArray[10]<<8) + (myArray[11]);
      /*-----------------------------Load fourth byte*/
      /*----------------Shift left 8 places*/
      /*-----------Load third byte*/
      // Loading the PLC text string.
      // Each character is converted individually from the station byte stream.
      for (int i = 0; i < myArray[13]; i++) {
```

```
            /*-------------------------Current number of characters*/
            // Reloading the individual character
            text[i] = (char)myArray[14 + i];
            /*---------------------------nth character*/
            /*----------------------Starting point in byte stream for first character*/
            /*---------Converts PLC value to char*/
         }
         // Output of the values in the applet
         tFWord1.setText("" + word1);
         tFWord2.setText("" + word2);
         tFDWord1.setText("" + dword1);
         tFTimeOfDay1.setText("" + tod);
         tFText1.setText(new String(text));
      }
   }
   /**
    * Method for handling events for the ActionListener interface.
    *
    * @param e java.awt.event.ActionEvent
    */
   public void actionPerformed(ActionEvent e) {
      // Query whether or not read button was triggered.
      if (e.getSource() == button1) {
         // If YES, read values from the PLC.
         // Reading is triggered by the processGet() method
         // of S7Variable bean.
         // If the new values exist, the S7Variable bean triggers a
         // PropertyChangeEvent.
         s7Variable1.processGet();
      }
      // Query whether or not write button was triggered.
      if (e.getSource() == button2) {
         // If YES, write values to PLC.
         // Declaration of local auxiliary variables for reloading individual
         // written values.
         String text;
         int iTmp;
         long lTmp;
         int[] myArray = new int[32];
         /*----------------------Size of the communication area in the example 32 bytes*/
         // Write first input value in the byte stream (WORD).
         iTmp = Integer.valueOf(tFWord1.getText()).intValue();
         /*-----------------------------------------Convert integer value to 'int' */
         /*----------------------------Read input text field*/
         /*-------------Converts the content of string to an integer value */
         myArray[0] = (iTmp & 0xFF00) >>> 8;
         /*---------------------------Shift 8 places right and pad with
                                 zeros*/
         /*-------------------Mask for first byte*/
         myArray[1] = iTmp & 0x00FF;
         /*-------------------Mask for second byte*/
         // Write second input value in the byte stream (WORD).
         iTmp = Integer.valueOf(tFWord2.getText()).intValue();
         /*-----------------------------------------Convert integer value to 'int' */
         /*----------------------------Read input text field*/
         /*-------------Converts the content of string to an integer value */
         myArray[2] = (iTmp & 0xFF00) >>> 8;
         /*---------------------------Shift 8 places right and pad with
                                 zeros*/
         /*-------------------Mask for first byte*/
         myArray[3] = iTmp & 0x00FF;
         /*-------------------Mask for second byte*/
         // Write third input value in the byte stream (DWORD).
         lTmp = Long.valueOf(tFDWord1.getText()).longValue();
```

```
/*-------------------------------------Convert long value to 'long' */
/*--------------------------Read input text field*/
/*----------Converts the content of string to a long value */
myArray[4] = (int)((lTmp & 0xFF000000) >>> 24);
/*------------------------------------ Shift 24 places right and
                                  pad with zeros */
/*-------------------------Mask for first byte*/
/*------------Convert to int*/
myArray[5] = (int)((lTmp & 0x00FF0000) >>> 16);
/*------------------------------------ Shift 16 places right and
                                pad with zeros */
/*-------------------------Mask for second byte*/
/*------------Convert to int*/
myArray[6] = (int)((lTmp & 0x0000FF00) >>> 8);
/*-------------------------------------Shift 8 places right and
                                  pad with zeros */
/*-------------------------Mask for third byte*/
/*------------Convert to int*/
myArray[7] = (int)(lTmp & 0x000000FF);
/*-------------------------Mask for fourth byte*/
/*------------Convert to int*/
// Write fourth input value to the byte stream (TIME_OF_DAY).
lTmp = Long.valueOf(tFTimeOfDay1.getText()).longValue();
/*-----------------------------------------Convert long value to 'long' */
/*-----------------------------Read input text field*/
/*----------Converts the content of string to a long value */
myArray[8] = (int)((lTmp & 0xFF000000) >>> 24);
/*-------------------------------------Shift 24 places right and
                                  pad with zeros */
/*-------------------------Mask for first byte*/
/*------------Convert to int*/
myArray[9] = (int)((lTmp & 0x00FF0000) >>> 16);
/*-------------------------------------Shift 16 places right and
                                  pad with zeros */
/*-------------------------Mask for second byte*/
/*------------Convert to int*/
myArray[10] = (int)((lTmp & 0x0000FF00) >>> 8);
/*-------------------------------------Shift 8 places right and
                                   pad with zeros*/
/*-------------------------Mask for third byte*/
/*------------Convert to int*/
myArray[11] = (int)(lTmp & 0x000000FF);
/*-------------------------Mask for fourth byte*/
/*------------Convert to int*/
// Write fourth input value to the byte stream (STRING).
text = tFText1.getText();
/*-------------Read input text field and reload*/
myArray[12] = 18;
/*------------Reserve maximum length for the string*/
/*------Position of the maximum length in the byte stream*/
// Check current length against max. length
if (text.length() > myArray[12]) {
   // If current length greater than maximum length,
   // then current length equal to max. length
   myArray[13] = myArray[12];
   /*--------------------Max. length*/
   /*------Current length*/
} else {
   // If current length less than maximum length,
   // then current length equal to input length
   myArray[13] = text.length();
   /*-----------------Input length*/
   /*------Current length*/
}
```

```
         // Write the individual characters to the byte stream
         for (int i=0;i<myArray[13];i++) {
            /*---------------------Current length*/
            myArray[14+i] = text.charAt(i);
            /*--------------------------Read nth character*/
            /*---------nth position in byte stream*/
            /*------Starting point in byte stream for first character*/
         }
         // The new data is passed to the setValue() method in an
         // int[] (IntegerArray).
         // The setValue() method then writes the byte stream to the station.
         // Calling setValue() automatically retriggers the processGet() method to
         // read the area.
         s7Variable1.setValue(myArray);
         // The waitOnNewData method only allows setValue() to be called again
         // only after a wait time has elapsed or after the arrival of new data.
         // This prevents fast button clicks.
         s7Variable1.waitOnNewData(2000);
         /*------------------------Wait time in msec. 2000 == 2 sec.*/
      }
   }
}
```

6.2 Example of Working with JBuilder

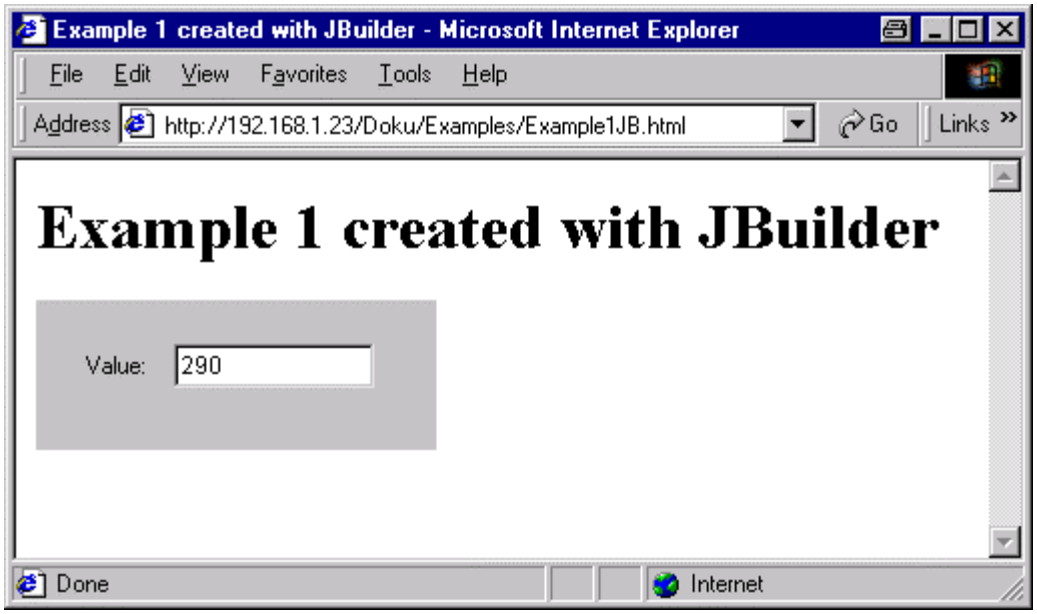
How the Function Works

A variable will be read from the SIMATIC S7 CPU and displayed.

The memory word MW 10 will be read as INT.

The example is created with the Java development environment JBuilder from Borland and in terms of function corresponds to Example 1 from Section 6.1.1. Only the source text is structured differently.

representation



Source Text

```
// This statement assigns all classes whose source text contains
// this statement to a package.
package de.siemens.simaticnet.itcp.example;
// By importing a package or a class, all declarations
// are made visible that may be made visible in other packages
// based on their access class.
import java.applet.*;
import java.awt.event.*;
import java.beans.*;
import de.siemens.simaticnet.itcp.api.*;
import de.siemens.simaticnet.itcp.gui.*;
/**
 * Example1JB.java
 * <p>Title: Example 1 of Using the ITCP Beans with the JBuilder IDE.</p>
 * <p>Description: Outputting a value using the CLTextOut Bean with an S7Variable.</p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Organization: Siemens AG SIMATIC NET</p>
 *
```

```
 * Outputting a value using the CLTextOut Bean with an S7Variable.
 * The memory word MW10 is read.
 *
 * Components used:
 * S7CP
 * S7Device
 * S7Variable
 * CLTimer
 * CLTextOut
 *
 * @author ITCP Team
 * @version 1.0
 *
 */
public class Example1JB extends Applet {
   /*-----------------------------Basic Class Applet*/
   // Declaration of the required components
   private S7CP s7CP1;
   private S7Device s7Device1;
   private S7Variable s7Variable1;
   private CLTimer cLTimer1;
   private CLTextOut cLTextOut1;
   // Construct the applet
   public Example1JB() {
   }
   /**
    * Is always called when the applet is initialized.
    * This occurs immediately after it is loaded.
    *
    * @see #start
    * @see #stop
    * @see #destroy
    */
   public void init() {
      try {
         // Initialization of the components
         jbInit();
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
   // Initialization of the components
   private void jbInit() throws Exception {
      // Create an instance for the S7CP bean.
      // S7CP is the Ethernet access point to the station
      s7CP1 = (S7CP) Beans.instantiate(getClass().getClassLoader(),
                              S7CP.class.getName());
      // Create an instance for the S7Device bean.
      // S7Device is used to address the communication partner in the station.
      s7Device1 = (S7Device) Beans.instantiate(getClass().getClassLoader(),
                                    S7Device.class.getName());
      // Create an instance for the S7Variable bean.
      // The S7Variable bean represents the variable to be read
      // or written.
      s7Variable1 = (S7Variable) Beans.instantiate(getClass().getClassLoader(),
                                       S7Variable.class.getName());
      // Create an instance for the CLTimer bean.
      // The CLTimer bean triggers a PropertyChangeEvent when
      // the time elapses. This event implements a cyclic data update
      cLTimer1 = (CLTimer) Beans.instantiate(getClass().getClassLoader(),
                                    CLTimer.class.getName());
      // Create an instance for the CLTextOut bean.
      // Using the CLTextOut bean, you as user can enter elementary
      // variables.
```

```
cLTextOut1 = (CLTextOut) Beans.instantiate(getClass().getClassLoader(),
                                    CLTextOut.class.getName());
// The setDelay() method sets the time interval.
cLTimer1.setDelay(2000);
/*----------------Time interval in msec. 2000 == 2 sec.*/
// This type of event adapter for interior classes created by JBuilder
// is known as an anonymous adapter. They prevent a separate
// adapter class from being created. The resulting code is compact.
// Create an event adapter for the CLTimer Bean.
cLTimer1.addActionListener(new java.awt.event.ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                           // Call the event method
                           cLTimer1_actionPerformed(e);
                        }
                     });
// The variable is described by an S7 ANY pointer
s7Variable1.setS7Anypointer(
   new de.siemens.simaticnet.itcp.api.S7Anypointer(
      (int)5, (int)1, (int)131, (int)0, (int)10, (int)0));
/*-------------------------------------------------Bit number 0 ..7*/
/*----------------------------------------Memory area offset*/
/*--------------------------------DB number or '0'*/
/*---------------------Memory area 131 == M*/
/*-------------Repetition factor 1 .. n*/
/*------Data type 5 == INT*/
// Create an event adapter for the S7Variable Bean.
s7Variable1.addPropertyChangeListener(new java.beans.PropertyChangeListener() {
   public void propertyChange(PropertyChangeEvent e) {
      // Call the event method
      s7Variable1_propertyChange(e);
   }
});
// The address is made up of the rack number and slot number of the
// module. The default of both methods for addressing is '0'.
// This means that the rack number is unnecessary (.setRack(0)).
// As we want to communicate with the CPU, the slot of the CPU must
// be entered.
// ######## Project-specific adaptation of the rack and slot number  ########
// ######## necessary                                                ########
s7Device1.setSlot(2);
/*----------------Slot number 2 (int)*/
// Create an event adapter for the S7Device Bean.
s7Device1.addPropertyChangeListener(new java.beans.PropertyChangeListener() {
   public void propertyChange(PropertyChangeEvent e) {
      // Call the event method
      s7Device1_propertyChange(e);
   }
});
// Assign the IP address
// ######## Project-specific adaptation of the IP address necessary          ########
s7CP1.setHostString(
   new de.siemens.simaticnet.itcp.api.HostString ("192.168.1.1:80"));
/*-----------------------------------------------------------Specify port
                                          number normally :80*/
/*-------------------------------------------------IP address as string*/
// Create an event adapter for the S7CP Bean.
s7CP1.addPropertyChangeListener(new java.beans.PropertyChangeListener() {
   public void propertyChange(PropertyChangeEvent e) {
      // Call the event method
      s7CP1_propertyChange(e);
   }
});
// Sets the dimension text to the specified character string.
cLTextOut1.setUnit("");
```

```
    // Sets the descriptive text to the specified character string.
    cLTextOut1.setLabel("Value:");
    // Specify the size of the output box.
    cLTextOut1.setOutFieldSize(100);
    // Insert the component in the applet.
    this.add(cLTextOut1, null);
}
/**
 * Executes after initialization of an applet.
 * With browsers, start() is then also called when a page containing
 * an applet is reloaded.
 *
 * @see #init
 * @see #stop
 * @see #destroy
 */
public void start() {
    super.start();
}
/**
 * There is a call when the browser or the applet viewer is minimized
 * to an icon or an HTML page containing an applet is exited in
 * a browser.
 *
 * @see #init
 * @see #start
 * @see #destroy
 */
public void stop() {
    super.stop();
}
/**
 * Is always called when the applet is destroyed.
 *
 * @see #init
 * @see #start
 * @see #stop
 */
public void destroy() {
    super.destroy();
    // This method deletes all S7Bean instances and discards all threads.
    // After calling this method, a reinitialization is necessary.
    S7Api.terminate();
}
/**
 * Method for handling events for the PropertyChangeListener of the S7CP bean.
 *
 * @param e PropertyChangeEvent
 */
void s7CP1_propertyChange(PropertyChangeEvent e) {
    // Pass event to the S7Device instance
    s7Device1.propertyChange(e);
}
/**
 * Method for handling events for the PropertyChangeListener of the
 * S7Device bean.
 *
 * @param e PropertyChangeEvent
 */
void s7Device1_propertyChange(PropertyChangeEvent e) {
    // Pass event to the S7Variable instance
    s7Variable1.propertyChange(e);
}
/**
```

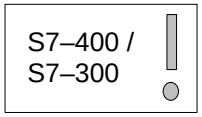
```
     * Method for handling events for the PropertyChangeListener of the
     * S7Variable Bean.
     *
     * @param e PropertyChangeEvent
     */
    void s7Variable1_propertyChange(PropertyChangeEvent e) {
        // Then transfer output value to the CLTextOut instance.
        cLTextOut1.propertyChange(e);
    }
    /**
     * Method for handling events for the ActionPerformedEvent of the CLTimer Bean.
     *
     * @param e ActionPerformedEvent
     */
    void cLTimer1_actionPerformed(ActionEvent e) {
        // Reading is triggered by the processGet() method
        // of S7Variable bean.
        // If the new values exist, the S7Variable bean triggers a
        // PropertyChangeEvent.
        s7Variable1.processGet();
    }
}
```

6.3 Developing Your Beans

The S7 beans comply with the conventions of JavaBeans and can always be accessed by getter and setter methods. The S7 beans also understand special PropertyChangeEvents. This means that an output bean (such as CLTextOut or CLLevel) passes the value received from an S7Variable using a PropertyChangeEvent directly to its setValue() method and displays the value.

The “S7–Variable” S7 bean, for example, adopts PCE name InValue and passes on the linked value to the CPU.

You create beans that interact with the S7BeansAPI as explained below (the example below applies only to the S7–300 / S7–400).



This is explained below based on the CLIdentOut.

CLIdentOut is an S7 bean required for the textual display of an identification number of an IT-CP or module using the S7CP bean or S7Device bean.

```
//Declaration Section

package de.siemens.simaticnet.itcp.gui;

import java.awt.*;
import java.awt.event.*;
import java.beans.*;
import java.io.Serializable;
import java.util.*;
public class CLIdentOut
extends Container
implements PropertyChangeListener, Serializable {

        private int outFieldSize;
        private String label;
        private Color outFieldColor = newColor (255, 255, 255);
        private Font theFont = new Font (”serif”, 0, 12);
        private String outValue;
        private Label label1 = new Label ();
        private TextField textField1 = new TextField ();
        private static CLGUIMessage international = CLGUIMessage.getInstance();

        public CLIdentOut() {
        try  {
        jbInit();
        }
        catch (Exception ex) {
        ex.printStackTrace();
        }
        }
```

//Initialize everything necessary first:

```
private void jbInit() throws Exception {
        this.setLayout(null);
        this.setSize(new Dimension(230, 60));
        this.setLayout(null);
        this.add(label1, null);
        this.add(textField1, null);


        label1.setAlignment(1);
        label1.setFont(theFont);
        label1.setText("?label?");
        label1.setBounds(new Rectangle(15, 17, 52,
        textField1.getMinimumSize(1).height));
        textField1.setFont(theFont);
        textField1.setEditable(false);
        textField1.setBackground(outFieldColor);
        textField1.setBounds(new Rectangle(66, 17, 160,
        textField1.getMinimumSize(1).height));
}
```

//The labeling is set here:

```
public void setLabel(String newLabel) {
        label = newLabel;
        label1.setText(newLabel);
        FontMetrics fm = label1.getFontMetrics(label1.getFont());
        int newWidth = fm.stringWidth(label)+ 20;
        if (newWidth != label1.getSize().width) {
                int h = newWidth – label1.getSize().width;
                Point p = new Point(label1.getLocation());
                label1.setBounds(p.x,p.y , newWidth,
                textField1.getMinimumSize(1).height);
                p = textField1.getLocation();
                p.translate(h,0);
                textField1.setBounds(p.x,p.y , textField1.getSize().width,
                textField1.getMinimumSize(1).height);
        }
}

public String getLabel() {
return label1.getText();
}
```

//The color of the TextField:

```
public void setOutFieldColor(Color newOutFieldColor) {
outFieldColor = newOutFieldColor;
textField1.setBackground(newOutFieldColor);
repaint();
}

public Color getOutFieldColor() {
return  textField1.getBackground();
}
```

//The size of the TextField:

```
public void setOutFieldSize(int newOutFieldSize) {
   //outFieldSize = newOutFieldSize;
        if(textField1.getSize().width != newOutFieldSize){
                int h = newOutFieldSize – textField1.getSize().width;
                Point p = new Point(textField1.getLocation());
                textField1.setBounds(p.x, p.y, newOutFieldSize,
                textField1.getMinimumSize(1).height);
        }

        repaint();
}


public int getOutFieldSize() {
   return textField1.getSize().width;
}
```

//Set the font:

```
public void setTheFont(Font newFont) {
        theFont = newFont;
        label1.setFont(newFont);
        textField1.setFont(newFont);

        repaint();
}


public Font getTheFont() {
   return theFont;
}
public Dimension getPreferredSize() {
   return new Dimension(230,60);
}
```

The specific code section that you must use in the appropriate beans follows here.

```
//This checks whether the incoming event (PropertyChangeEvent) contains
//the module identification. Only then is the event evaluated and
//the value passed to setValue() so that the value can be output:

        public void propertyChange(PropertyChangeEvent pce) {
                if (pce.getPropertyName() == ”identification”) {
                        try {
                        setValue(pce.getNewValue().toString());
                        } catch (Exception e) {}
                }
        }
```

```
//The passed string is displayed in the TextField:

        public void setValue(String newOutValue) {
                outValue = newOutValue;
                textField1.setText(outValue);
        }

        public String getValue() {
           return outValue;
        }

        public void addMouseListener(MouseListener listener) {
                super.addMouseListener(listener);
                label1.addMouseListener(listener);
                textField1.addMouseListener(listener);
        }

        public void removeMouseListener(MouseListener listener) {
                super.removeMouseListener(listener);
                label1.removeMouseListener(listener);
                textField1.removeMouseListener(listener);
        }

        public synchronized void paint(Graphics g) {
                super.paint(g);

                Rectangle bl = label1.getBounds();
                Rectangle bt = textField1.getBounds();
                label1.setBounds(bl.x, bl.y, bl.width, textField1.getMinimumSize(1).height);
                textField1.setBounds(bt.x, bt.y, bt.width,
                textField1.getMinimumSize(1).height);
        }

}
```

6.4 Java Applications with S7Beans

Advantages

With Java applications, you have a far greater range of options than with Java applets because the applications are not subject to the strict security guidelines of the sandbox. You can, for example, implement access to a different SIMATIC stations at the same time and collect the data of these stations centrally on a PC; alternatively variable values of an S7–CPU can be stored on local PC for further processing.

Structure of Java Applications

The structure of the applications is somewhat different from that of the applets because instead of the applet browser interface methods init(), start(), stop(), and destroy(), the main() main method is the core routine of the application:

public static void main(java.lang.String[] args);

Within this public, static method, frames and panels can once again the use just as in the applet to represent the graphic user interface.

Requirements

To create Java applications and to compile them, you require either a Java development environment such as JBuilder from Borland or VisualAge from IBM or you use the native tools of the Java Development Kit (JDK) from Sun.

How the Example Works

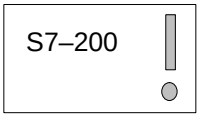
Below, we will introduce you to a sample Java application that reads data cyclically from any SIMATIC S7 station (naturally equipped with an IT–CP and Ethernet connection to the local PC) and saves the data in files on the local hard disk. This data can originate from any data block with a freely selectable offset and length. (Caution! Refer to the manual for the maximum length of transferable data) .

The data can, for example, be written to files on the local hard disk that work as ring buffers. In other words, the files have numbers appended to them before the file extension. Once the last file has been written (maximum number selectable) the data is once again written to the first file with index 0. The data is read as binary data and stored as binary data in the file.

Logging can be started and stopped with a central button and the success of the operation or error messages are displayed in the status bar.

The Sample as the Basis for Concrete Applications

The source text of the sample can be used as a basis for more complex applications since it illustrates the essential features of using the S7Beans for data exchange and file access. To keep the sample short, complex exception handling and validity checks have been deliberately omitted. To implement the graphic user interface, the sample uses swing components and should therefore be compiled with a Java2 compiler and started in a Java2 runtime environment (JRE 1.2.x, 1.3.x or 1.4.x).



Notice

The S7–200 does not support data blocks. You should therefore always enter ”1” as the DB number. This means that you automatically access the ”variable memory” of the S7–200. The STL source in the sample must be adapted for the S7–200.

Source Text

```
// Assignment of the classes to the ”IT-CP Example” package
package de.siemens.simaticnet.itcp.example;

// Display the name spaces of the classes used
import de.siemens.simaticnet.itcp.api.*;
import java.io.*;
import javax.swing.*;
import javax.swing.border.*;
import java.beans.*;
import java.awt.*;
import java.awt.event.*;
/**
 * LogfileDemo.java
 * <p>Title: Example of Using S7Beans in Java Applications.</p>
 * <p>Description: Reading data from a SIMATIC S7 CPU cyclically and <br>
 * writing the data to files on the local PC.</p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Organization: Siemens AG SIMATIC NET</p>
 *
 * Reading data from a SIMATIC S7 CPU cyclically and
 * writing the data to the local PC. Using
 * SWING components under Java2.
 *
 * Components used:
 * S7CP
 * S7Device
 * S7Variable
 * CLTimer
 * SWING components JFrame, JPanel, JLabel, JTextField and JButton
 *
 * @author ITCP Team
 * @version 1.0
 *
 */
public class LogfileDemo extends JFrame implements ActionListener, PropertyChangeListener {

   // Declaration of the required components
   private JPanel JFrameContentPane = null;
   private JPanel LogfileDemoPane = null;
```

```
private JButton btnStartStop = null;
private CLTimer s7Timer = null;
private S7CP s7CP = null;
private S7Device s7Device = null;
private S7Variable s7Variable = null;
private JTextField tfDB = null;
private JTextField tfExtension = null;
private JTextField tfHostname = null;
private JTextField tfLength = null;
private JTextField tfOffset = null;
private JTextField tfPeriod = null;
private JTextField tfPrefix = null;
private JTextField tfRack = null;
private JTextField tfSlot = null;
private JLabel labCPOptions = null;
private JLabel labCPUOptions = null;
private JLabel labData = null;
private JLabel labExtension = null;
private JLabel labHostname = null;
private JLabel labLogOptions = null;
private JLabel labPeriod = null;
private JLabel labPrefix = null;
private JLabel labRackSlot = null;
private boolean bRunning = false;
private int iCounter = 0;
private JLabel labMaxCounter = null;
private JTextField tfMaxCounter = null;
private JLabel labMsg1 = null;
private JLabel labMsg2 = null;
private JPanel StatusBarPane = null;
/**
 * Constructor with the initialization of all objects.
 and*/
public LogfileDemo() {
   // Call constructor of the basic class
   super();
   // Set important window attributes
   // Window title
   setTitle("Logfile Demo");
   // Window size
   setSize(400, 400);
   // Exit application on closing window
   setDefaultCloseOperation(EXIT_ON_CLOSE);

   // Initialize all user interface elements
   // Create object
   labCPOptions = new JLabel();
   // Specify text for display
   labCPOptions.setText("SIMATIC S7 IT-CP Options");
   // Set position and size
   labCPOptions.setBounds(20, 15, 181, 20);
   labHostname = new JLabel();
   labHostname.setText("Hostname or IP-Address:");
   labHostname.setBounds(34, 44, 163, 14);
   tfHostname = new JTextField();
   tfHostname.setText("192.168.0.1");
   tfHostname.setBounds(210, 41, 150, 20);
   labCPUOptions = new JLabel();
   labCPUOptions.setText("SIMATIC S7 CPU Options");
   labCPUOptions.setBounds(20, 73, 181, 20);
   labRackSlot = new JLabel();
   labRackSlot.setText("Rack / Slot:");
   labRackSlot.setBounds(34, 102, 173, 14);
   tfRack = new JTextField();
```

```
tfRack.setText("0");
tfRack.setBounds(210, 99, 40, 20);
tfSlot = new JTextField();
tfSlot.setText("3");
tfSlot.setBounds(265, 99, 40, 20);
labData = new JLabel();
labData.setText("DB / offset / length:");
labData.setBounds(34, 128, 173, 14);
tfDB = new JTextField();
tfDB.setText("17");
tfDB.setBounds(210, 125, 40, 20);
tfOffset = new JTextField();
tfOffset.setText("0");
tfOffset.setBounds(265, 125, 40, 20);
tfLength = new JTextField();
tfLength.setText("8");
tfLength.setBounds(320, 125, 40, 20);
labLogOptions = new JLabel();
labLogOptions.setText("Logging Options");
labLogOptions.setBounds(20, 157, 181, 20);
labPrefix = new JLabel();
labPrefix.setText("Logfile prefix:");
labPrefix.setBounds(34, 186, 173, 14);
tfPrefix = new JTextField();
tfPrefix.setText("Demo");
tfPrefix.setBounds(210, 183, 150, 20);
labExtension = new JLabel();
labExtension.setText("Logfile extension:");
labExtension.setBounds(34, 212, 173, 14);

tfExtension = new JTextField();
tfExtension.setText("dat");
tfExtension.setBounds(210, 209, 150, 20);
labMaxCounter = new JLabel();
labMaxCounter.setText("Max. Logfile Count:");
labMaxCounter.setBounds(34, 238, 173, 14);
tfMaxCounter = new JTextField();
tfMaxCounter.setText("20");
tfMaxCounter.setBounds(210, 235, 40, 20);
labPeriod = new JLabel();
labPeriod.setText("Logging period in seconds:");
labPeriod.setBounds(34, 264, 173, 14);
tfPeriod = new JTextField();
tfPeriod.setText("3");
tfPeriod.setBounds(210, 261, 40, 20);
btnStartStop = new JButton();
btnStartStop.setText("Start Logging");
btnStartStop.setBounds(134, 315, 134, 25);
// Create a panel to be able to position the objects
// with pixel accuracy.
LogfileDemoPane = new JPanel();
LogfileDemoPane.setLayout(null);
LogfileDemoPane.add(labCPUOptions);
LogfileDemoPane.add(labHostname);
LogfileDemoPane.add(tfHostname);
LogfileDemoPane.add(labCPOptions);
LogfileDemoPane.add(labRackSlot);
LogfileDemoPane.add(tfRack);
LogfileDemoPane.add(tfSlot);
LogfileDemoPane.add(labData);
LogfileDemoPane.add(tfDB);
LogfileDemoPane.add(tfOffset);
LogfileDemoPane.add(tfLength);
LogfileDemoPane.add(labLogOptions);
```

```
    LogfileDemoPane.add(labPrefix);
    LogfileDemoPane.add(tfPrefix);
    LogfileDemoPane.add(labExtension);
    LogfileDemoPane.add(tfExtension);
    LogfileDemoPane.add(labMaxCounter);
    LogfileDemoPane.add(tfMaxCounter);
    LogfileDemoPane.add(labPeriod);
    LogfileDemoPane.add(tfPeriod);
    LogfileDemoPane.add(btnStartStop);
    // Create and position objects for
    // the status bar
    labMsg1 = new JLabel();
    labMsg1.setBorder(new EtchedBorder());
    labMsg1.setText("Not running   ");
    labMsg2 = new JLabel();
    labMsg2.setBorder(new EtchedBorder());
    labMsg2.setText("");
    StatusBarPane = new JPanel();
    StatusBarPane.setLayout(new BorderLayout());
    StatusBarPane.add(labMsg1, "West");
    StatusBarPane.add(labMsg2, "Center");
    // Create panel for positioning the main and status
    // panels. Include subordinate panels and
    // register them with the frame in the main panel
    JFrameContentPane = new JPanel();
    JFrameContentPane.setLayout(new BorderLayout());
    JFrameContentPane.add(StatusBarPane, "South");
    JFrameContentPane.add(LogfileDemoPane, "Center");
    setContentPane(JFrameContentPane);

    // Create S7Beans for data communication
    s7CP = new S7CP();
    s7Device = new S7Device();
    s7Variable = new S7Variable();
    s7Timer = new CLTimer();
    // Stop the autostart mechanism of the timer
    s7Timer.stop();

    // Register the objects with the event source.
    s7CP.addPropertyChangeListener(this);
    s7Device.addPropertyChangeListener(this);
    s7Variable.addPropertyChangeListener(this);
    s7Timer.addActionListener(this);
    btnStartStop.addActionListener(this);
}

/**
 * Method for handling events for the ActionListener
 * interface.
 *
 * @param e java.awt.event.ActionEvent
 */
public void actionPerformed(ActionEvent e) {
    // Timer elapsed?
    if (e.getSource() == s7Timer)
        // then fetch new data from SIMATIC S7
        // As soon as the new data has arrived at the S7Variable,
        // this triggers a Property-Change-Event.
        s7Variable.processGet();
    // Start/Stop button pressed?
    if (e.getSource() == btnStartStop)
        // then call relevant handler
        onStartStop();
}
```

```
/**
 * Method for handling events for the PropertyChangeListener
 * interface.
 *
 * @param evt PropertyChangeEvent
 */
public void propertyChange(PropertyChangeEvent evt) {
   // Event triggered by S7CP?
   if (evt.getSource() == s7CP)
      // then pass event to the S7Device instance
      s7Device.propertyChange(evt);
   // Event triggered by S7Device?
   if (evt.getSource() == s7Device)
      // then pass event to the S7Variable instance
      s7Variable.propertyChange(evt);
   // Event triggered by S7Variable?
   if (evt.getSource() == s7Variable)
      // then call relevant handler
      onNewS7Data(evt);
}

/**
 * Event handling routine for pressing the
 * Start/Stop button.
 */
public void onStartStop() {
   // If logging is currently active
   if (bRunning) {
      // Stop timer so that no further data
      // are fetched from the SIMATIC S7 and written
      // as files.
      s7Timer.stop();
      bRunning = false;

      // Set status text and button text accordingly
      labMsg1.setText("Not running   ");
      btnStartStop.setText("Start Logging");
   } else {
      // Read in parameter settings before starting and
      // set S7 beans accordingly..
      // Set host name or IP address
      s7CP.setHost(tfHostname.getText());

      // Set rack/slot number
      int iRack = Integer.parseInt(tfRack.getText());
      int iSlot = Integer.parseInt(tfSlot.getText());
      s7Device.setRack(iRack);
      s7Device.setSlot(iSlot);

      // Form S7 Any pointer in the format "P#DBxx.DBXyy BYTE zz"
      // from the data entered
      int iDB = Integer.parseInt(tfDB.getText());
      int iLength = Integer.parseInt(tfLength.getText());
      int iOffset = Integer.parseInt(tfOffset.getText());
      s7Variable.setS7Anypointer(new S7Anypointer(
         S7Anypointer.S7_BYTE, iLength, S7Anypointer.MEM_AREA_DB,
         iDB, iOffset, 0));

      // Set timer object to the entered time in milliseconds
      // and start it
      int iPeriod = Integer.parseInt(tfPeriod.getText())*1000;
      s7Timer.setDelay(iPeriod);
      s7Timer.start();
```

```
        bRunning = true;

        // Set status text and button text accordingly
        labMsg1.setText("Running       ");
        btnStartStop.setText("Stop Logging");
    }
}

/**
 * Event handling routing for the arrival of new data
 * from the SIMATIC S7.
 *
 * @param evt PropertyChangeEvent of S7Variable
 */
public void onNewS7Data(PropertyChangeEvent evt) {
    // Safety query so that no data arriving late
    // will be written to a file
    if (!bRunning) {
        return;
    }

    // If a communication error occurs, the S7Variable transfers
    // the text "S7Error" at the start of the new PCE value
    if (evt.getNewValue().toString().startsWith("S7Error")) {
        // Display error message in the status bar
        labMsg2.setText(evt.getNewValue().toString());
        return;
    }
    // Form current file name from the entered file prefix, the
    // current file index, and the file extension
    int iMaxCounter = Integer.parseInt(tfMaxCounter.getText());
    String strFilno = "00000000" + iCounter;
    // Calculate maximum number of places required for file index
    int iMaxCntWidth = (int)Math.ceil(Math.log(iMaxCounter) /
        Math.log(10.0));
    strFilno = strFilno.substring(strFilno.length() - iMaxCntWidth);
    String strFilename = tfPrefix.getText() + strFilno + "." +
        tfExtension.getText();
    // Open file with the file name just obtained for write access
    // and write the current data of the S7Variable as a byte array in
    // binary format to the open file
    try {
        File file = new File(strFilename);
        FileOutputStream out = new FileOutputStream(file);
        out.write(s7Variable.getDataObjectAsByteArray());
        out.flush();
        out.close();
    } catch (Exception e) {
        // If a file error occurs, display the exception in the
        // status bar
        labMsg2.setText(e.toString());
        return;
    }
    // Display success in the status bar
    labMsg2.setText("Data logged into file " + strFilename);

    // Reserve file index for next run
    iCounter++;
    if (iCounter >= iMaxCounter) {
        iCounter = 0;
    }
}

/**
```

```
    * Main function of the application
    *
    * @param args array with the command line arguments
    */
   public static void main(java.lang.String[] args) {
      /* Create main window */
      LogfileDemo aLogfileDemo = new LogfileDemo();
      /* Register event handling for "close window" */
      aLogfileDemo.addWindowListener(new WindowAdapter() {
         public void windowClosed(WindowEvent e) {
            System.exit(0);
         };
      });
      // Display main window
      aLogfileDemo.setVisible(true);
   }
}
```

Compiling and Starting

You can import the Java source text into your Java development environment, compile it, and start it. Even without an IDE, you can compile and start the example with the native Java tools.

Assuming you have the JAR files of the S7BeansApi and the source files of the examples on your Windows PC in the folder C:\S7BeansApi and have installed the Java Development Kit (JDK) in the folder C:\jdk1.4.x", then you would make the following command line entry to compile the example:

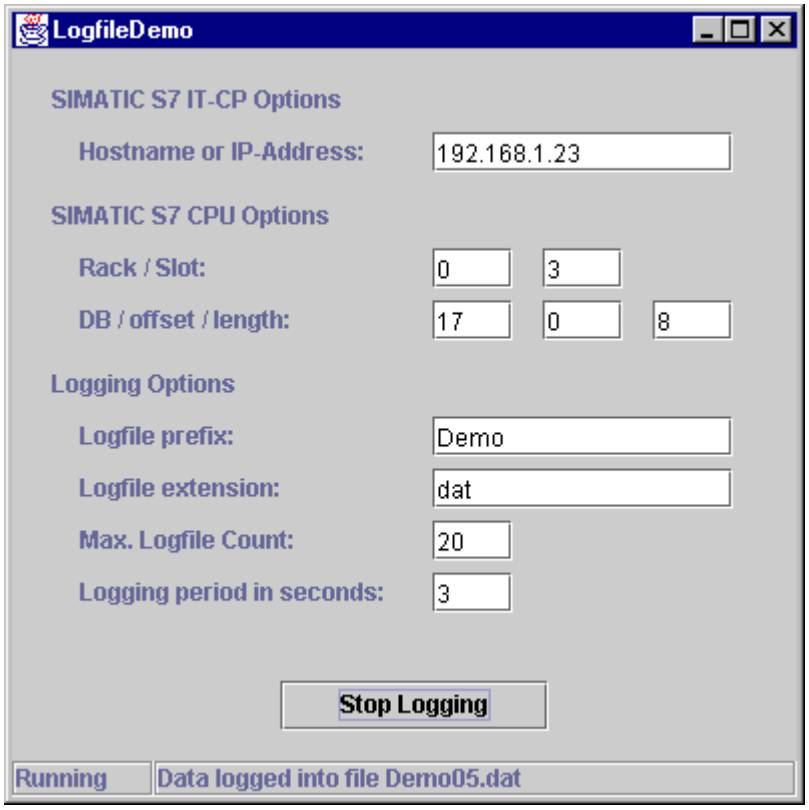
```
C:\S7BeansApi>C:\jdk1.4.x\bin\javac.exe-classpath
"C:\S7BeansApi\s7api.jar" -d "." "LogfileDemo.java"
```

The Java archive of the S7Beans "s7api.jar" must be specified in the class path since the example uses four beans from this library. The parameter sequence –d "." specifies that the class files displayed are displayed under the current path using the folder structure resulting from the package names. In the example, this would be the path "C:\ S7BeansApi\de\siemens\simaticnet\itcp\example".

If compilation was successful, the application can now be started using the following command line call:

```
C:\S7BeansApi>C:\jdk1.4.x\bin\java.exe-cp ".";"C:\S7BeansApi\s7api.jar"
de.siemens.simaticnet.itcp.example.LogfileDemo
```

Screenshot of the Started Java Application



In the situation shown in the screenshot, a SIMATIC S7 CPU in rack 0 slot 3 is being accessed via an IT–CP with IP address 192.168.1.23 and every 3 seconds, 8 bytes are be fetched from DB17 starting at offset 0. The data is written to files in the current folder using the scheme Demo00.dat – Demo19.dat.

To allow the example to run with these settings and without error messages, DB17 must exist and must be at least 8 bytes long.

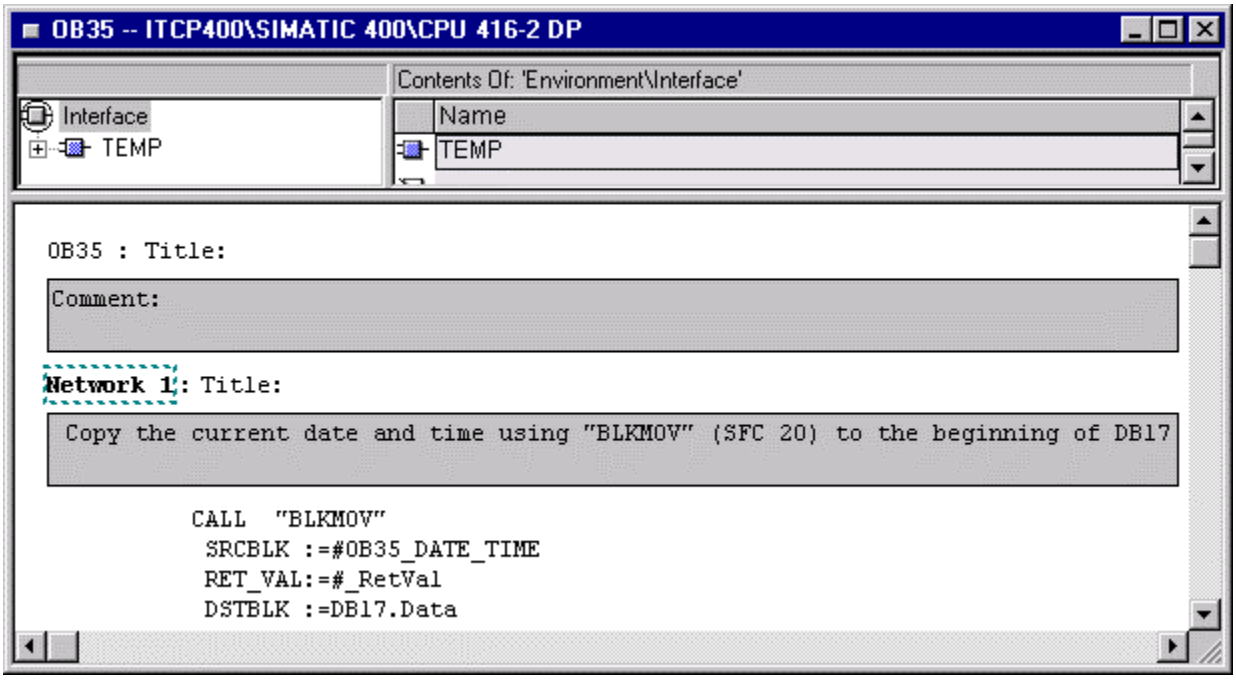
This requires the following SCL source that provides DB17 and copies the current CPU date and time of day into the DB every 100 milliseconds using the cyclic interrupt 5 (OB35). After compiling the source, uploading the blocks, and starting the Java application with the suitable parameters, the date and time of day of the CPU is written to a file on the PC in BCD code every three seconds.

Text of the STL Source

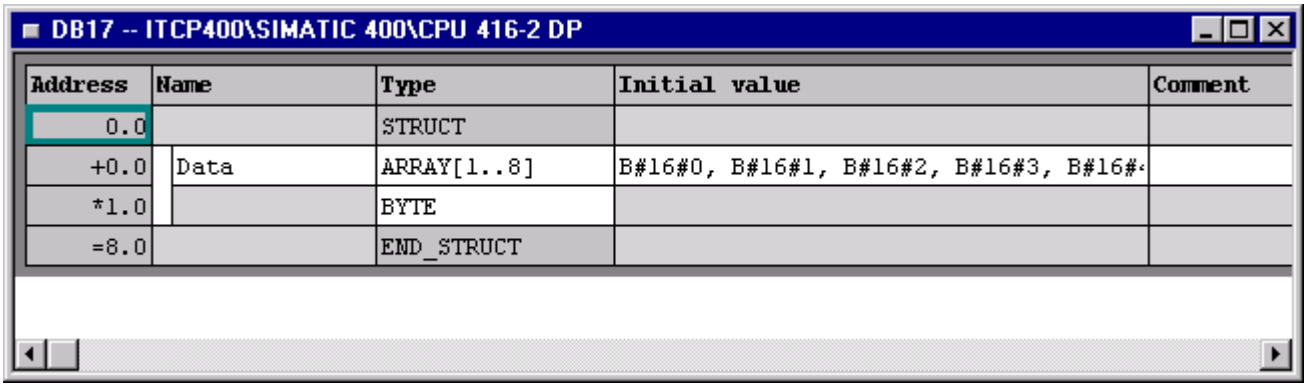
```
DATA_BLOCK DB 17

  STRUCT
   Data : ARRAY  [1 .. 8 ] OF
   BYTE  := B#16#0, B#16#1, B#16#2, B#16#3, B#16#4, B#16#5, B#16#6, B#16#7;
  END_STRUCT ;
BEGIN
```

```
END_DATA_BLOCK

ORGANIZATION_BLOCK OB35

VAR_TEMP
  OB35_EV_CLASS : BYTE ;    // Bits 0-3 = 1 (event entering state), bits 4-7 = 1
(event class 1)
  OB35_STRT_INF : BYTE ;    // 16#36 (OB35 has started)
  OB35_PRIORITY : BYTE ;    // Priority of OB execution
  OB35_OB_NUMBR : BYTE ;    // 35 (Organization block 35, OB35)
  OB35_RESERVED_1 : BYTE ;  // Reserved for system
  OB35_RESERVED_2 : BYTE ;  // Reserved for system
  OB35_PHS_OFFSET : INT ;   // Phase offset (integer, milliseconds)
  OB35_RESERVED_3 : INT ;   // Reserved for system
  OB35_EXC_FREQ : INT ;     // Frequency of execution (msec)
  OB35_DATE_TIME : DATE_AND_TIME ;  // Date and time OB35 started
  _RetVal : INT ;           // Return value for BLKMOV
END_VAR
BEGIN
// Copy the current date and time using "BLKMOV" (SFC 20) to the beginning of
DB17
      CALL SFC 20 (
           SRCBLK                  := #OB35_DATE_TIME,
           RET_VAL                 := _RetVal,
           DSTBLK                  := P#DB17.DBX0.0 BYTE 8);

END_ORGANIZATION_BLOCK
```

Screenshot of OB35



Screenshot of DB17



A Property Change Events – Overview

The following table shows which S7Beans expect which Property Change Events (PCE) and the meaning or type that *NewValue* will have. A ' * ' as PCE name means that the name will not be checked; in other words, all incoming PCEs or PCEs that were not previously filtered out are interpreted in the same way.

Table A-1

Receiver Bean	PCE Name	Type / Description of the Expected PCE Value (NewValue)
S7 Variable	inValue	Value to be set for the S7 from input bean e.g. CLTextIn
	*	Value to be set for the S7 (produces warning in Java console)
CLPipe	*	Value to be displayed by bean (value > 0 = full)
CLHPipe	*	Value to be displayed by bean (value > 0 = full)
CLVPipe	*	Value to be displayed by bean (value > 0 = full)
CLLevel	*	Value to be displayed by bean (numeric value)
CLTacho	*	Value to be displayed by bean (numeric value)
CLThermo	*	Value to be displayed by bean (numeric value)
CLValve	*	Value to be displayed by bean (value > 0 = open)
CLTextOut	*	Value or text to be displayed by bean
ConvertNumberSystem	*	Value to be converted by bean (decimal number)
COUNTER	*	Value to be converted and then passed The data type determines the direction: byte[] : S7Variable –> COUNTER –> output bean String: input bean –> COUNTER –> S7Variable
DATE *)	*	Value to be converted and then passed The data type determines the direction: Integer : S7Variable –> DATE –> output bean String: input bean –> DATE –> S7Variable
DATEandTIME *)	*	Value to be converted and then passed The data type determines the direction: byte[]: S7Variable –> DATEandTIME –> output bean String: input bean –> DATEandTIME –> S7Variable
TIME *)	*	Value to be converted and then passed The data type determines the direction: Long: S7Variable –> TIME –> output bean String: input bean –> TIME –> S7Variable

*) these S7 types are not supported on an S7–200.

The following table shows which S7Beans which PCEs they send to their listeners themselves and the meaning or type that *NewValue* will have.

Table A-2

Sender	PCE Name	Type / Description of the Sent PCE Value (NewValue)
S7CP	S7CP	Object of the type S7CP
	slot	Slot as integer (on S7–200 always value=0)
	rack	Rack as integer (on S7–200 always value=0)
	host	Host name as string
	s7NetAddress	Object of the type S7NetAddress
	state	Status as integer
	identification	Identification as string
	moduleName	Module name as string
S7 Device	S7 Device	Object of the type S7Device
	slot	Slot as integer (on S7–200 always value=0)
	rack	Rack as integer (on S7–200 always value=0)
	host	Host name as string
	s7NetAddress	Object of the type S7NetAddress
	state	Status as integer
	identification	Identification as string
	moduleName	Module name as string
S7 Variable	[Variable_name]	The new value of the variable after a “GET”. Note: The variable name is assigned individually by the user during configuration of the S7Variable bean
CLTextIn	inValue	String newInValue
ConvertNumberSystem	ConvertNumberSystem	Converted string depending on setting, for example, as hex or octal string
COUNTER *)	COUNTER	Converted string
DATE *)	DATE	Converted string
DATEandTIME *)	DATEandTIME	Converted string
S5TIME *)	S5TIME	Converted string
TIME *)	TIME	Converted string

Table A-2 , continued

Sender	PCE Name	Type / Description of the Sent PCE Value (NewValue)
TIMEofDAY *)	TIMEofDAY	Converted string
TIMER	TIMER	Converted string

*) these S7 types are not supported on an S7–200.

❑

B Further Information / FAQs

B.1 Lack of Resources in Netscape 4.x

Question

After calling up my HTML pages with the applets developed on the basis of S7BeansAPI Applets, the browser gets slower and slower, uses more and more system resources and at some point even crashes. In some situations, even my operating system is involved. How can I prevent this?

Answer

To avoid resource problems when using the applets developed with the S7 beans, all resources should be released when the applet is exited and all threads stopped. Since the S7BeansAPI uses static resources and threads internally for communication with the IT–CP, as of version V2.3 the terminate() method was created in the S7Api class to release these resources and to stop all threads.

Terminate() can normally be called after releasing all resources in the destroy() method of the applet. Since some browsers (for example, Netscape) do not call the destroy() method immediately when the HTML page is exited, but only when the applet is removed from the History cache of the browser, it is advisable to release the resources in the stop() method of the applet and to call terminate() from the S7Api class immediately. This does, however, require reinitializing all resources in the start() method of the applet because the applet, for example under Netscape, is also stopped and started again by changing the browser window size.

The combination of MS Windows NT 4.0 or Windows 2000/XP and Internet Explorer 5.5 or 6.0 has proved to be the most stable combination. When using Windows 95 or 98 or Windows ME, resource problems in the Browser can soon lead to problems in the operating system. When changing HTML pages with lots of applets quickly, the use of the Netscape browser is not recommended because it allocates system resouces more quickly than they are released.

Example

Below, there is a printout of the example introduced in Section 6.1.1 adapted for use with Netscape 4.x to avoid the problems described above.

```
package de.siemens.simaticnet.itcp.example;

import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.beans.*;
```

```
import de.siemens.simaticnet.itcp.api.*;
import de.siemens.simaticnet.itcp.gui.*;

/**
 * @author       ITCP Team
 */
public class Example1Net4x extends Applet implements PropertyChangeListener, ActionListener {
        private CLTextOut cLTextOut1 = null;
        private CLTimer cLTimer1 = null;
        private S7CP s7CP1 = null;
        private S7Device s7Device1 = null;
        private S7Variable s7Variable1 = null;
        private boolean isInit = false;
// flag to indicate whether or not init was done

        /**
         * Initializes the applet.
         *
         * @see #start
         * @see #stop
         * @see #destroy
         */
        public void init() {
                super.init();
                setName("VariantA");
                setLayout(null);
                setSize(426, 240);

                s7CP1 = new S7CP();
                s7CP1.setHostString(new HostString ("192.168.1.23"));

                s7Device1 = new S7Device();
                s7Device1.setSlot(3);

                s7Variable1 = new S7Variable();
                s7Variable1.setS7Anypointer(new S7Anypointer((int)2, (int)1, (int)131,
                (int)0, (int)10, (int)0));
                s7Variable1.setVariableName("s7Variable1");

                cLTimer1 = new CLTimer();
                cLTimer1.setDelay(2000);

                cLTextOut1 = new CLTextOut();
                cLTextOut1.setName("cLTextOut1");
                cLTextOut1.setBounds(0, 0, 200, 45);
                cLTextOut1.setOutFieldSize(100);
                cLTextOut1.setLabel("Value:");
                cLTextOut1.setUnit("");
                add(cLTextOut1, cLTextOut1.getName());

                /* init connections */
                s7CP1.addPropertyChangeListener(this);
                s7Device1.addPropertyChangeListener(this);
                cLTimer1.addActionListener(this);
                s7Variable1.addPropertyChangeListener(this);

                // initialization is valid
                isInit = true;

        }

        /**
         * Is called to start the applet. This method never needs be called directly.
         * It is called when the applet document is accessed.
         * @see #init
         * @see #stop
```

```
         * @see #destroy
         */
        public void start() {
                super.start();

                // Is an init call necessary?

                if (!isInit)
                        init();
// applet was stopped earlier, so call init again
        }
        /**
         * Is called to stop the applet.
         * Is called when the applet document is no longer
         * displayed. Is always called before destroy()
         * is called. This method must never be called directly.
         * @see #init
         * @see #start
         * @see #destroy
         */
        public void stop() {
                super.stop();

                // remove all components from this container
                removeAll();

                // destroy all used resources here
                cLTextOut1 = null;
                cLTimer1 = null;
                s7CP1 = null;
                s7Device1 = null;
                s7Variable1 = null;

                // remember that initialization was not done
                isInit = false;

                // terminate the api mechanisms, finalize objects
                // and run the Garbage Collector
                S7Api.terminate();
        }
        /**
         * Cleans up whatever resources are being held. If the applet is active
         * it is stopped.
         *
         * @see #init
         * @see #start
         * @see #stop
         */
        public void destroy() {
                super.destroy();

                // clean up now in stop method !
        }

        /**
         * Method to handle events for the PropertyChangeListener interface.
         * @param evt PropertyChangeEvent
         */
        public void propertyChange(PropertyChangeEvent pce) {
                if (pce.getSource() == s7CP1)
                        s7Device1.propertyChange(pce);
                if (pce.getSource() == s7Device1)
                        s7Variable1.propertyChange(pce);
                if (pce.getSource() == s7Variable1)
                        cLTextOut1.propertyChange(pce);
        }
```

```
        /**
         * Method to handle events for the ActionListener interface.
         * @param e java.awt.event.ActionEvent
         */
        public void actionPerformed(ActionEvent ae) {
                if (ae.getSource() == cLTimer1)
                        s7Variable1.processGet();
        }
}
```

C Comparison of S7 Types and Java Types

S7 types generally differ from the corresponding Java types in the range of values. To avoid problems here with signs etc., the corresponding types are as shown in the following table. If you use arrays (repetition factor (n) > 1), the elementary data types are used in Java. Such an array (for example, int[]) is considered as an object by Java which is not the case with other simple data types. This is particularly important for us for parameter passing, for example for setValue() of S7Variable.

Table C-1

S7 Type	Java Type	Java Array
BOOL	Boolean	–
BYTE	Integer	int[]
CHAR	Character	char[]
WORD	Integer	int[]
INT	Integer	int[]
DWORD	Long	long[]
DINT	Long	long[]
REAL	Float	float[]
DATE *)	Integer	int[]
Time Of Day *)	Long	long[]
TIME *)	Long	long[]
S5TIME *)	byte[2]	byte[n*2]
Date And Time	byte[8]	byte[n*8]
STRING	String	–
COUNTER *)	byte[2]	byte[n*2]
TIMER	byte[2]	byte[n*2]

*) these S7 types are not supported on an S7–200.

Notice

– An array of the type BOOL cannot be formed.

– An array of the type STRING cannot be formed.

D References

/1/ SIMATIC NET CP Manual
Description of Handling the Device and Installation
SIEMENS AG

/2/ NCM S7 for Industrial Ethernet Manual
Part
– of the documentation package NCM S7 for Industrial Ethernet
– of the online documentation in the STEP 7 optional package NCM S7 for Industrial Ethernet
SIEMENS AG

/3/ Information Technology in SIMATIC S7 with CP 343–1 IT and CP 443–1 IT
Part of
– the NCM S7 for Industrial Ethernet manual package
– the online documentation in STEP 7 / Option NCM S7 for Industrial Ethernet
Siemens AG

/4/ CP 243–1 IT manual
Communications Processor for Industrial Ethernet and Information Technology
Configuration / Programming
Siemens AG

/5/ Programming Tips for S7 Beans (for Visual Age)
Siemens AG
can be obtained from the Web

Order Numbers

The order numbers for the SIEMENS documentation listed above can be found in the catalogs ”SIMATIC NET Industrial Communication, Catalog IK10” and ”SIMATIC Programmable Controllers SIMATIC S7 / M7 / C7 – Components for Fully Integrated Automation, Catalog ST70”.

You can obtain these catalogs and any other information you require from your local SIEMENS branch and national subsidiary

Some of the documents listed here are also on the Manual Collection CD supplied with every S7–CP.

Further recommended reading on the topics Internet/Web, HTML, Java

/6/ Web–Publishing with HTML 4
Deborah S. Ray / Eric J. Ray
Sybex Verlag 1998

/7/ Durchblick im Netz
Vom PC–LAN zum Internet
Kauffels, F–J.
Internat. Thomson Publ., 1998
ISBN 3–8266–0413–X

/8/ Campione/ Walrat
The JavaTM Tutorial
Second Edition
Object–Oriented Programming for the Internet
ADDISON–WESLEY, 1998
ISBN 0–201–31007–4

In the meantime, there are numerous books available with which you can learn Java; we recommend:

/9/ Java in 21 Days
by Laura Lemay and Charles L. Perkins
ISBN: 3827255783

/10/ Java in a Nutshell
by David Flanagan
ISBN: 3897211009

/11/ Java Examples in a Nutshell
by David Flanagan
ISBN: 3897211122
(for getting to know Java quickly and gaining programming experience)

❏

T

U

W