View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Lund University Publications - Student Papers

ISSN 0280-5316
ISRN LUTFD2/TFRT--5678--SE

Generic Web Server in Embedded Control Systems

Andreas Ekstrand
Jonas Ludvigsson

Department of Automatic Control
Lund Institute of Technology
November 2001

Department of Automatic Control
Lund Institute of Technology
Box 118
SE-221 00 Lund Sweden

Document name
MASTER THESES

Date of issue
November 2001

Document Number
ISRN LUTFD2/TFR--5678--SE

Author(s)
Andreas Ekstrand
Jonas Ludvigsson

Supervisor
André Ekfeldt, ABB
Karl Erik Årzén, LTH

Sponsoring organization

Title and subtitle
Generic Web Server in Embedded Control Systems. (Webserver i inbyggt styrsystem).

Abstract
The goal for this master thesis was to investigate the possibility of using an existing web server and modify it so it can be used in ABB controllers.

The work has resulted in two different versions of a web server. The basic functionality is the same for both versions. The web servers are able to present a number of controller-related information. For example: hardware, which control programs that are executing in the controller, information about the firmware, and also system information like heap memory usage. It can also be used for changing different parameters.

The two versions use different techniques for presenting the information. They are also implemented in different ways. The first version has a more user-friendly interface but requires more memory, 71 kb. The second version requires less memory, 48 kb, but does not have the same user-friendly interface.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
68

Recipient's notes

Security classification

The report may be ordered from the Department of Automatic Control or borrowed through:
University Library 2, Box 3, SE-221 00 Lund, Sweden
Fax +46 46 222 44 22 E-mail ub2@ub2.se

Generic web server in embedded control systems.

2001-11-14

Andreas Ekstrand
Jonas Ludvigsson

Table of contents

1 Introduction ....
1.1 Outline of the report 5

2 System overview ....
2.1 The idea of a web server 6
2.2 Control^{IT} 6
2.3 Control system overview 8
2.4 Operating system 10
2.5 Web server in the control system 11
2.6 Summary 11

3 Internet technologies ....
3.1 HTTP Protocol 12
3.1.1 Request and response 12
3.2 URL 13
3.3 Web server 14
3.3.1 How does it work? 14
3.3.2 Embedded web server 14
3.4 Summary 14

4 Markup languages and SOAP ....
4.1 HTML 15
4.1.1 Advantages and limitations 16
4.2 XML 16
4.2.1 XML vs. HTML 16
4.2.2 The XML Document 18
4.3 XSL 20
4.3.1 XSLT 20
4.3.2 XSL-FO 22
4.4 SOAP 22
4.4.1 How does it work? 22
4.4.2 SOAP Message 23
4.4.3 A SOAP example 24
4.5 Summary 24

5 Scripts ....
5.1 CGI script 25
5.1.1 Why CGI? 25
5.1.2 Functionality 25
5.1.3 Common use 26
5.1.4 Limitations 26
5.2 JavaScript 26
5.2.1 Why JavaScript? 26
5.2.2 Functionality 27
5.3 Summary 28

6 Web server ....
6.1 Different web servers 29
6.2 GoAhead® WebServer™ 29
6.2.1 Features and limitations 29
6.2.2 GoForms 30
6.2.3 Modifications 30

6.3 Generic web server ... 31
6.4 Real-time properties and memory handling ... 32
6.4.1 Real-time properties ... 32
6.4.2 Memory handling ... 33
6.5 Summary ... 34

7 The program ...
7.1 Overview of the program ... 35
7.2 Common techniques ... 36
7.2.1 Initialization ... 36
7.2.2 Generating and sending web pages ... 37
7.3 Start page ... 42
7.3.1 JavaScript version ... 43
7.3.2 GoForms version ... 45
7.4 Access variables ... 46
7.4.1 Implementation ... 46
7.5 Hardware ... 47
7.5.1 Implementation ... 47
7.5.2 Firmware ... 49
7.5.3 Ethernet ... 51
7.6 Applications ... 53
7.6.1 Implementation ... 53
7.7 Miscellaneous ... 54
7.7.1 Heap information ... 54
7.7.2 Controller log ... 55
7.8 Summary ... 55

8 Future developments ...

9 Summary and conclusions ...

10 User guide ...
10.1 Tutorial ... 58

11 References ...

Definitions and abbreviations

AC	ABB Controller.
AC 800M	ABB's most powerful controller, known as "Common controller"
AC 800C	ABB Controller suitable for minor applications.
AC 250	Flexible and modular ABB controller.
ASP	Active Server Page. A Microsoft technique to execute scripts in a web page.
CGI	Common Gateway Interface.
Control^{IT}	ABB's name for a family ABB developed controllers and equipment.
Control Builder	Programming tool in Control^{IT}.
ControlNet	A high-performance network for industrial applications. It is a standard (developed by Allen-Bradley) that is used for fast communication between control systems and distributed I/O units.
DTD	Document Type Definition. Used in XML.
Ethernet	Network cabling system according to IEE 802.3
FBD	Function Block Diagram. An 1131 language.
GML	Generalized Markup Language.
HTML	Hypertext Markup Language.
HTTP	Hypertext Transfer Protocol.
IEC 61131-3	A family of programming languages, usually only called 1131.
I/O	Input/Output.
IL	Instruction List. An 1131 language.
LD	Ladder Diagram. An 1131 language.
PDA	Personal Digital Assistant.
PROFIBUS	A fieldbus originally developed by Siemens.
pSOSystem	Real-time operating system.
PPP	Point-to-Point Protocol, serial communication.
RS232	A serial communication protocol.
RPC	Remote Procedure Calls.
RTOS	Real-time operating system.
SattBus	A control network. SattBus communication can be used in a Soft Controller or an AC 250.
SFC	Sequential Function Chart. An 1131 language.
SGML	Standard Generalized Markup Language.
SLC	Satt Line Control
SOAP	Simple Object Access Protocol.
Soft Controller	PC based ABB controller.
ST	Structured Text. An 1131 language.
TCP/IP	Transmission Control Protocol/Internet Protocol.
URI	Uniform Resource Identifier.
URL	Uniform Resource Locator.
XML	eXtensible Markup Language.
XSL	eXtensible Stylesheet Language.

1 Introduction

The use of web servers in control systems opens new doors in maintenance and supervision of industrial processes and machines. Because of the well-known interface the web server becomes a helpful tool for operators or engineers.

The goal for this master thesis was to implement a generic web server in an embedded control system. The work has been done in cooperation with ABB Automation Products AB Malmö Sweden. The idea was not to implement a web server from scratch, rather to investigate the possibility of using an existing web server and modify it. A number of different web servers have been investigated. One of them filled the requirements for implementation in the ABB Control Software.
Different techniques for how to create web pages and how to interact with the web server have been investigated and a prototype program has been implemented.

1.1 Outline of the report

Chapter 2 describes the idea of a web server in an embedded control system and also gives an overview of the control system, its hardware and software. Chapter 3 gives an overview of some common Internet techniques like the HTTP protocol, the URL and how a web server works. In Chapter 4 three different markup languages are described, HTML, XML and XSL. An application of XML, SOAP, is also described. Chapter 5 gives a description about how to interact with a web server using CGI scripts and JavaScript.
Chapter 6 contains a more detailed description about the web server used in the implementation and some real-time properties and memory handling. Chapter 7 describes the implementation of the web server program. Future developments are discussed in Chapter 8. Chapter 9 contains a summary of this thesis and some conclusions. The last chapter is a user guide that includes a tutorial on how to access and use the web server.

2 System overview

This chapter gives the background to the idea behind the web server and where it is intended to execute. A system overview, which gives a brief insight to ABB's Control^{IT}, is presented. The hardware, the operating system, and tools for communication are also discussed.

2.1 The idea of a web server

Almost all industrial processes are controlled in some way and they are often located in noisy environments. Common work with the machine, like supervision, controller tuning and error correction are done via some kind of connected I/O. If these tasks could be performed in a calmer environment, like an office, they would be handled faster and perhaps better. Intranet exists in nearly all plants nowadays. If a web server is implemented in the machine and the machine is connected to the intranet any computer on the intranet could get in contact with the machine. Of course the computer must contain a web browser, but that is not a problem today, they can be downloaded free from the Internet. A web browser is an easy and well-known interface and they can even run on handheld computers, PDA's. It would also be possible to reach the machine trough the Internet and then, for instance, machine maintenance could be performed remotely and it would quite simply be easier to get in contact with the machine. The connected terminals would become unnecessary and the cost for these would disappear.

2.2 Control^{IT}

Control^{IT} is ABB's name for a family of ABB developed controllers, programming and configuration tools. Here follows a brief description of some of the parts in Control^{IT} [1].

Control Builder, see Figure 2.1, is a programming tool for configuration and programming of the ABB controllers AC 800M, AC 800C, AC 250 and Soft Controller. For further description of the controllers, see Section 2.3. With Control Builder follows a set of predefined functions. These include data types, functions, function blocks and control modules that can be used in the programs. The programs, called applications, can be written in five different IEC 61131-3 (usually only called 1131) programming languages. The five languages are FBD, ST, IL, LD and SFC. The controllers are the targets for the programs developed with Control Builder. It is to such a unit that the program code will be downloaded and executed.

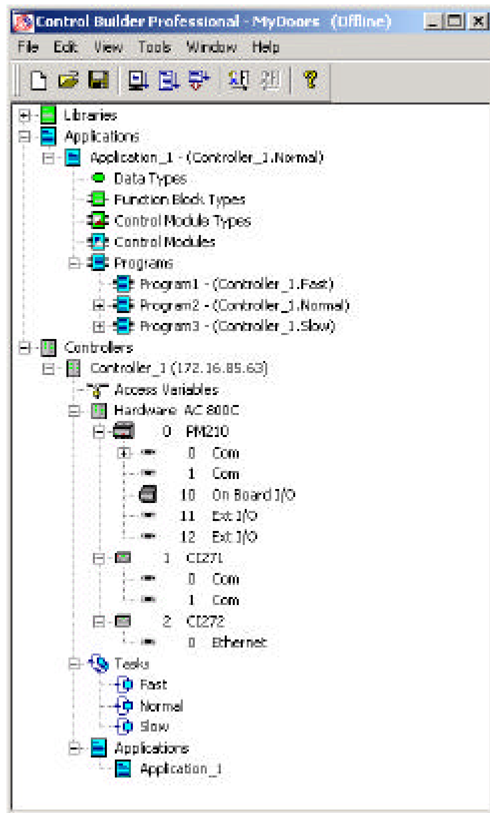


Figure 2.1 Example of a project in Control Builder.

The programs created in the Control Builder can be part of more or less advanced control systems. In the most trivial case a PC running Control Builder is connected to a controller, see Figure 2.2.

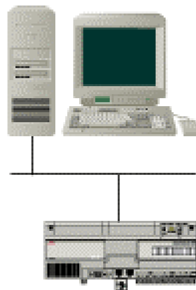


Figure 2.2 A PC running Control Builder and a controller connected via Ethernet.

In more advanced control systems, like in Figure 2.3, Control Builder can run on many computers. This makes it possible to share the project between developers. The project can be distributed to a number of controllers, which results in that each controller can execute a part of the project.

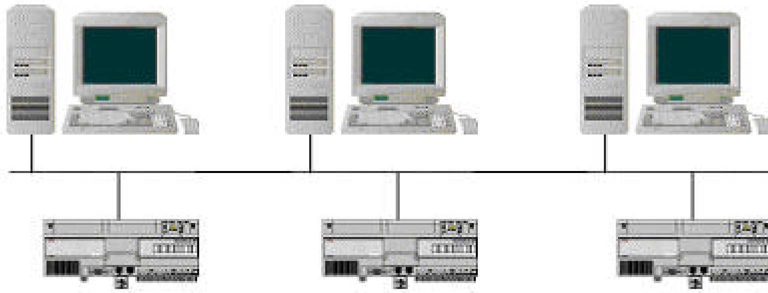


Figure 2.3 Several PC's running .

2.3 Control system overview

The controllers AC 800M, AC 800C and AC 250 consist of a controller unit, a power supply unit, and a number of connections where it is possible to add communication interfaces.
The controller unit contains for example a processor (Motorola manufactured), RAM-memory and FLASH-memory. The Soft Controller is a PC based controller. Here follows a brief description of the controllers, for technical data see Table 2.1.

AC 800M, see Figure 2.1, is the most modern and powerful controller. It contains much memory and is suitable for large applications. Twelve local I/O-modules are available and twelve communication interfaces can be added.

Figure 2.1 The AC 800M controller.

AC 800C, see Figure 2.2, is a compact controller suitable for minor applications. It contains less memory than AC800 M. An on-board I/O with ten digital inputs and six digital outputs is built in. Two communication interfaces can be added.

Figure 2.2 The AC 800C controller.

AC 250, see Figure 2.3, is an older controller. It contains various CPUs and memory. Its strength is the modularity and flexibility. It can be suited to fit the desired system configuration.

Figure 2.3 The AC 250 controller.

Soft Controller is a PC based controller. High real-time performance can be achieved. How large applications it can execute depend on the PC's internal memory.

	Processor type	Clock frequency	Performance indicator**	RAM memory	FLASH memory
AC 800M	MPC860	48 MHz	0.3 ms	8 MB	2 MB
AC 800C	MC68332	18.432 MHz	2.2 ms	2 MB	2 MB
AC 250*	MC68020	16.7 MHz	***	2 MB	-

Table 2.1 Technical data for AC 800M, AC 800C and AC 250. *Example of one configuration. **Execution time/1000 lines of program code, Boolean operations. ***No data available.

The controllers need software to work. The software consists of two parts, the basic part called firmware and the application. The firmware is downloaded to the controller via Ethernet or serial connections, see Figure 2.4. Firmware is a program that inserted into read-only-memory becomes a part of a computing device. The firmware consists for example of drivers, different protocols, hardware configurations, and runtime environment for execution of the applications. The application created in the Control Builder is also downloaded.

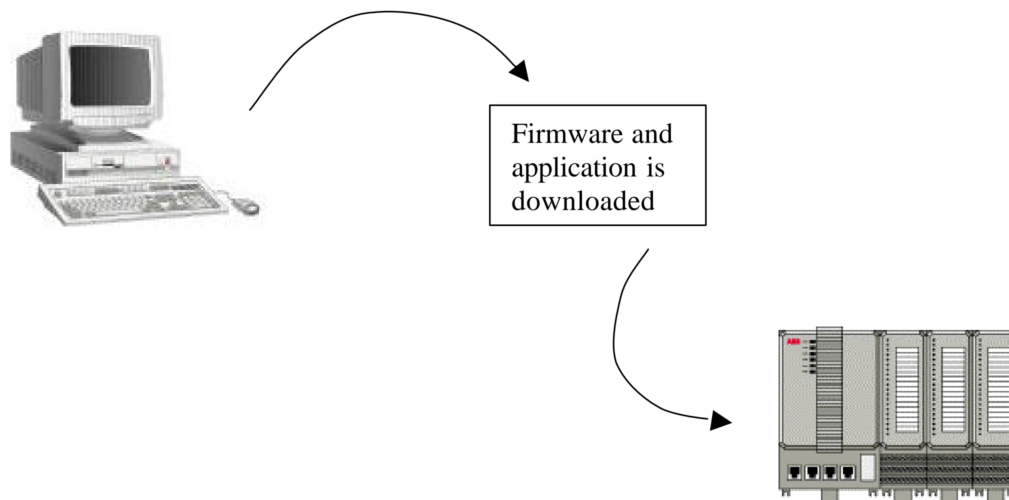


Figure 2.4 The firmware and the application is downloaded to the controller via Ethernet or a serial connection.

The communication interfaces that can be added makes it possible for the controllers to communicate with other units via RS232, Ethernet, SattBus, ControlNet or PROFIBUS.

In Figure 2.5 a control system is shown. There are four controllers, one AC 800M with two communication interfaces and six local I/O, two AC 800C with Ethernet interfaces, and one AC 800C with Serial interface. Via PROFIBUS remote I/O's are connected to the AC 800M. A PC is connected to the control system via an Ethernet connection.

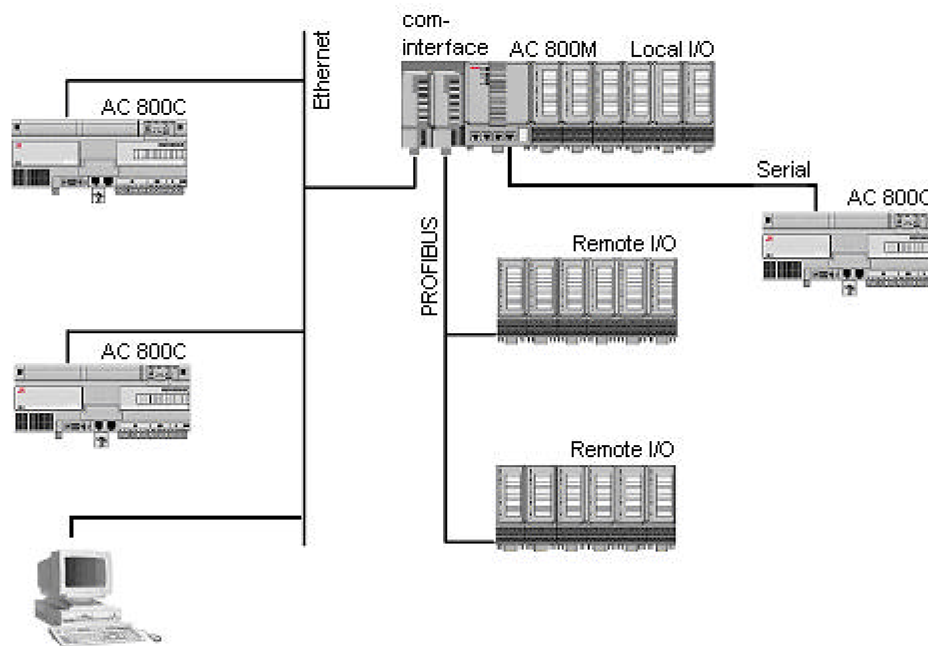


Figure 2.5 A control system with one AC 800M controller, three AC 800C controllers, remote I/O's and a PC. They are connected via different communication units.

2.4 Operating system

In the controller runs the real-time operating system, pSOS. It is developed by Integrated Systems Inc. pSOS is a modular, high-performance real-time operating system designed for embedded systems. It is built around the pSOS+ real-time multi-tasking kernel [2].

pSOS consists of the following components (see also Figure 2.1):

- **pSOS+ Real-time Multitasking Kernel.**
- **pSOS+m Multiprocessor Multitasking Kernel.** Extends the pSOS+ feature set to operate across multiple, tightly coupled or distributed processors.
- **pNA+ TCP/IP Network Manager.** A complete TCP/IP implementation including gateway routing, UDP (User Datagram Protocol), ARP (Address Resolution Protocol) and ICMP (Internet Control Message Protocol).
- **pRPC+ Remote Procedure Call Library.** Offers SUN-compatible Remote Procedure Call (RPC) services.
- **pHILE+ File System Manager.** Gives efficient access to mass storage devices, both local and on a network. Includes support for CD-ROM devices, MS-DOS compatible floppy disks and a high-speed proprietary file system.
- **pREPC+ ANSI C Standard Library.** Provides familiar ANSI C run-time functions such as `printf()` in the target environment.

- **pROBE+ Debugger.** System-level debugger, and (optional) high-level debugger. The high-level debugger executes on the host computer and works in conjunction with the pROBE+ system-level debugger, which runs on the target system.

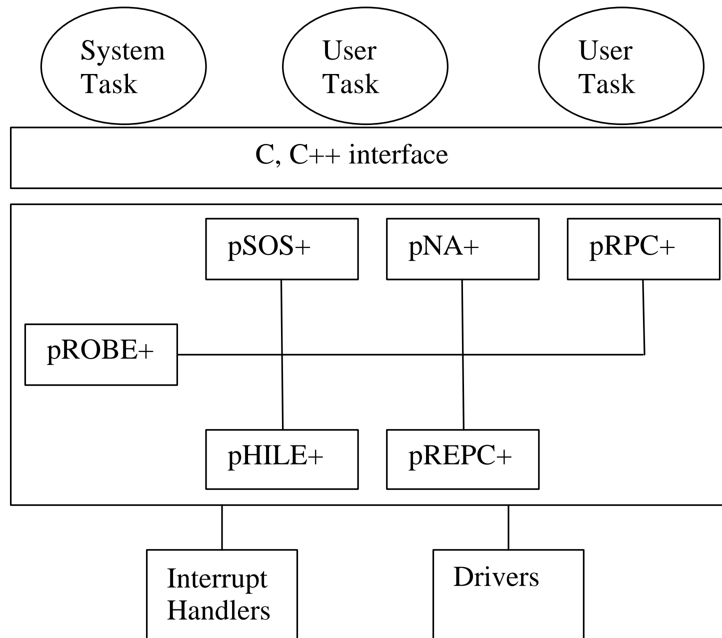


Figure 2.1 The pSOS environment

2.5 Web server in the control system

The web server code is part of the firmware and is downloaded to the controller. From a web browser it is possible to get information about the control system. The information can be: configuration, connected communication interfaces, status for different communication protocols, and downloaded firmware. It is also possible to read and change a value of an 1131 application variable. This is possible if the variable is declared as an access variable. Access variables are accessible for all units on the control network. The control programs are much more important than the web server. Therefore the web server program shall not use much CPU resources. The web server should be small, that is, it should not require much memory, and its priority should be low. The web server is intended to run in the controllers AC 800M and AC 800C.

2.6 Summary

Control^{IT} is ABB's name for a family of ABB developed controllers and equipment. The operating system pSOS runs in the controllers. A web server in Control^{IT} would make supervision of controllers easier and faster. The web server code is part of the firmware. With a web browser information about the controller can be viewed.

3 Internet technologies

This chapter gives an overview over some common Internet technologies: the HTTP protocol and URL. The HTTP protocol is the most used protocol for the World Wide Web transactions. The URL is the way of addressing files on the Internet. A brief description of the web server functionality is also included.

3.1 HTTP Protocol

The Hypertext Transfer Protocol (HTTP) is a set of rules for transferring files (text, images, sound, video etc.) over a network (Internet or a local network). The HTTP protocol uses TCP/IP for connections between clients and servers [3].

HTTP is a simple protocol. The clients establish a TCP connection to the server, sends a request, and reads back the server's response. After the transfer is complete the connection will close.

Today there are two versions of the HTTP protocol, 1.0 and 1.1. The difference between them is that 1.1 tries to minimize the number of TCP connections. Instead of opening and closing a new connection for every object on a web site (picture, sound clips) the requests are buffered and sent together. This results in less Internet traffic and improved performance for the user.

3.1.1 Request and response

There are two types of HTTP messages: request and response. The request is sent from a client to a server and responses are sent from server to client.
The format of a HTTP *request* is:

```
request-line
headers (zero or more)
<blank line>
body
```

The format for a HTTP *response* is:

```
status-line
headers (zero or more)
<blank line>
body
```

Request line
The format of the request line is:

```
request request-URI HTTP-version
```

There are three kinds of requests: `GET`, `HEAD`, and `POST`. The `GET` request returns the information that is identified by the `request-URI` (Uniform Resource Identifier). The `HEAD` request is similar to the `GET` request, but only the server's header information is returned. Finally the `POST` request is used for posting email or sending in forms that can be filled in by the user.
An example of a request line:

```
GET /main.html HTTP/1.1
```

Status line
The format of the status line is:

```
HTTP-version response-code response-phrase
```

The `response-code` is a 3-digit numeric response code and the `response-phrase` is a human-readable answer. The response codes are divided into four categories.

```
2xx:  Success: e.g. ("HTTP/1.1 200 OK")
3xx:  Redirection – Further action by the user is needed
4xx:  Client Error: e.g. ("HTTP/1.1 404 Not found")
5xx:  Server Error: e.g. ("HTTP/1.1 503 Service temporarily
      unavaible")
```

Headers
Both the request and the response can contain a number of header fields. The header fields contain some additional information about the request, response and the body.
A header has the following format:

```
field name: field value
```

Examples of request header fields:

- `From: mailbox`
  An Internet e-mail address
- `If-Modified-Since: HTTP-date`
  On the form: day "," date month year hh:mm:ss "GMT"

Response header fields:

- `Date: HTTP-date`
  On the form: day "," date month year hh:mm:ss "GMT"
- `Server: server-software`
  Information about the server program and version

Body header fields:

- `Content-Type: type/subtype`
  Specified the data type of the body, e.g text/html, text/xml, image/gif etc.
- `Content-length: Content-length`
  Specifies the size in bytes of the body.

3.2 URL

The Uniform Resource Locator (URL) is the address of a file on the Internet. The address starts with a protocol specification (e.g. http:// or ftp://), next comes the host name that usually starts with "www" and ending with the domain (.com, .se), after that the path and finally the file name.

```
http://www.hostname.domain/path/file
```

The host name is just a more user-friendly way of presenting the server name. Every server on the Internet has a corresponding IP-number. In some special server called "name servers" there are tables with the host names and the corresponding IP-number.

3.3 Web server

3.3.1 How does it work?

A web server is basically a program that can respond to requests from web browsers. The server “listens” to a port (usually port 80). Using HTTP a web browser sends a request to the server (1) see Figure 3.1. The request contains the name and location of the wanted file (URL). The server receives the request and tries to interpret the request (2). If the web server can find the requested file it will return it to the web browser (3) otherwise an error message will be returned.

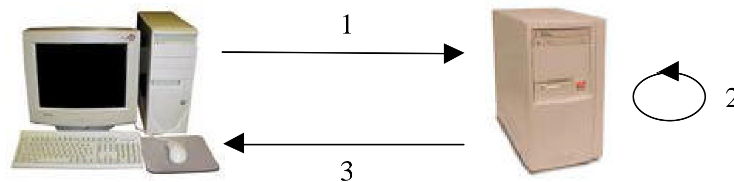


Figure 3.1 Request and response of a web page.

3.3.2 Embedded web server

An embedded web server is a web server that has been designed to be included in different kinds of small devices. In this project it is a controller, but it could also be a printer, an industrial machine or even a dishwasher. It is an easy and well-defined way for information access. When designing an embedded web server there are some requirements to take notice of. First of all it has to have a small memory footprint (>100 kb is quite much). Often the devices do not have a file system (hard drive), so it must be possible to store the web content on ROM or to have the web pages created dynamically (created when requested).

3.4 Summary

The HTTP protocol is a simple protocol that uses TCP/IP for the connections between clients and servers. There are two types of messages: requests and responses. A client sends a request to a server and the server sends back a response. The URL is used for retrieving a specific file over the Internet. A web server is simply a program that can respond to a request for a certain file and send it back to the caller. A special version of a web server is the embedded web server. It can be included in a device that is connected to a network. The main requirement on an embedded web server is the memory footprint, which cannot be too large.

4 Markup languages and SOAP

The first modern markup language was GML, Generalized Markup Language. It was intended to be a meta-language, a language that could be used to describe other languages, their grammar and vocabularies. GML later became SGML, Standard Generalized Markup Language. SGML is a very complex markup language, mostly used in large industries that process large volumes of data.

A markup language is a set of rules that describe how a text is to be presented or worked on. Everything in a document, except the text itself, is markup. This chapter gives a brief overview of the most common markup languages.

4.1 HTML

HTML, HyperText Markup Language, is derived from SGML. HTML is one of the pillars of the World Wide Web. It is a set of markup symbols, which inserted in a document make the document viewable by a web browser. Unlike e.g. Word file format, HTML is not a complex file format. Pictures and shape are for example stored in separate files. HTML stores describing instructions about what every fragment of information is, like main heading, tables, lists etc. It can look like this:

```
<html>
      <head>
            <title>Example</title>
      </head>
      <body>
            <h1>1 Header</h1>
                  Here comes the body text.
            <h2>1.1 A list</h2>
            <ul>
                  <li>First element</li>
                  <li>Second element</li>
            </ul>
      </body>
</html>
```

Some words are capsulated within “<” and “>”. These are called elements or tags and are instructions to the program, often a web browser, which is supposed to read and present the file. A start tag is a left angle bracket “<” followed by the instruction and then a right angle bracket “>”. The end tag looks like the start tag, but the left angle bracket is preceded by a slash “/”. The start- and end tag tell the presenting program (web browser) how to interpret the involving characters and how to present them. HTML is not case sensitive, the body tag can be written in many ways, <BODY>, <body> or <BoDy>. Here follows a description of the tags used in the example.

The `<html>` tag tells the web browser that this is a HyperText-document.

The `<head>` tag indicates the main header.

The `<title>` tag writes the title in the title bar.

The `<body>` tag indicates that here starts the content of the page.

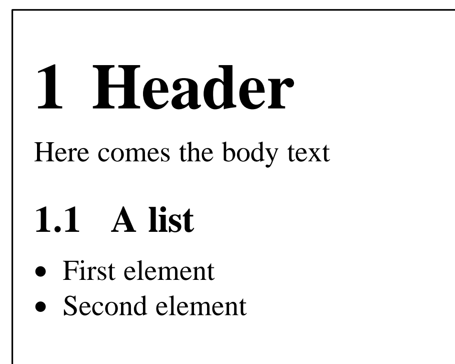
The <h1> instruction indicates that it is a header and Microsoft Internet Explorer interprets it as Times New Roman, 24 points, bold.

The <h2> instruction indicates that it is a header and Microsoft Internet Explorer interprets it as Times New Roman, 14 points, bold.

The <ul> tag means that here starts an unordered list.

For every <li> tag a new bullet (•) is created.

The example code presented in Microsoft Internet Explorer looks like this:



4.1.1 Advantages and limitations

It is easy to write HTML code and it is widely spread around the world. Other benefits are that it exists a lot of predefined tags and that all HTML documents have the same base parts. The HTML file format is not dependent on a certain operating system or program. If using a computer without a web browser it is still possible to read and understand a HTML document.

If the presented information is updated frequently, HTML is very limited. Instead a database is tied to a web page. The information is stored in the database and converted when a request for the web page occurs. If a web page includes a form, e.g. a login form, the user-entered data must be received and processed by the server. HTML cannot manage this, but CGI scripts can, see Section 5.1.

4.2 XML

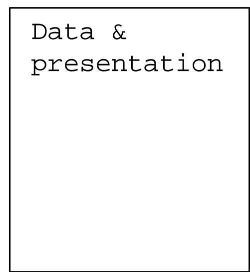
XML (eXtensible Markup Language) is subset of the SGML (Standard Generalized Markup Language) [4]. It is designed to make it easy to send and receive structured data over the Internet.

4.2.1 XML vs. HTML

In HTML both the content (the data) and the presentation of the content are described in one document. In XML the data and the presentation have been separated into two parts. The first part describes the data, the actual XML documents. The second part contains the presentation of the data; the so-called “stylesheets”, see Figure 4.1

An advantage with using a separate representation is that a single XML document can be presented in a number of ways. For example one stylesheet for printing, another for viewing on the screen and one when using a Personal Digital Assistant, PDA (e.g. PalmPilot).

HTML | **XML with stylesheet**



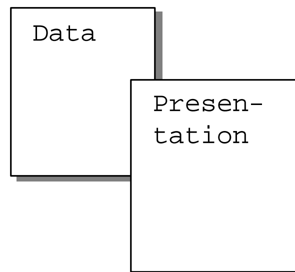


Figure 4.1 XML vs. HTML

Example

This short example shows some similarities and differences between HTML and XML. Example of HTML code for a simple table:

```
<TABLE BORDER=1>
      <TR>
            <TH>Product ID</TH>
            <TH>Description</TH>
            <TH>Price</TH>
      </TR>
      <TR>
            <TD>1234578-Q</TD>
            <TD>AC 800M</TD>
            <TD>$99.99</TD>
      </TR>
</TABLE>
```

This is how it would look in a web browser:

Product ID	Description	Price
1234578-Q	AC 800M	$99.99

In XML code the same thing could look like this:

```
<PRODUCT>
      <id>12345678-Q</id>
      <description>AC 800M</description>
      <price>$99.99</price>
</PRODUCT>
```

In a web browser it would look like:

```
<PRODUCT>
      <id>12345678-Q</id>
      <description>AC800M</description>
      <price>$99.99</price>
</PRODUCT>
```

As seen in the example XML has a clear advantage over HTML in describing the content of the document, but it lacks in presentation. This is because no style sheet was used.
Style sheets will be described later in Section 4.3.

Like HTML, XML uses "tags" for describing the content of the document. The tags are the second major difference between HTML and XML.
As shown in the chapter about HTML, there are several different tags used when building web sites in HTML. But unlike HTML, where all tags are predefined, XML allows programmers to specify their own tags. This means that XML is a meta-markup language.

4.2.2 The XML Document

The tree structure of a XML document looks like Figure 4.1 [5]:

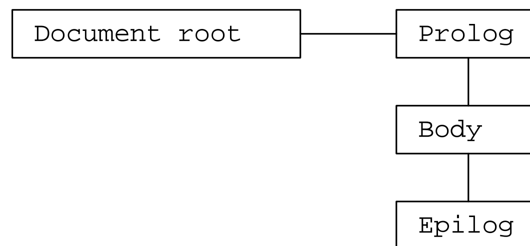


Figure 4.1 XML Document

Prolog

The prolog is used to signal the beginning of XML data. It contains the *XML Declaration*, *Document Type Declaration*, *Processing Instructions* (PIs) and perhaps some comments.
All parts of the prolog are optional.

XML Declaration
All XML documents *should* begin with an XML Declaration. In cases when the XML data use an encoding other than UTF-8 or UTF-16, the XML Declaration *must* be used. UTF-8 and UTF-16 are encoding rules that specify how characters should be coded, like ASCII or Unicode.

In XML 1.0 the declaration consists of three parameters:

- `version` This is required, and its value currently must be '1.0'.
- `encoding` This is optional, and its value must be a legal character encoding name, such as "UTF-8", "UTF-16" or "ISO-8859-1" (Latin-1). If this parameter is not included, UTF-8 or UTF-16 is assumed.
- `standalone` This is optional, its value must be "yes" or "no". If "yes" is used it means that the document itself contains all information needed for processing and displaying.

A typical XML Declaration:

```
<?xml version="1.0" encoding='ISO-8859-1' standalone="yes"?>
```

Document Type Declaration (DOCTYPE)
The Document Type *Declaration* (DOCTYPE) is used to link a Document Type *Definition* (DTD) to a XML document. This declaration may only appear once in an XML document, and it must follow the XML Declaration.
A declaration example:

```
<!DOCTYPE doc_element SYSTEM location >

<!DOCTYPE PRODUCT SYSTEM "http://127.0.0.1/DTD/file.dtd" >
```

Document Type Definition (DTD)
The Document Type Definition is used to define the elements specified by the user. Elements are defined using Element Type Declarations with the keyword ELEMENT. It specifies the name and the type of the elements.

The DTD can both be internal or external. That means that all the definitions are done within the XML document or having the definitions in a separate file.

In the example above, the tags `<PRODUCT>, <id>, <description>` and `<price>` where used. Using DTD they will be defined like:

```
<!DOCTYPE PRODUCT [
      <!ELEMENT id (#PCDATA) >
      <!ELEMENT description (#PCDATA) >
      <!ELEMENT price (#PCDATA) >
]>
```

There are five types of elements: Any, None, Text (PCDATA), Element and Mixed.

Processing Instructions (PIs)
A Processing Instruction contains information for the application using the XML document. This means that the XML interpreter (parser) passes the instructions to the application.
The PIs follow the generic syntax of:

```
<?target …instruction… ?>
```

For example:

```
<?xml-stylesheet href="mystyle.css" type="text/css"?>
```

Body
The body contains the payload of the XML data. It consists of a number of components. For example: tags, elements, and comments. The components are used to build the body tree structure, with a single root node.

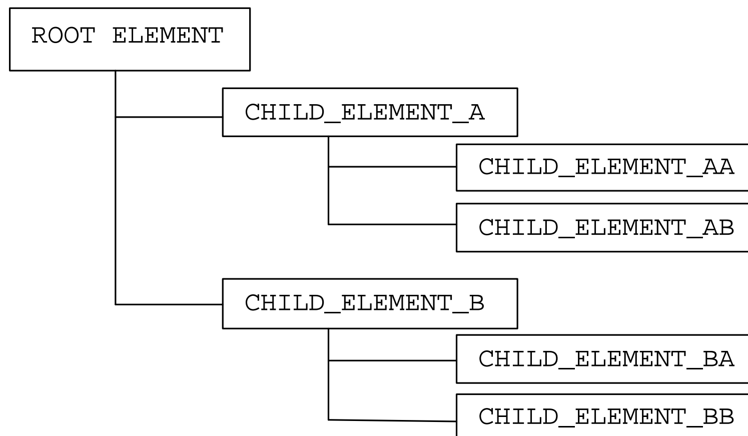


Figure 4.2 XML Body

Tags
A tag begins with a '<' and ends with a '>' and between them a tag name. There are two kind of tags, start- and end tags. The difference between them is that the end tag has a '/' sign before the tag name.
For example:

```
start tag          <NAME>
end tag            </NAME>
```

Because of XML is case sensitive the tags `<NAME>` and `<name>` are not equivalent.

Elements
Elements are the basic unit of a XML document. In short it consist of a start tag, data and an end tag. For example:

```
<NAME>Homer Simpson</NAME>
```

Comments
Comments in XML are written in the same way as in HTML.

```
<!-- This is a comment -->
```

Epilog

The epilog is optional and can contain some comments and/or processing instructions (PIs).

4.3 XSL

XSL (eXtensible Stylesheet Language) is a language for creating "style sheets" that describes how the data of an XML document is to be presented to the user.

XSL consist of two parts:

- one method for **transforming** XML documents, XSLT
- one method for **formatting** XML documents, XSL-FO

4.3.1 XSLT

XSL-Transformation is a way of transforming XML into something else. It could be from XML to HTML or from one XML document into another XML document. Here will only be discussed transformation from XML into HTML.

The example in Section 4.2.1 shows that XML without any style sheets do not look good. The XML document looked like this:

```
<PRODUCT>
      <id>12345678-Q</id>
      <description>AC 800M</description>
      <price>$99.99</price>
</PRODUCT>
```

An XSL – style sheet could be written like this:

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<html>
<body>
      <table border="2" bgcolor="white">
            <tr>
                  <th>Id</th>
                  <th>Description</th>
                  <th>Price</th>
            </tr>
            <xsl:for-each select="PRODUCT">
            <tr>
                  <td><xsl:value-of select="id"/></td>
                  <td><xsl:value-of select="description"/></td>
                  <td><xsl:value-of select="price"/></td>
            </tr>
            </xsl:for-each>
      </table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

As expected it looks like this in the web browser:

Id	Description	Price
12345678-Q	AC 800M	$99.99

In this example the advantage of having to write two separate documents to accomplish the same thing as in HTML may not be seen. But you do benefit from it.
For example: if there are several “products” it is possible to iterate through them with the command `<xsl:for-each` and they would be displayed as shown below.

Id	Description	Price
12345678-Q	AC 800M	$99.99
24682468-P	AC 800C	$49.95
…	…	…

There are a number of XSLT elements like `<xsl:for-each` or `<xsl:value-of` to work with when building style sheets. With XSL it is possible to remove, rearrange and sort XML elements, and also make decisions about which elements to display.

This is the great advantage of using stylesheets, it allows a single XML document to be displayed in many different ways.

XSLT Syntax
XSLT basically use the same structure as XML [6]. It includes the XML Declaration and a Process Instruction to indicate that the document is an XSLT style sheet.

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
      …
</xsl:stylesheet>
```

For displaying the information HTML syntax is used to create a simple table with headings in bold text (**Id, Description, Price**) and the data in plain text.

4.3.2 XSL-FO

XSL Formatting Objects is another way of presenting XML. While XSLT primarily was intended for uses on the Web, XSL-FO is more focused on paper. For example it is possible to store information on a database as XML documents and be able to display it both on the web using XSLT and on paper with XSL-FO [5].

4.4 SOAP

The Simple Object Access Protocol (SOAP) is a protocol for exchanging information in a distributed environment [5], for example, between computers on a LAN (Local Area Network) or over the Internet. The idea of SOAP is to make it possible for computers to talk to each other, regardless of their operating system.

4.4.1 How does it work?

A SOAP message is basically a one-way message. However, often it is used for request/response applications. A SOAP message is based on the XML technology and uses HTTP as the transmission protocol. By using HTTP, SOAP is a firewall-friendly application. To be able to use SOAP, all peers (computers, servers etc) in a network need a SOAP parser. A parser is a program that can interpret and understand the message, and be able to respond to it. A SOAP parser can be written in different languages, e.g., Visual Basic, Java, and C++.

An interesting application of SOAP is Remote Procedure Calls (RPC). RPC is a protocol that a program can use to request a service from a program on another computer in a network, see Figure 4.1. The RPC uses the client/server model. The requesting program is the client and the called program is the server. A program/programmer that uses RPC does not need to know any network details. This is taken care of in a RPC runtime program, which can send and receive calls over the network.

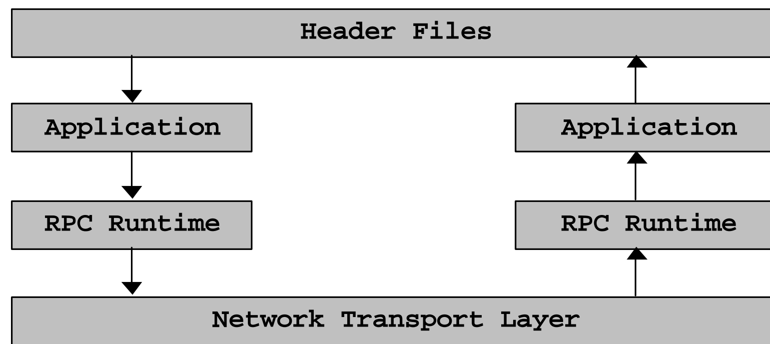


Figure 4.1 Remote Procedure Call

4.4.2 SOAP Message

A SOAP message consists of three parts: envelope, header and body, see Figure 4.1.

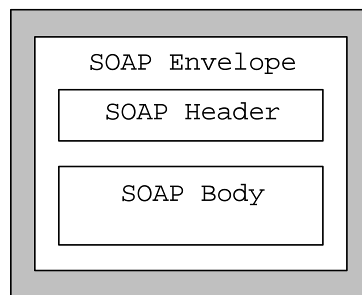


Figure 4.1 SOAP Message

Envelope

The envelope defines what is in the messages, and wraps the payload (contents). You can think of it as envelops used in the regular postal mail. The envelope is a mandatory element in a SOAP message. The envelope element is written like this:

```
<SOAP-ENV: Envelope>
      ...
</SOAP-ENV: Envelope>
```

Header

The header element is an optional element. It can for example include information for authentication or user information. The header element looks like:

```
<SOAP-ENV: Header>
      ...
</SOAP-ENV: Header>
```

Body

The body contains the payload of the SOAP message. It can contain RPC calls and replies, error messages or other one-way messages. The body element uses the same syntax as the other SOAP elements.

```
<SOAP-ENV: Body>
      ...
</SOAP-ENV: Body>
```

4.4.3 A SOAP example

In this example two computers in a distributed system will communicate over the network using SOAP. Computer A will ask Computer B to calculate an output with the function `CalcSum()` with the parameters `Input1` and `Input2`. Computer B will execute the instructions and return a value `Sum`. The function `CalcSum()`can be written in different languages, the only requirement is that both sides can interpret the call.

The SOAP message from A to B:

```
<SOAP-ENV: Envelope>
      <SOAP-ENV: Header>
            <SOAP-ENV: Body>
                  <m:CalcSum>
                        <Input1>1.32</Input1>
                        <Input2>1.45</Input2>
                  </m:CalcSum>
            </SOAP-ENV: Body>
      </SOAP-ENV: Header>
</SOAP-ENV: Envelope>
```

After processing, B will return the value:

```
<SOAP-ENV: Envelope>
      <SOAP-ENV: Header>
            <SOAP-ENV: Body>
                  <m:CalcSumResponse>
                        <Sum>2.77</Sum>
                  </m:CalcSumResponse>
            </SOAP-ENV: Body>
      </SOAP-ENV: Header>
</SOAP-ENV: Envelope>
```

Notice the extra `Response` added to the function name.

4.5 Summary

HTML and XML/XSL are two ways of storing and presenting data. When viewing the documents in a web browser it is impossible to notice the difference because XML documents are translated to HTML via XSLT. The difference lies in how the data is structured. HTML documents contains all information needed for processing and displaying. XML documents separates data from presentation, which makes it possible to display a single XML document in many different ways. This makes XML a much more powerful tool, than HTML.

XML can also be used in other applications. SOAP uses XML documents for creating firewall-friendly calls between computers and servers. SOAP is a protocol for exchanging information on a network, regardless of the operating system.

5 Scripts

Script languages are simpler than ordinary programming languages like C and C++. Scripts are used in all web pages where some kind of user input is required.

A script language is an interpreted and limited language. Interpreted because programs are runnable directly when the program is written. The programs are treated and shown in a web browser directly, they do not have to be compiled into machine code first. Limited since it is not as powerful as ordinary programming languages.

Scripts are relatively easy to use, but there are limitations. If ordinary programming language programs contain syntax errors, the user will be informed about these during the compilation phase. Since a script language does not have to be compiled the user misses this information and it is hard to see if the program is accurate.

5.1 CGI script

CGI is short for Common Gateway Interface [7].

Common – CGI programs can be written with many languages e.g. C, C++, Java, Perl or any other language that accepts user input, processes that input and responds with output. CGI works with many different types of systems, e.g. Mac, NT, Windows and UNIX.

Gateway – CGI's premier goal is not to accomplish things on its own. CGI can be seen as a middleman or a translator whose job is to help more powerful resources like databases or network applications to talk to each other.

Interface – CGI is not a language, nor a program. It is a standard of communication, an interface that provides well-defined rules for creating partnership. If everyone follows the rules of the interface, then everyone can talk to everyone.

5.1.1 Why CGI?

HTML is good at distributing pre-prepared web pages on request, but when it comes to dynamically generated web pages HTML is very limited. A client using a web browser asks the web server via HTTP for a specific HTML document. The web server then sends the requested document back to the web browser, which in turn, displays the document. The interaction between the client and the server is extremely trivial. The server can only provide static HTML documents that have been encoded in advance. With CGI scripts web pages can be created on the fly.

5.1.2 Functionality

When the user requests a web page, for example by clicking on a hyperlink or entering a web site address (URL), the server sends back the requested page. However, when a user fills out a form on a web page and sends it to the server, it usually needs to be processed by an application program. The web server typically passes the form information to a small application program that processes the data and may send back a confirmation message. This method or convention for passing data back and forth between the server and the application is called the Common Gateway Interface (CGI). It is part of the Web's Hypertext Transfer Protocol (HTTP). In Figure 5.1 an example shows how a form is processed.

1. The user fills out a form and sends it to the server.
2. The server executes a CGI script.
3. The CGI script uses other server resources.

4. The CGI script creates HTML pages with dynamically obtained information (the data filled in the form).
5. The server sends the HTML page to the browser.

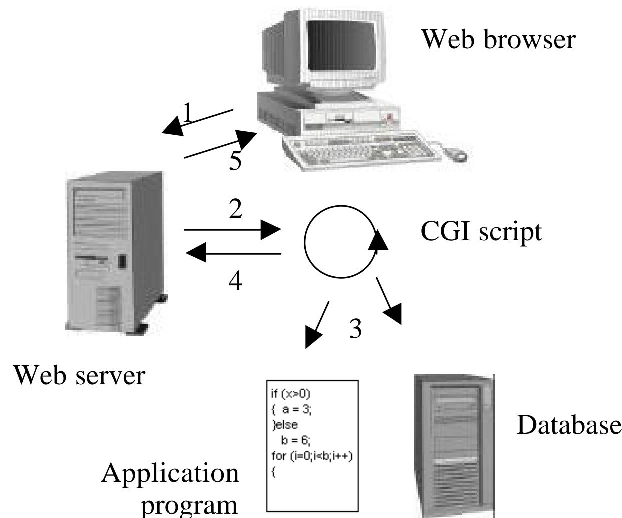


Figure 5.1 Example of how a form is processed with CGI script.

5.1.3 Common use

Some of CGI's most common use is as part of

- counters on web pages, i.e. how many hits you have on a web page.
- processing forms, e.g. when a web surfer enters data into a HTML form and sends it to a web administrator.
- ongoing dialog between multiple clients, e.g. real time chat.
- guest books on web pages.

5.1.4 Limitations

A web server can have several clients at the same time. If every one of them does something (e.g. fills out a form) that causes the server to run a CGI script, the load on the server becomes very heavy. If the server has limited performance it can result in a server crash. The problem is that every CGI request from a user starts a new process in the server. Even if the server has sufficient performance many parallel executing processes makes the server slower. Therefore, CGI scripts are not suitable for embedded systems that demand compact, high performance solutions. The web server used in this thesis uses a kind of CGI script called "GoForms" that does not create a new process for every request, see Section 6.2.2.

5.2 JavaScript

JavaScript was developed by Netscape and was intended to be a uniform replacement for all different CGI languages. It was first called LiveScript, but when Netscape and the Java developer Sun made a union, it became JavaScript [8].

5.2.1 Why JavaScript?

The problem with ordinary CGI scripts, as mentioned before, is that the load on the server can become very heavy. The main problem is that all processing are done on the server side. If

some of it were done on the client side it would relieve the pressure on the server. This is what JavaScript can do.

5.2.2 Functionality

When a user fills out a form on a web page some uncomplicated processing can be made on the client side, see Figure 5.1. Things that could be done are for instance checking that all fields in a form are filled, that the e-mail address is valid etc. This leads to less communication between the server and the client i.e. a reduction of the network load. The main goal though is to reduce the load on the server.

```
1. The user fills out a form.
2. The client side check that all fields in the form are filled
   and correct.
3. The client sends it to the server.
4. The server executes a JavaScript.
5. The JavaScript uses other server resources.
6. The JavaScript creates HTML pages with dynamically obtained
   information (the data filled in the form).
7. The server sends the HTML page to the browser.
```

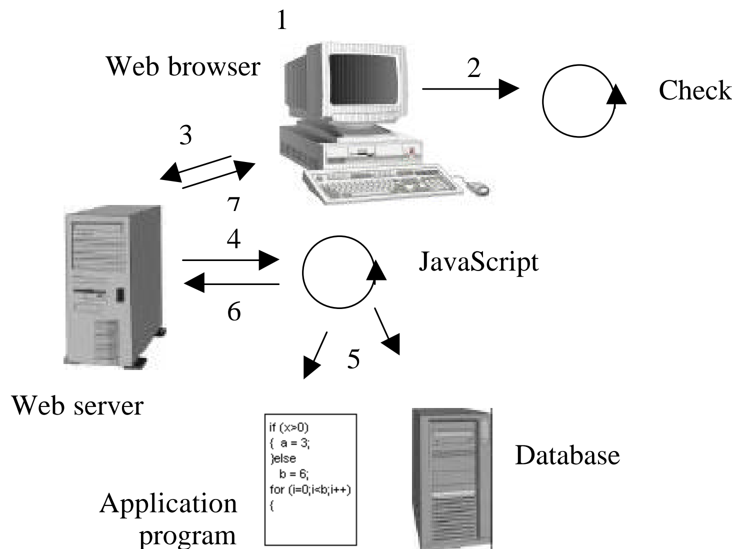


Figure 5.1 Example of how a form is processed with JavaScript.

JavaScript uses an object-oriented model with predefined objects like math, date and string. With these objects follows a set of functions, the string object has for example several methods to work with the content in the string, but there also exists methods that return text translated to HTML code. The date object has for instance one method to automatically update today's date on the web page. It is also possible to create your own objects and functions.

The JavaScript code is a part of the HTML code, not a separate component. JavaScript is not a strictly typed language, which means that a variable does not have to be declared as a particular data type and that data types are automatically converted when needed.

Events are essential in JavaScript. An event occurs when the user for example clicks on a button or moves the mouse over a hyperlink. JavaScript has certain event-handlers that handle the occurred event, e.g. "onClick" and "onMouseOver".

There is a major problem with JavaScript and that is the different web browsers. They all treat JavaScript differently and some of them are not able to interpret JavaScript at all. JavaScript's can be used for checking that forms are filled out correctly. It can also be used for graphical applications, e.g. showing fancy text in the status bar when the mouse is moved over a hyperlink.

5.3 Summary

Script languages are easier and faster to code in than more structured and compiled languages such as C and C++ and are ideal for programs of very limited capability. However, a script takes longer to run than a compiled program since each instruction is being interpreted and handled by another program first rather than by the basic instruction processor. CGI script is a way to process data on the server side. In JavaScript on the other hand some of the processing is done on the client side.

6 Web server

This chapter contains information about different web servers and a more detailed description of the web server that is used in the implementation. A brief overview of how the ABB Controller software is organized and where the web server is located in the source code is also presented. The last section contains a discussion about some real-time properties and memory handling.

6.1 Different web servers

This section contains a comparison between some of the web servers that can be found on the Internet. The basic requirements were that it should be designed for embedded systems, have support for CGI-script, and be compatible with the real-time operating system pSOS. Five different servers were found:

- Wind River System: Wind Web Server 2.0
- Virata Corporation: EmWebServer.
- Quiotix: Quiotix Embedded WebServer
- GoAhead: Software Inc., GoAhead® WebServer™ 2.0 and 2.1

Table 6.1 contains a comparison of the web servers according to the basic requirements and also how much memory they use and if they are open source (if the code is accessible for the user). Also if they support the use of storing web pages on ROM (if no file system exists) and finally if they are freeware or not.

	Embedded	CGI support	pSOS compatible	ROM storage	Memory footprint	Open source	Free
Wind Web Server 2.0	yes	yes	no	yes	10-60 kb	no	no
Virata EmWebServer	yes	yes	yes	**	20-25 kb	**	no
Quiotix Embedded WebServer	yes	yes	yes	yes	**	yes	no
GoAhead Web Server 2.0	yes	yes	yes	yes*	50 kb	yes	yes
GoAhead Web Server 2.1	yes	yes	no	yes*	60 kb	yes	yes

Table 6.1 Different web servers, * = not fully implemented, ** = no information.

6.2 GoAheadâ WebServerä

The GoAhead® WebServer™ 2.0 was developed by GoAhead Software Inc and released in June 1999 [9]. It has been designed for embedded systems. It has a relative small memory footprint and is compatible with pSOS. Further more it is free and the code is open source. This means that programmers have access to the source code, and are allowed to modify it so it meets their needs. This is why the GoAhead Webserver was chosen in this project.
The web server also has support for other operating systems. These are Windows 95/98/NT/CE, VxWorks 5.3.1, LynxOS, UNIX and Linux. The company Innocor has ported the version used from VxWorks to pSOS.

6.2.1 Features and limitations

The GoAhead® WebServer™ 2.0 includes the basic techniques used in the Internet world. It has support for Active Server Page (ASP), an embedded JavaScript parser, and an in-memory forms processing technique (CGI-script) called GoForms™. Active Server Page is a technique for processing scripts in a HTML page developed by Microsoft. Furthermore it has an

extensible method for handling URLs. The GoAhead® WebServer™ 2.0 also has support for retrieving web pages stored in ROM. A compiler is included in the source code and makes it possible to build web pages and then compile them into C source code, which can be downloaded to the system. The web server also has support for login access. Further, there is support for HTTP/1.1, memory and stack usage tracking and support for proxy capability. The GoAhead® WebServer™ 2.0 has some limitations. The ROM page retrieving functionality is not fully implemented. The support for proxy usage and HTTP 1.1 are not fully tested and may include some errors. ASP pages are processed in-memory. This means a large ASP page can consume significant memory, which could slow down or even starve out other processes in the controller. The login access support also has some limitations. There is only a single global password that is set by the programmer before compiling and downloading. Also, there is no encoding/decoding of the password.

6.2.2 GoForms

The GoAhead implementation of the standard Common Gateway Interface (CGI) is called GoForms. Ordinary CGI processing results in the creation of a new process for every request to a CGI, but GoForms procedures run without creating a new process for each browser connection. GoForms is therefore a more suitable solution for embedded systems that require high performance solutions. The GoForms implementation results in that CGI variables are easy to access. When a form is filled out and its action function is called the URL can look like this:

```
/goform/menulayout?node=hardware&item=PM210
```

The action function is `menulayout` and the CGI variables are `node` and `item`. `node` has the value `hardware` and `item PM210`. With the function `websGetVar()` the value of the CGI variables can be accessed. `websGetVar()` is used like this:

```
char *cgiNode, *cgiItem;
cgiNode = (char*) websGetVar(wp, "node", "error");
cgiItem = (char*) websGetVar(wp, "item", "error");
```

In this example `cgiNode` gets the value `hardware` and `cgiItem PM210`. The last argument is a default value, which will be returned if the CGI variable is not accessible. It is then possible to check that the CGI variable has an appropriate value with a simple if statement.

```
if (cgiNode != "error"){
        ...
}
```

6.2.3 Modifications

Some modifications in the source code have been done in this thesis. The main modification is that web pages are not stored in a file system nor using compiled pre-defined web pages stored in ROM. Instead there are a number of template web pages. These are then dynamically created when requested from a web browser. In Chapter 7 this will be explained further.

Another modification that have been made was a bug fix, in the routine called '`websResponse`' which, in short, sends the HTTP header and the web page to the requesting web browser. The HTTP header included an error, the response code (see 3.1) was not correct. The result was that web pages in XML format could not be understood by a web browser and be displayed correctly. Functionality for choosing what kind of content-type

(section 3.1) that should be used in the HTTP header, i.e. `text/html`, `text/xml` or `image/gif` was also added.

6.3 Generic web server

The software for the ABB Controller includes everything from the Control Builder program, control application to hardware drivers. The source code is divided into two major parts; Atlas and Omega. See Figure 6.1.

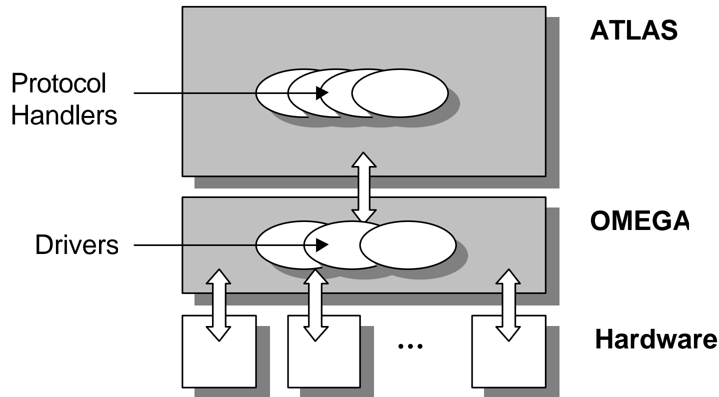


Figure 6.1 ABB Controller source code.

Atlas contains the generic parts of the source code, for example protocols for communications (PROFIBUS, RS232). It is executed both in Control Builder and in the different controllers (e.g. Soft Controller, AC 800M). Generic means that it is hardware independent. Atlas communicates with the hardware through Omega. Since Atlas is generic, the interface between Atlas and Omega is the same regardless of the underlying hardware.
Omega contains the hardware specific code, operating system and hardware drivers, e.g., serial communication drivers and socket routines.

The controller software, that is a part of Atlas, is the executing part in the controller. It is in the controller software (or firmware) the web server is implemented, see Figure 6.2.

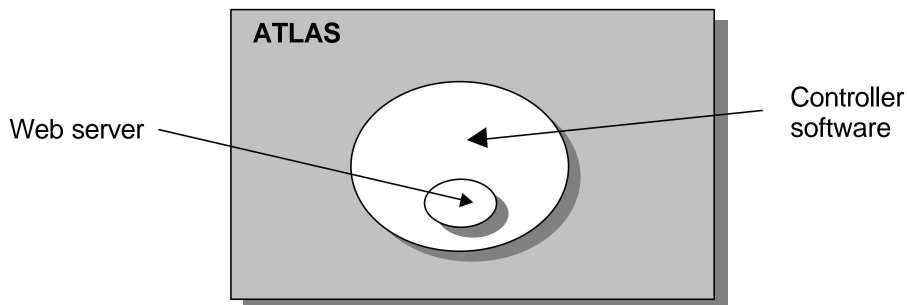


Figure 6.2 The location of the web server.

By including the web server in the controller software and using the well-defined interface between Atlas and Omega the server becomes generic, and can be used on different controllers without any changes.

6.4 Real-time properties and memory handling

6.4.1 Real-time properties

In the controller there are a number of parallel processes/threads. Since the controller is a system with a single CPU unit, true parallel execution is not possible. This leads to concurrency between the different processes. The available CPU capacity must be shared between the processes. A control system is often a hard real-time system, which means that all deadlines must be met otherwise the system might fail or, even worse, crash.
The web server program runs as a single separate thread with a priority that is lower than the priority of the main control program. This is to guarantee that the main control program always gets to run when needed. If the web server is running and the main program wants to execute, a context switch occurs and changes the running process to the main program.

In pSOS the priorities lies between 0-255 (255 = highest priority).
Figure 6.1 shows an overview of some of the different threads that are executing in the controller [10].

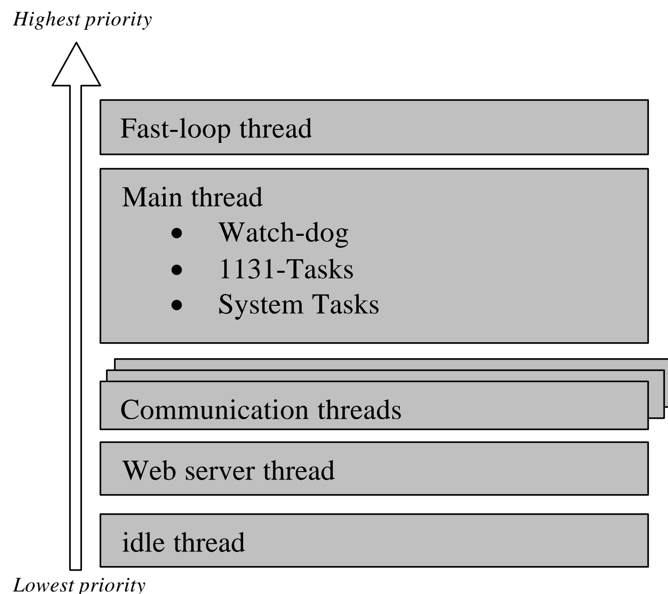


Figure 6.1 An overview of the threads in the controller.

- The Fast-loop thread (also named Time Critical task) –
  executes Control Applications Tasks with short intervals and time critical priority.
- The Main thread –
  executes the Scheduler for Control Application Tasks and other system generated tasks (sockets, memory handling).
- A set of Communication Protocol threads–
  executes different protocols, like TCP/IP and PPP.
- The web server thread.
- The idle thread –
  executes when no other thread is active. Its task is only to wake the Main thread up again.

The programs are implemented in C and C++. These are not real-time programming languages. To achieve the real-time functionality a real-time operating system (RTOS) must be used. As discussed in Section 2.4 the operation system used in the controller has a real-time kernel (pSOS+), which makes this possible.

6.4.2 Memory handling

In an embedded real-time environment correct memory handling is very important. Because of the limited amount of memory even a minor memory leak will eventually lead to that a program stops executing. To avoid this it is of outmost importance to ensure that no such leaks exist.

The memory is divided into four parts or areas: code, data, stack and heap.
The code area consists of the compiled and linked object code. The code is never modified during execution and is fixed in size. The data area is a static area, which holds global variables that are stored during compilation, e.g. variables declared '`static`' in a C program. The stack area is used for parameters in function calls and also holds local variables in blocks and functions. The stack size can often be estimated before execution. It is a function of the maximum depth of procedure calls and the amount of memory needed by the procedures. The stack is well behaved. It expands and contracts at one end with no wasted size.
The heap is used as memory area for dynamic memory allocation. The running programs create data structures during run-time that are then accessed by pointers. The heap storage requirement cannot be calculated in advance since dynamic allocation is usually used for data structures like lists whose size depends on the input to the program. The heap is not well behaved because allocation and de-allocation of data areas of different sizes can occur at any time and at any place within the heap.

The problem with dynamic memory allocation is that the system can run out of memory during runtime. If this happens, the system can stop running. To minimize the risk of memory shortage it is preferable to avoid dynamic memory allocation if possible. It is in most cases not possible to do this. It is very important that all memory that is allocated during run-time is de-allocated.

The basic implementation of the GoAhead® WebServer™ 2.0, with some modifications, needs about 18kb of RAM for the code and static variables. Table 6.1 is a list of the different versions and their memory requirements.

Version	Memory
Basic, with no additions	18kb
JavaScript version (including pictures)	71kb
CGI-script version	48kb
Pictures	17kb

Table 6.1 Different version of the web server and their memory requirements

The basic version of the web server is a version without any web pages, so it is useless. The JavaScript version uses a JavaScript in the start page and it includes a number of pictures. The CGI-script version is a simpler version of the start page, which does not use any pictures.
In Section 7.3 more information about the start page can be found.

The web server thread uses 10kb of stack memory. To ensure that there is enough free memory on the heap when performing allocations, the web server program always checks if it is possible to allocate the requested memory.

```
char *msg=new char[100];
if (msg != NULL){
	...
	delete [] msg;
}
else {
```

```
        websHeader(wp);
        websWrite(wp, "Error: memory allocation failed");
        websFooter(wp);
        websDone(wp, 200);
    }
```

When trying to allocate a string of characters (`char *msg = new char[100]`) and there is not enough memory on the heap, `new` will return `NULL` and an error message will be shown in the web browser. If the allocation went well it is very important not to forget to de-allocate the string (`delete [] msg;`). For more information about `websHeader`, `websWrite`, `websFooter` and `websDone`, see Section 7.2.

6.5 Summary

There are a number of web servers that can be found on the Internet. To be able to implement a web server in the ABB environment some important features were required: pSOS compatibility, small memory footprint, and design for embedded systems. The GoAhead® WebServer™ 2.0 met these requirements.
The web server code is inserted in the generic part of the ABB software called Atlas. This is done because the web server shall be used in different controllers.
The web server executes as a single thread with a priority lower than the main control program. This is to ensure that the web server does not use unnecessary CPU resources.
Memory handling is also very important in an embedded system. Dynamic allocation and de-allocation of memory must be done in a controlled way, to ensure that no memory leaks occur.

7 The program

This chapter describes the implementation of the web server program. It starts with an overview. Then follows a discussion about the initialization of the program and a description of the common procedures and techniques. Finally the program is described in detail.

7.1 Overview of the program

After initialization where the thread is created and started, the web server waits for a connection. The web server listens to a socket for a HTTP-request. When a request is detected a new socket is opened and the request is received. After deciphering the request a check is made to see if there are any matching URL’s. If so, the requested page (or picture) are created and sent to the browser. Finally the socket is closed and the web server waits for a new request. An overview is given in Figure 7.1.

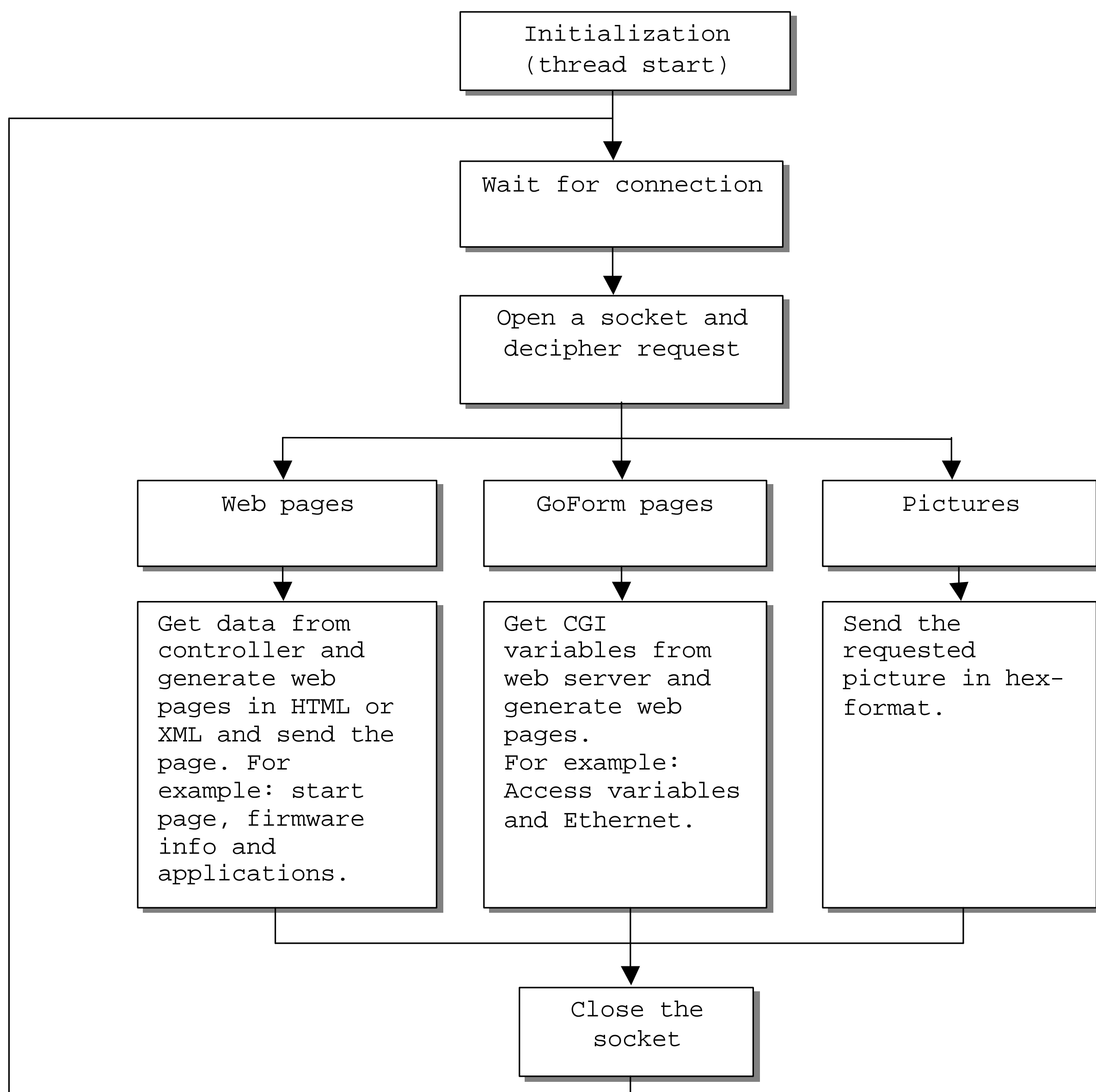


Figure 7.1 Overview of the program.

7.2 Common techniques

7.2.1 Initialization

In the initialization phase of the program some handlers are defined. These are handlers for different URL's (URL handlers) and handlers for different forms (form handlers).

A handler is tied to an URL. The handler is no more than an ordinary procedure. The line of code that defines an URL is:

```
websUrlHandlerDefine("/index.html", NULL, 0, Handler, 0);
```

Here the URL `/index.html` is defined. `WebsUrlHandlerDefine()` is part of the GoAhead web server source code. The handler is a procedure that contains if statements which compare the incoming URL with the defined URL's. If the URL is found the function that is supposed to handle that URL is called.

```
if (gstrcmp(url, "/index.html") == 0) {
      indexPage(wp, "index.html");
}
```

The function that handles a call with `/index.html` is `indexPage()`.
Form handlers are defined with the GoAhead function `websFormDefine()`.

```
websFormDefine("accVarForm", accVarForm);
```

In the example above the function `accVarForm` is defined as a form handler. The reason why `websFormDefine()` has two parameters is that both the name as a string (first argument) and the function it self (second argument) is needed. The HTML tag <form> creates a form.

```
<form action=\"/goform/accVarForm\">
```

The action argument declares the function that should process the form when it is filled out. The different form handler names and form handler functions are stored in a table, see Table 7.1. This table is needed when a form is filled out and the corresponding handle function should be found. The GoAhead web server contains a technique to treat forms. This technique is called GoForms, see Section 6.2.2.

Form handler name	**Form handler function**
accVarForm	accVarForm()
serialForm	serialForm()

Table 7.1 The handlers are stored in a table

After defining the handlers the thread is created and started. This is done with calls to pSOS functions:

```
rc = t_create("tWWW", 10, 10000, 10000, T_LOCAL | T_NOFPU,
&tid);

rc = t_start(tid, T_SUPV | T_PREEMPT | T_TSLICE | T_NOASR |
T_ISR, webserver_main, args);
```

The arguments in t_create() are: task name, task priority, task supervisor stack size, task user stack size, task attributes and task identifier and in t_start(): task identifier, initial task mode, task address and startup task arguments. The task address is here the function webserver_main() and this is where the program waits for a connection [11].

7.2.2 Generating and sending web pages

The method used for generating web pages is based upon a string that is extended piece by piece. The C standard function strcat() is used to append pieces to the string.

```
char *strcat(char *strDestination, const char *strSource);
```

It is used like this:

```
char *msg=new char[50];
strcpy(msg, "");
strcat(msg, "<HTML><HEAD><TITLE>");
```

Here the string msg is filled with <HTML><HEAD><TITLE>. Depending on what is to be displayed different things are appended. If, e.g., a table should be displayed the next line would be

```
strcat(msg, "<TABLE>");.
```

The example in Section 4.1 is created like this:

```
char *msg=new char[500];
strcpy(msg, "");
strcat(msg, "<HTML><HEAD><TITLE>Example</title></head>");
strcat(msg, "<body><h1>4Header</h1>Here comes the body text.");
strcat(msg, "<h2>4.1 A list</h2><ul><li>First element</li>");
strcat(msg, "<li>Second element</li></ul></body></html>");
```

It is hard to see if the HTML code is correct using this compact way of writing. The page creation is not very generic either. Therefore a set of help functions is implemented. These functions can bee viewed as a HTML generator and an XML generator. The function structures are the same, text and tags are appended to the string and the string is returned. Before any characters are appended to the string a memory check is performed, by calling CheckLength(), to check that it is enough space in the string to append the characters. If there are not enough space left, CheckLength() allocates space for 500 more characters. Below the functions are shown. The strings that are returned are printed below each function. Some of the functions are custom-made for one special task in the program, others are more general. Therefore, some functions are quite large with a number of input arguments.

HTML

The NewPage() function creates a new page by appending <HTML>, <BODY> and BGCOLOR tags to the string *pstr.

```
char* NewPage(char *pstr, char *Color);
      //<HTML><BODY><BODY BGCOLOR="Color">
```

The `NewFramePage()` function creates a new frame page with three frames.

```
char* NewFramePage(char *pstr, char *Title, char *logo, char
*left, char *right);
      //<HTML><HEAD><TITLE>Title</TITLE></HEAD>
      //<FRAMESET COLS="300,*">
      //<FRAMESET ROWS="90,*" BORDER="1" FRAMEBORDER="1">
      //<FRAME SRC="logo" NAME="Logo">
      //<FRAME SRC="left" NAME="Left">
      //</FRAMESET><FRAME NAME="Right">
      //</FRAMESET></HTML>
```

The `NewRow()` function creates `Nbr` new rows.

```
char* NewRow(char *pstr, int Nbr);
      //<BR>
```

The `Font()` function activates the font `Font` with the font size `Size`.

```
char* Font(char *pstr, char *Font, char *Size);
      // <FONT FACE="Font" SIZE=Size />
```

The `Text()` function adds the text `Text`.

```
char* Text(char *pstr, char *Text);
      //"Text"
```

The `BoldText()` function adds the text `Text` in bold.

```
char* BoldText(char *pstr, char *Text);
      //<B>"Text"</B>
```

The `Line()` function creates a line with a thickness determined by `Size`.

```
char* Line(char *pstr, char* Size);
      //<HR SIZE="Size">
```

The `NewTable()` function creates a table with the color `Color`.

```
char* NewTable(char *pstr, char *Color);
      //<TABLE BORDER="1" CELLPADDING="1" BGCOLOR="Color">
```

The `NewAlignedTable()` function creates a table with the color `Color` and with the alignment `Align`.

```
char* NewAlignedTable(char *pstr, char *Color, char *Align);
      //<TABLE BORDER="1" CELLSPACING="1"
      //BGCOLOR="Color" ALIGN="Align">
```

The `NewTableCol()` function creates adds a new cell in a table.

```
char* NewTableCol(char *pstr);
      //<TD>
```

The `NewAlignedTableCol()` function adds a new cell in a table with the alignment `Align`.

```
char* NewAlignedTableCol(char *pstr, char *Align);
      //<TD ALIGN="Align">
```

The `EndOfTableCol()` function ends a table column.

```
char* EndOfTableCol(char *pstr);
      //</TD>
```

The `NewTableRow()` function creates a new row in a table.

```
char* NewTableRow(char *pstr);
      //<TR>
```

The `EndOfTableRow()` function ends a table row.

```
char* EndOfTableRow(char *pstr);
      //</TR>
```

The `NewTableHeading()` function adds a header cell to a table.

```
char* NewTableHeading(char *pstr);
      //<TH>
```

The `EndOfTableHeading()` function ends a table header cell.

```
char* EndOfTableHeading(char *pstr);
      //</TH>
```

The `EndOfTable()` function ends a table.

```
char* EndOfTable(char *pstr);
      //</TABLE>
```

The `NewList()` function adds a list.

```
char* NewList(char *pstr);
      // <DL COMPACT>
```

The `GroupTitle()` function adds a group title to a list.

```
char* GroupTitle(char *pstr);
      // <DT>
```

The `GroupItem()` function adds a new row in a list.

```
char* GroupItem(char *pstr);
      // <DD>
```

The `EndOfList()` function ends a list.

```
char* EndOfList(char *pstr);
      // </DL>
```

The `Center()` function centers the whole page or a part of the page.

```
char* Center(char *pstr);
      //<CENTER>
```

The `EndOfCenter()` function ends the center alignment.

```
char* EndOfCenter(char *pstr);
      //</CENTER>
```

The `HyperLink()` function adds a link to `LinkFilename` with the link text `LinkText`.

```
char* HyperLink(char *pstr, char *LinkFilename,char* LinkText);
      //<A HREF="LinkFilename" STYLE="TEXT-DECORATION: NONE">
      //LinkText</A>
```

The `MenuHyperLink()` function adds a menu link to `cgiItem` under the `cgiNode` category in the menu. It has the link text `LinkText`.

```
char* MenuHyperLink(char *pstr, char *cgiNode, char *cgiItem,
char *LinkText);
      //<A HREF="goform/menuLayout?node=cgiNode&item=cgiItem">
      //LinkText</A>
```

The `AdvancedHyperLink()` function adds a link to `LinkFilename` with the link text `LinkText`. The page is displayed in the `Target` frame.

```
char* AdvancedHyperLink(char *pstr, char *LinkFilename,char
```

```
          *Target, char *LinkText);
          //<A HREF="LinkFilename"
          //TARGET="Target">"LinkText"</A>
```

The `EndOfPage()` function ends the page.

```
    char* EndOfPage(char *pstr);
          //</BODY></HTML>
```

XML

The `xmlDeclatation()` function generates the XML declaration.

```
    char* xmlDeclaration(char *str);
          //<?xml version='1.0' ?>
```

The `StylesheetToUse()` function adds the xsl stylesheet located in file `file`.

```
    char* StylesheetToUse(char *str, char *file);
          //<?xml-stylesheet type="text/xsl" href="file" ?>
```

The `StartTag()` function adds the start tag `tag`.

```
    char* StartTag(char *str, char *tag);
          // Start tag: "<tag>"
```

The `EndTag()` function adds the end tag `tag`.

```
    char* EndTag(char *str, char *tag);
          // End tag: "</tag>"
```

The `EmptyTag()` function adds an empty tag `tag`.

```
    char* EmptyTag(char *str, char *tag);
          // Empty tag; "<tag />"
```

The `NewElement()` function adds the element `cont` between the start and end tag `tag`.

```
    char* NewElement(char *str, char *tag, char *cont);
          // Element: "<tag>cont</tag>"
```

The `Attribute()` function adds a tag `tag` with the element `name`. The value of the element is `value`.

```
    char* Attribute(char *str, char *tag, char *name, char *value);
          // Attribute: "<tag name='value'/>"
```

The `Comment()` function adds the comment `text`.

```
    char* Comment(char *str, char *text);
          // Comment: "<!-- text -->"
```

The `DOCTYPE()` function defines the document type with `element`, `param` and `location`. If using external DTD files, `param` must be 'SYSTEM' or 'PUBLIC'.

```
    char* DOCTYPE(char *str, char *element, char *param, char
    *location);
          //<!DOCTYPE element param location >
```

The `DTD_start()` function indicates the start of the internal DOCTYPE definition.

```
    char* DTD_start(char *str, char *element);
           //<!DOCTYPE element [
```

The `DTD_end()` function ends the internal DOCTYPE definition.

```
    char* DTD_end(char *str);
          // ]>
```

The `PI()` function indicates a processing instruction.

```
    char* PI(char *str, char *target, char *instr);
          //<?target instr ?>
```

XSL

The `NewStylesheet()` function declares the start of a xsl document.

```
char* NewStylesheet(char *str);
//<xsl:stylesheet xmlns:xsl=
//"http://www.w3.org/TR/WD-xsl">"
```

The `EndOfStylesheet()` function ends an xsl document.

```
char* EndOfStylesheet(char *str);
        //</xsl:stylesheet>
```

The `NewTemplate()` function adds a new template with the text `text`.

```
char* NewTemplate(char *str, char *text);
        //<xsl:template text>
```

The `EndOfTemplate()` function ends a template.

```
char* EndOfTemplate(char *str);
        //</xsl:template>
```

The `ValueOf()` function adds the value of the attribute `attr`.

```
char* ValueOf(char *str, char *attr);
        //<xsl:value-of select="attr"/>
```

The `NewForEach()` function iterates through every element of type `element`.

```
char* NewForEach(char *str, char *element);
        //<xsl:for-each select="element"/>
```

The `EndOfForEach()` function ends the iteration.

```
char* EndOfForEach(char *str);
        //<xsl:for-each>
```

When the string is filled it is sent to the browser. This is done with the GoAhead function `websWrite()`. Some pages are quite large and `websWrite()` cannot write too many characters at one time. Therefore, when the string is filled with around 300 characters the string is sent. Then the string is reset. This is simply done by copying an empty string to the message string.

```
websWrite(wp, msg);
strcpy(msg, "");
```

Then the string can be filled again. Before the page can be sent some initial protocol specific instructions must be sent. This is done with the GoAhead function `websHeader()`. In Section 3.1 these instructions are described. When the entire page is sent `websDone()` must be called. `websDone()` finishes the communication and closes the socket.

In Figure 7.1 the page creation and sending are shown. First comes the initialization part where the message string (msg) is declared and memory is allocated. The setup function, `websHeader()`, is also called. Then comes the part where the actual page is sent. First the string is reset then the string is filled and at last sent. This is repeated until the whole page is sent. Finally comes the part where `websDone()` is called and the allocated memory for the string (`msg`) is deallocated.

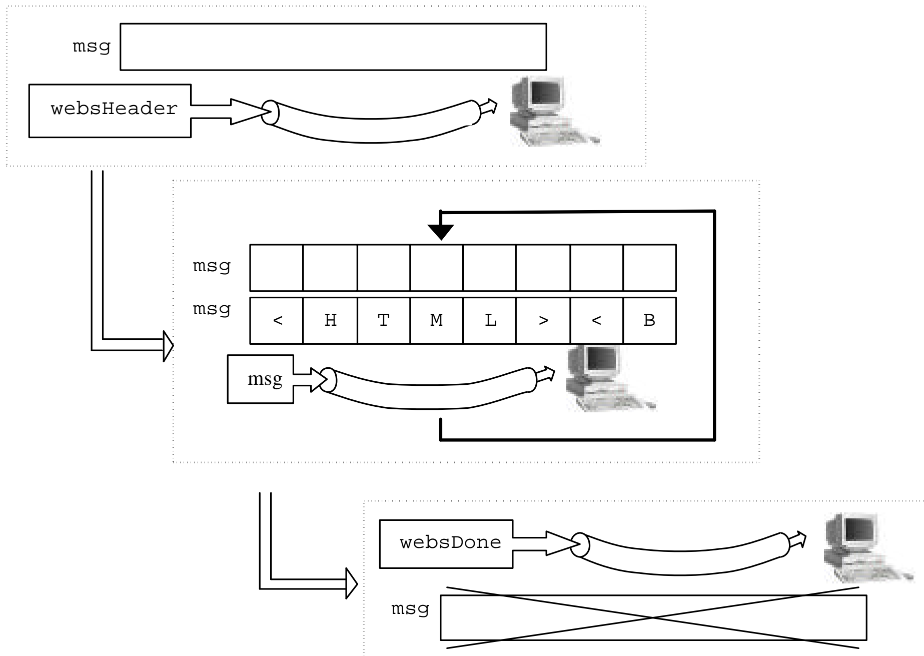


Figure 7.1 Page creating and sending.

7.3 Start page

The start page is intended to have the same tree structure as in the Control Builder (Figure 2.1). By using a well-known interface the web server becomes a user-friendly tool.
In the tree structure there are four main categories:

- Access Variables
- Hardware
- Applications
- Miscellaneous

The Access Variables category contains the names of the different access variables in the controller and a hyperlink to a web page with more details about the variables, see Section 7.4.
The Hardware category contains the different hardware units that are present in the controller. Their position and information about their firmware are also presented, see Section 7.5.
The Applications category contains the names of the 1131 applications that have been downloaded to the controller and a hyperlink to a web page with more details about the applications, see Section 7.6.
The Miscellaneous category contains information about heap memory usage and a log file containing various information, see Section 7.7.
The start page also includes an ABB-logo and information about the type of controller, the version and the manufacturers name. In Figure 7.1 the layout of the start page is shown. It consists of three frames, the logoFrame in which an ABB-logo is placed, the treeFrame in which the tree structure is located, and the rightFrame where various information about the controller is shown.

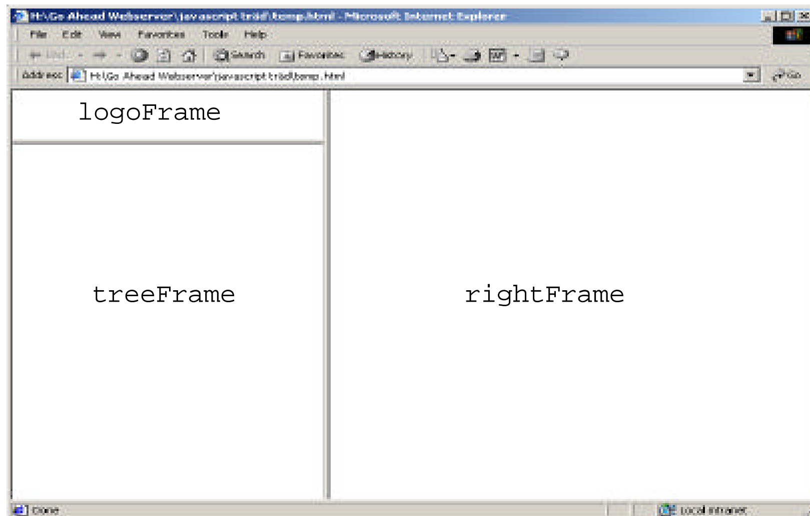


Figure 7.1 The frames in the start page.

There are two versions of the start page. They are based on two different techniques, JavaScript and GoForms. The reason for having two versions is to show different ways of displaying the information. The JavaScript version is used to get a tree-structure that is very similar to the project tree in Control Builder. The GoForms version is used for retrieving information with a small memory usage.

The advantage of using a JavaScript is that once the script and the pictures has been sent to the browser no communication with the server is needed, i.e., no unnecessary load on the controller. Another advantage is the similarity with Control Builder. The disadvantage is that it requires more memory than the GoForms version because it includes a number of small pictures that has to be stored in the controller and the amount of information can be large in a complex control system.
The advantage of using GoForms is that it requires less memory and it only sends the requested information. The drawbacks of using GoForms are that every time the tree is explored a request is sent to the web server and that the interface is not so user-friendly.

	Advantages	Disadvantages
JavaScript	Only generated and sent once, no unnecessary load on the web server and controller. A nice user interface, very similar to Control Builder.	Memory requirements for code and pictures.
GoForms	Small memory requirements.	Generates and sends every time the tree is explored. Does not have the same user-friendly interface.

Table 7.1 JavaScript vs. GoForms start page.

7.3.1 JavaScript version

This web page is based on a JavaScript called "OmenTree" [12]. It is free and can be downloaded from the Internet. OmenTree is flexible, easy to use and well suited for dynamically generated web pages. The JavaScript is stored and created in C-source code in the web server program. This JavaScript does only execute in the browser and not as a script described in Section 5.2.

The script contains a number of functions, the main functions are: `drawTree()`, `drawBranch()`and `loadData()`. The JavaScript creates the three frames on the web page (Figure 7.1), and when this is done the function `start()`is called. This function calls `loadData()` and `drawTree()`.

The function `loadData()` is divided into three sections.

- The tree structure.
- User defined variables.
- Additional HTML code.

The tree structure is where the content of the tree is defined. It is dynamically created when a request to the server is done. By calling four different functions the information in the folders (Access Variables, Hardware, Applications, and Miscellaneous) are added to the tree. These C-functions are `getAccVar()`, `getHWUnits()`, `getApplication()`, and `getMiscNodes()` and they are responsible for checking the controller for access variables, hardware modules, applications and miscellaneous information. These functions are only called once, when the JavaScript is generated in the controller. The user-defined variables are some default parameters, like `defaultLinkIcon` and `defaultTargetFrame`. Finally there is some additional HTML code, which include information about the product, version, and information about the vendor.

The JavaScript function `drawTree()` initiates the web page (sets background colour, font etc.) and after that calls the function `drawBranch()`. The function `drawBranch()` is the main function of the script. This recursive function draws the tree with its folders and branches.

Figure 7.1 An example of the JavaScript version.

The pictures used in the tree are of gif format and 19x16 pixels in size. They each use about 900 bytes of memory. Because the system (controller and operating system) does not have a file system (hard drive) the pictures cannot be downloaded as they are. Instead they must be converted into C source code that can be compiled and downloaded to the controller. The pictures are converted into hex-code and stored in a character array in a header file. For example the picture "cpu.gif" takes 906 bytes in size:

```
char cpu[906]={0x47, 0x49, 0x46, 0x38, 0x37, 0x61, 0x13, 0x00,
      0x10, 0x00, 0xF7, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80,
      0x00, 0x00, 0x00, 0x80, 0x00, 0x80, 0x00, 0x80, ...
```

Because the coded pictures might include the 'null-character' sign the function `websWrite()` cannot be used. This is due to that `websWrite()` uses string-related C-functions and this will lead to that the picture will not be sent correctly. Instead the system function `send()` is used. The ABB logo is also coded in the same way and the amount of memory required is about 1300 bytes.

The JavaScript is only generated once, which means that all information in the tree (all sub trees) is sent. If the controller includes a large number of access variables and applications the amount of code can become very large.

7.3.2 GoForms version

The second version of the start page is based on GoForms (Section 6.2.2). Even if the user does not fill out a form, this technique is suitable for interaction with the web server. The different categories in the tree structure (Access Variables, Hardware, Applications, and Miscellaneous) are in fact hyperlinks with two CGI-variables. By using this technique it is possible to create a tree that can be expanded. The two CGI-variables are `node` and `item`. The variable `node` is used to determine which category that has been selected (Access Variables, Hardware, Application or Miscellaneous) and `item` to determine which subcategory to expand (e.g. hardware units). When a link in the tree is clicked the page is rebuilt in the web server and updated in the browser.

When the start page is requested the corresponding C-function, `menyLayout()`, retrieves the two CGI-variables and then proceeds to generate the page. If the CGI-variable `node` corresponds to "accvar" the Access Variables branch is requested. The function `NamesOfAccVar()` is called to get the access variables. If `node` corresponds to "hardware" the different hardware units are retrieved with the function `getHWModules()`. If `node` corresponds to "applications" the application names are retrieved with `NamesOfApplication()`. Finally if `node` corresponds to "misc" the function `getMiscNodes()` is called.

Figure 7.1 An example of the GoForms version.

7.4 Access variables

This section describes the methods for retrieving information about the access variables in the controller. If the plus sign next to “Access Variables” is clicked the tree expands, see Figure 7.1. The present access variables in this example are “selected”, “online”, “frequency” and “initValue”. If more information is desired the hyperlink “More information” can be clicked.

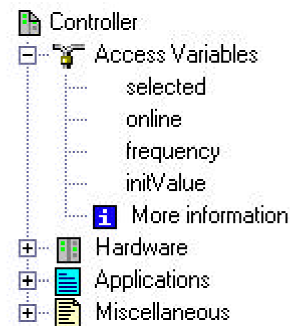


Figure 7.1 Access variables sub tree.

7.4.1 Implementation

In the file “accVar.c” there are three functions, `NamesOfAccVar()`, `getAccVar()` and `AccVarForm()`.

The function `NamesOfAccVar()` is used in the Goforms-version of the start page. It takes the string that is to be filled with the HTML-code as an in-parameter. The function goes through the list of access variables and adds HTML tags and the names of the access variables to the string. It goes through the list with a system object called `pVarAccessItem`. It is through this object the names are accessible. In the end the string is returned.

The function `getAccVar()` is called from the JavaScript version of the start page. This function also goes through the list of access variables, but does not add pieces to a string. Instead it uses `websWrite()` directly and sends the names to the calling browser.

These two functions only bring out the names of the access variables, no more detailed information. If the user clicks the hyperlink “More information” this is done and `AccVarForm()` is used. If the hyperlink is clicked a new page will appear in the right frame in the browser see Figure 7.1.

Name	Type	Value	Writeable	NewValue
selected	Bool	0	Yes	On ○ Off ○
online	Bool	1	Yes	On ○ Off ○
frequency	DInt	2000	Yes	
initValue	Bool	1	No	

Submit!

Reset

Figure 7.1 The access variables web page.

For every access variable their type, value and if the value can be changed (writeable) is shown. If the value can be changed the most right column is also filled. Depending on the variables type that column looks differently. If it e.g., is a variable of boolean type a radiobutton is shown and if it is of integer type a textfield is shown. If the user changes a value and presses the submit button the `AccVarForm()` function will be called again. The `AccVarForm()` function has the following structure:

```
1. Get the first access variable and create a string.
    2. Check if the variable has been changed.
        3. If it is changed, update.
    4. Retrieve the name, type, value and if it is writeable.
    5. Append this information to the string.
6. If there exists more access variables, get next. Go to 2.
7. Send the string to the browser and call websDone().
```

Since it is possible to change the values of the access variables `AccVarForm()` must be able to obtain the new values. GoForms is used to do this. The CGI variables have the same name as the access variables in order to be able to check weather they have been changed or not. If a value of an access variable is changed the new value is written to the controller.

7.5 Hardware

This section describes the methods to retrieve information about the different hardware units or modules that are present in the controller and how it is used by the program. By clicking on the “Hardware” folder the tree will expand and the result may look like Figure 7.1. Each hardware unit is a subfolder that can be expanded further for more information. In front of the hardware unit name there is a number (0-2 for AC 800C), which corresponds to the position of the module on the controller board.

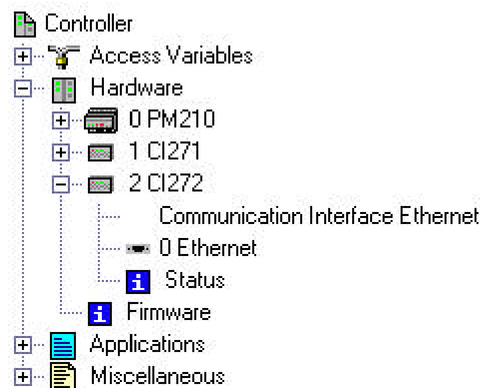


Figure 7.1 Hardware sub tree.

7.5.1 Implementation

In the C-file “Hardware.c” there are two functions: `getHWModules()` and `getHWUnits()`. These functions are used to retrieve information and create code for the start pages.

The function `getHWModules()` is used in the GoForms-version of start page. Figure 7.1 shows the calling sequence. First (1) the function `menyLayout()` is called, this is done when a request for the start pages occurs. If hardware is requested the function `getHWModules()`(2) is called, it takes a character string as an in-parameter. It will in turn call `getModuleTypes()`(3) that retrieves the modules and store them in a local array,

ModuleTypes[] (4). After that the HTML-code is generated in getHWModules() (5) and then returned to menyLayout()(6) and finally the function WebsWrite()(7) is called.

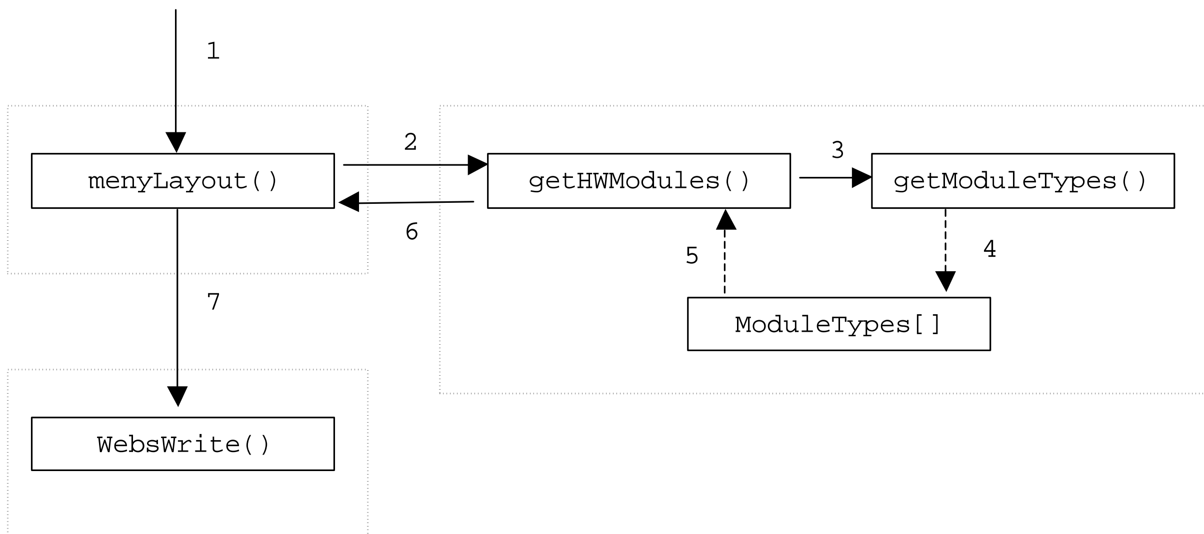


Figure 7.1 The getHWModules() function.

The function getHWUnits() used in the JavaScript version of the start page works in a similar way. The calling sequence for the JavaScript is shown in Figure 7.2. When a request for the start page occurs the function JavaPage()(1) is called. It calls the function getHWUnits() (2) which calls getModuleTypes() (3) (the same function as in the GoForms version). The JavaScript code is generated in getHWUnits()and sent with the function websWrite()(4).

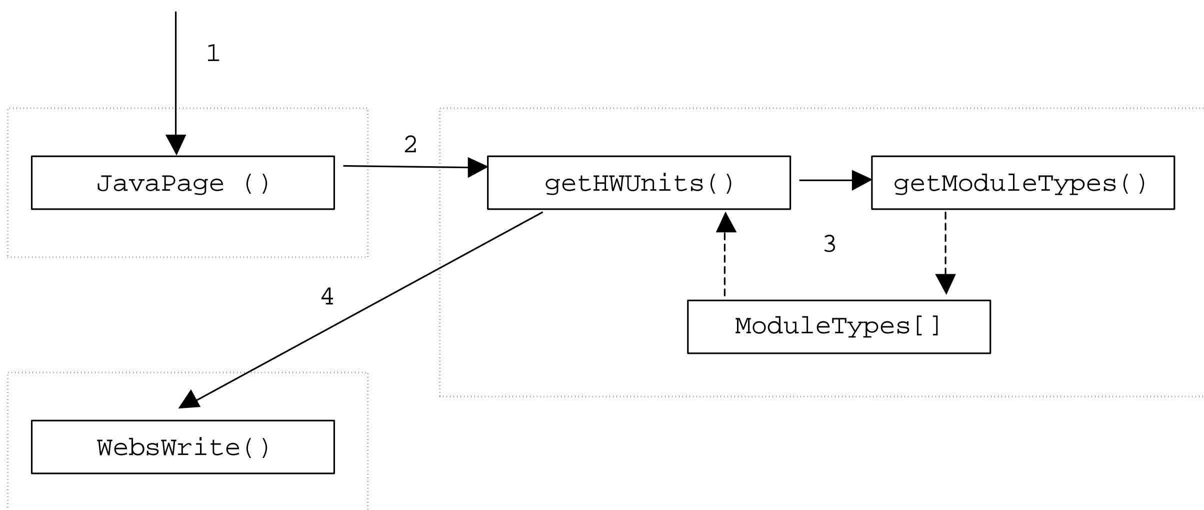


Figure 7.2 The getHWUnits() function

The information that can be retrieved from getModuleTypes()is only a module id number and its position. With a function called ModuleTypeToStrig()it is possible to convert the id number to a readable string (e.g. PM210, CI272). This is the amount of information that is possible to get from the system. But in the functions getHWUnits()and getHWModules() there are methods to generate more information. Because the modules physical layout never changes, it is possible to append more information to the module name.

For example a CI271-module (serial communication) always have two com-ports, and a CI272-module (Ethernet) always have one com-port. To every module extra information is appended containing an explaining text and which hardware it has. In some cases where more information can be obtained a hyperlink to another page is also appended, for example Ethernet status see Figure 7.1. Under the "Hardware" folder in the tree there is also a hyperlink called "Firmware". If this link is pressed a new page will appear in the right frame, with information about the modules and the downloaded firmware.

7.5.2 Firmware

By clicking on the "Firmware" hyperlink in the "Hardware" folder (Figure 7.1) a new page will appear in the right frame in the browser see Figure 7.1.

Module	Firmware	Date	Version	Major	Minor	Subversion	Working Version	Upgradeable
PM210	FW210	2001-09-20	0.43.13.6	-	-	-	-	Yes
PM210	OMEGA	2001-09-17	1.1.13.10	-	-	-	-	No
PM210	HW Config	1999-09-06	1.4	1	4	-	-	No
CI271	CI271	1999-12-17	1.0.4.0	1	0	4	-	Yes
CI272	FWCPUCIE	2000-01-20	1.1.0.0	-	-	-	-	Yes

Figure 7.1 The firmware web page.

In the C-file "Firmware.c" there are two functions `firmwareXML()` and `firmwareXSL()`. When a request for the firmware page occurs the function `firmwareXML()` is called. This function in turn calls `getModuleTypes()` (Figure 7.1) and the system function `GetAllFirmwareinfos()` which returns the firmware for all modules on the controller. Further it will create and send a XML page. The XML document can look like this:

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="firmware.xsl"?>
<FirmwareInfo>
      <Firmware>
            <Module>PM210</Module>
            <Firmware>FW210</Firmware>
            <Date>2001-09-20</Date>
            <Version>0.43.13.6</Version>
            <Major>-</Major>
            <Minor>-</Minor>
            <Subversion>-</Subversion>
            <WorkingVersion>-</WorkingVersion>
            <Upgrade>Yes</Upgrade>
      </Firmware>
      <Firmware>
            <Module>PM210</Module>
            <Firmware>OMEGA</Firmware>
            <Date>2001-09-17</Date>
            <Version>1.1.13.10</Version>
            <Major>-</Major>
            <Minor>-</Minor>
            <Subversion>-</Subversion>
            <WorkingVersion>-</WorkingVersion>
            <Upgrade>No</Upgrade>
      </Firmware>
            ...
      <Firmware>
            <Module>CI272</Module>
            <Firmware>FWCPUCIE</Firmware>
```

```
                <Date>2000-01-20</Date>
                <Version>1.1.0.0</Version>
                <Major>-</Major>
                <Minor>-</Minor>
                <Subversion>-</Subversion>
                <WorkingVersion>-</WorkingVersion>
                <Upgrade>Yes</Upgrade>
        </Firmware>
</FirmwareInfo>
```

The XML documents root element is called “FirmwareInfo” and it includes a number of “Firmware” elements, which in turn includes nine information elements (Module, Firmware, Date etc.). As seen in Figure 7.1 each row in the table corresponds to a “Firmware” element.

In the xml-stylesheet definition (second row) the xsl-stylesheet “firmware.xsl” is defined. The browser will automatically request this page from the server. When this page is requested the server will call the function `firmwareXSL()`. This function creates a XSLT document for transforming the XML-page into a viewable web page (Figure 7.1).

The XSLT document looks like this:

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<HTML>
<HEAD>
        <STYLE TYPE="text/css">
                TABLE {font-family: MS Sans Serif }
                TH,TD {font-size: 8pt}
        </STYLE>
</HEAD>
<BODY BGCOLOR="white">
<h3>Firmware Information</h3>
<TABLE BORDER="1" CELLSPACING="1" BGCOLOR="#99CCFF">
        <TR>
                <TH>Module</TH>
                <TH>Firmware</TH>
                <TH>Date</TH>
                <TH>Version</TH>
                <TH>Major</TH>
                <TH>Minor</TH>
                <TH>Subversion</TH>
                <TH>Working Version</TH>
                <TH>Upgradeable</TH>
        </TR>
        <xsl:for-each select="FirmwareInfo/Firmware">
        <TR>
                <TD><xsl:value-of select="Module"/></TD>
                <TD><xsl:value-of select="Firmware"/></TD>
                <TD><xsl:value-of select="Date"/></TD>
                <TD><xsl:value-of select="Version"/></TD>
                <TD><xsl:value-of select="Major"/></TD>
                <TD><xsl:value-of select="Minor"/></TD>
                <TD><xsl:value-of select="Subversion"/></TD>
                <TD><xsl:value-of select="WorkingVersion"/></TD>
                <TD><xsl:value-of select="Upgrade"/></TD>
        </TR>
        </xsl:for-each>
</TABLE>
</BODY>
```

```
</HTML>
</xsl:template>
</xsl:stylesheet>
```

This document converts the root element “FirmwareInfo” to a table (Figure 7.1). The first row in the table is constructed with header cells (<TH>) and then the other rows are generated. This is done with the tag <xsl:for-each select="FirmwareInfo/Firmware">, which iterates through the root element and for each “Firmware” element it creates a new table row (<TR>). To extract the data from the different elements (Module, Firmware etc.) the tag <xsl:value-of select=""/> is used. The tag <TD> creates a new table cell.

When creating these pages the functions in the HTML- and XML generators are used. The XSLT page is a static page, while the XML page is dynamic. This means that it is a template page and filled according to the firmware that has been downloaded to the controller.

7.5.3 Ethernet

In Figure 7.1 the module “CI272” has been expanded and an information text shows that it is an “Ethernet module”, how many com-ports it has and also a “Status” hyperlink. When clicking on the hyperlink a new page will appear in the right frame, Figure 7.1. The information that is available is status for the Ethernet channel, for example number of transmitted packets and missed packets.

Status	Active
Transmitted packet	182
TX packet status	1
Free Receive Buffers	18
Free Transmit Buffers	6
Received broadcasts (TCP/IP)	11100
Rejected broadcasts due to not wanted IP-type	247
Received packets (TCP/IP)	191
Rejected packets due to not wanted IP-type	0
Received packets (OSI)	0
Rejected due to OSI-channel not open	912
Rejected due to LSAP mismatch	0
Rejected due to ieee802.3 length error	0
Missed packet	1
CRC error	0

Figure 7.1 The Ethernet status page

This page uses the GoForms functionality to be able to tell the function `ethernetXML()` which position on the controller the module has (for AC 800C these are 1 or 2). The page uses XML and XSL documents. The function `ethernetXML()` is responsible for retrieving the different status information from the system and creating the XML page. Before creating the XML document an error code from the system is checked, if an error occurred a web page with the message: "*Error when trying to obtain Ethernet status*" will be sent to the browser. If there did not occur any errors an XML page will be generated.

Example of the Ethernet XML code:

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="../ethernet.xsl"?>
<EthernetInfo>
      <Info>
```

```
            <Text>Status</Text>
            <Value>Active</Value>
      </Info>
      <Info>
            <Text>Transmitted packet</Text>
            <Value>182</Value>
      </Info>
      <Info>
            <Text>TX packet status</Text>
            <Value>1</Value>
      </Info>
      <Info>
            <Text>Free Receive Buffers</Text>
            <Value>18</Value>
      </Info>
            .
            .
            .
      </Info>
      <Info>
            <Text>Receive Buffers Exhausted</Text>
            <Value>0</Value>
      </Info>
</EthernetInfo>
```

The XSL document used is called ‘ethernet.xsl’ and corresponds with the function `ethernetXSL()`. This function creates a style sheet that look like this:

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<HTML>
      <HEAD>
            <STYLE TYPE="text/css">
                  TABLE,INPUT {font-family: MS Sans Serif}
                  TH,TD,INPUT {font-size: 8pt}
            </STYLE>
      </HEAD>
      <BODY BGCOLOR="white">
      <h3>Ethernet Status</h3>
      <TABLE BORDER="1" CELLSPACING="1" BGCOLOR="#99CCFF">
      <xsl:for-each select=" EthernetInfo/Info">
      <TR>
            <TD>
                  <xsl:value-of select="Text" />
            </TD>
            <TD ALIGN="RIGHT">
                  <xsl:value-of select="Value" />
            </TD>
      </TR>
      </xsl:for-each>
      </TABLE>
      </BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>
```

The transformation of the Ethernet XML document works in a similar way as the transformation of the firmware XML document (section 7.5.2). A table with header cells are first created and then each row with an explaining text and the corresponding data.

7.6 Applications

This section describes the methods for retrieving information about the executing applications in the controller. If the plus sign next to "Applications" is clicked the tree expands see Figure 7.1. In this example the applications are called "Application_1" and "Tank_Process_". If more information is desired the hyperlink "More information" can be clicked.

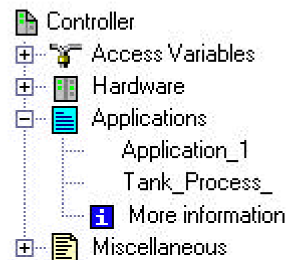


Figure 7.1 Application sub tree.

7.6.1 Implementation

In the file "Application.c" there are three functions, `NamesOfApplication()`, `GetApplication()`, and `applTable()`.

`NamesOfApplication()` is used in the Goforms version of the start page. It takes a string as an in-parameter. The application names are stored as strings in the controller and these can be obtained through system calls. When the names are obtained these and HTML tags are appended to the string. In the end the string is returned.

`GetApplication()` is used in the JavaScript version of the start page. This function also obtains the application names through system calls, but does not append these to a string. Instead `websWrite()` is used and the names are directly send to the browser.

These two functions only retrieve the names of the applications. If the user clicks the hyperlink "More information" the function `applTable()` is called. Then a new page appears in the right frame in the browser, see Figure 7.1.

Application Name	Task Name	Priority	CycleTime	Cyclic
Application_1	Normal	2	250	Yes
Tank_Process_	Fast	1	50	Yes

Figure 7.1 The application web page.

For every application its task connection, priority, cycle time, and cyclic are presented. Cyclic is a boolean variable that indicates if the application executes cyclic. `applTable()` goes through a list of program instances that is part of the ABB source code. For every program instance the information (application name, task connection, priority, etc.) is sent to the browser.

7.7 Miscellaneous

This section describes two methods to retrieve heap usage information and the controller log. If the plus sign next to "Miscellaneous" is clicked the tree expands, see Figure 7.1.

Figure 7.1 Miscellaneous sub tree.

7.7.1 Heap information

This page shows heap information, such as total heap size, used heap, and the amount of free heap memory. It also shows the maximum heap usage since the start and since the reset. The functions used for retrieving the heap information are `GetHeapUsage()`, `GetFreeHeapSize()`, `ResetPeakSizeValue()`, and `GetPeakSizeValues()`. These functions are a part of the ABB implemented heap functionality. Figure 7.1 shows how the information is presented. By clicking on the button "`Reset!`" the function `ResetPeakSizeValue()` is called.

Information:	Nbr of bits:
Total heap size:	1932904
Used heap size:	498768
Free heap size:	1434136
Max memory allocated (from start):	499300
Max memory allocated (from reset):	498768

Options:

Press the button to reset the heap peak counter [Reset!]

Press the button to set a setpoint [Set]

Press the button to calculate heap usage [Calculate]

Statistics:

Current heap size:	498768
Setpoint value (bits):	498768
Used heap memory since setpoint :	

Figure 7.1 Heap information

On this page there are also a tool for checking the controller program for memory leaks. If the button "`Set`" is pressed, the current heap usage will be stored. After performing the operations that should be checked for memory leaks the button "`Calculate`" should be pressed. The program will again check the heap usage and compare it with the stored value. The difference between the current value and the stored value is calculated and presented.

7.7.2 Controller log

The controller log is a log file that is stored in the controller. Since the controller does not contain any file system the log file is not a file, but an array. The array can store 16348 characters and new characters are pushed in, i.e., the oldest characters are lost when new ones are inserted. The controller log contains all product printouts to inform about warnings or errors. For example, if a task is aborted or if an overload in the controller occurs, then this is written to the controller log. If the controller crashes the controller log is a useful tool to investigate why the crash occurred. The controller log is available after a controller crash, i.e., the controller log memory is not destroyed in a crash. The controller log is sent to the browser as plain text, not as text within HTML tags. In order to make the browser understand the received information the HTTP-header instruction `Content-Type` is sent as:

```
Content-Type: text/plain
```

7.8 Summary

The method used for generating web pages is based upon a string that is extended piece by piece. With the HTML generator and the XML generator this can be done in a more structured way. GoForms are used to interact with the web server. Two different start pages exist, one based on GoForms and one based on JavaScript. Four main categories are available in the project tree: access variables, hardware, applications, and miscellaneous information.

8 Future developments

There are a number of possibilities to develop the web server further. More information like CPU load or other real-time information could be displayed in the browser. The cycle time and priority of an application could be changed via the browser.
Security is a major part that could be investigated and developed. Perhaps a simple login form with password would be enough, but it can also be possible to have a more advanced security structure. If different user levels were used, the lowest levels were only allowed to read information. The higher levels would be authorized to change the values of, e.g., access variables. The GoAhead® Webserver™ 2.1 contains a structure with different user levels. If the web server can be called through the Internet a more robust security is needed, but more likely the web server can only be called within an intranet and then a more simple security structure will be enough.
If different XSL style sheets could be downloaded to the controller (via Control Builder) some interesting features occurs. With these style sheets the information could be displayed in different ways depending on the calling system. For example one style sheet could be used in a PDA and another in an ordinary PC. Then it is possible to present certain information in a number of ways. The user can choose to present the information in a way he wants.
Another interesting application is the use of SOAP. For example, instead of calling each controller separately a "gateway" controller could be used. This controller could in turn call the other controllers on the network and retrieve the web related information using SOAP messages.

9 Summary and conclusions

This project has resulted in a web server that is intended to run in the ABB controllers AC 800M and AC 800C. It is a part of the ABB concept Control^{IT}. The web server runs under the operating system pSOS. The controllers only contain RAM and FLASH memory and lacks file system. Therefore no web pages can be statically stored, instead the web pages must be dynamically generated. As a result of that a set of functions that generate HTML and XML code have been developed.

A comparison between different web servers resulted in that the GoAhead web server was used as the base web server. It is from this the further developments and adaptations to the ABB system have been done.

When a user calls the web server a simple and easy overviewed structure of the controller configuration is displayed in the browser. Through hyperlinks more detailed information can be obtained.

In the report different Internet technologies like HTTP, HTML, XML, SOAP, CGI script, and JavaScript have been described.

10 User guide

This chapter contains an overview of how to use the web page. A tutorial goes trough how information about a controller is obtained using the JavaScript version. It is an example with one specific controller configuration, of course a controller can have many different configurations.

10.1 Tutorial

First a web browser must be started. In this example Microsoft Internet Explorer 5.5 have been used. When the controller's IP number is entered in the address bar the start page is presented, see Figure 10.1.

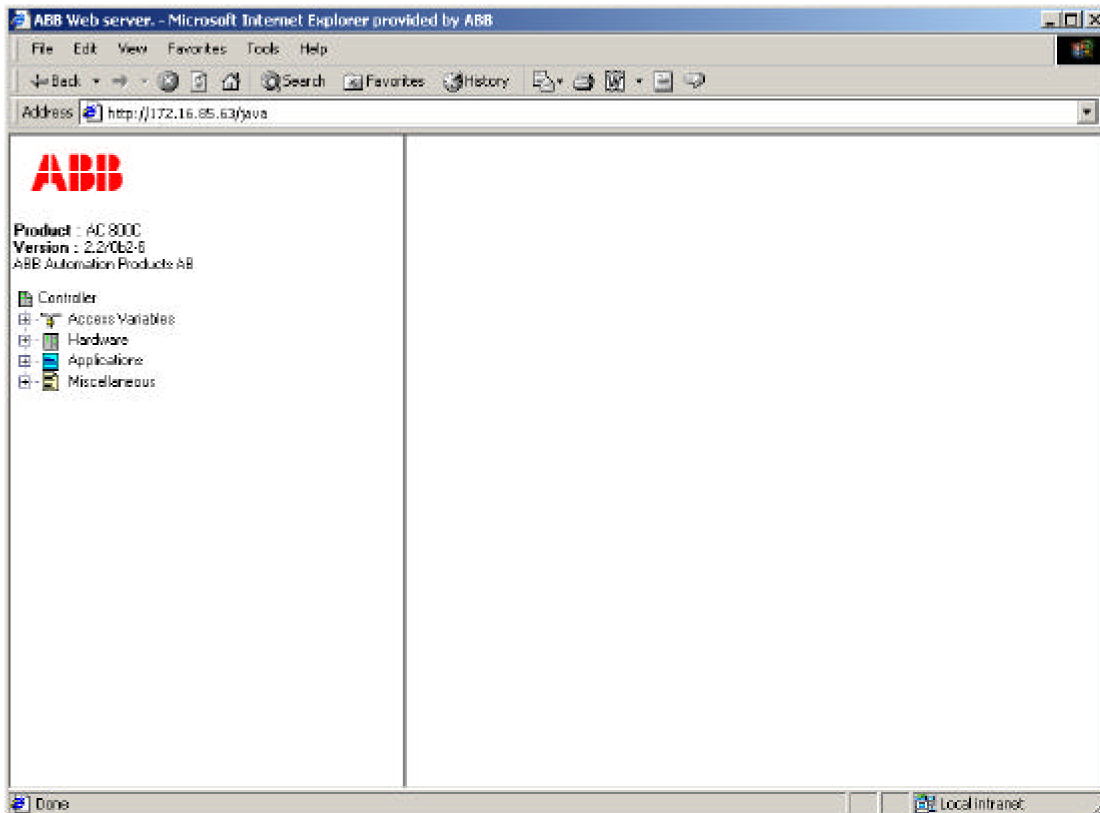


Figure 10.1 Start page.

Information about which controller and its version is presented. In this example the controller is "AC 800C" and the version "2.2/0b2-6". Underneath, a tree structure is shown. In the tree there are four main parts: Access Variables, Hardware, Applications, and Miscellaneous. To the left of these four keywords plus signs (+) are placed. If a plus sign is clicked the tree expands and more information about the clicked category is presented. In Figure 10.2 the plus sign next to "Access Variables" has been clicked. Four access variables exist in the controller, "selected", "online", "frequency" and "initValue". It is only the names of the access variables that are presented in the tree.

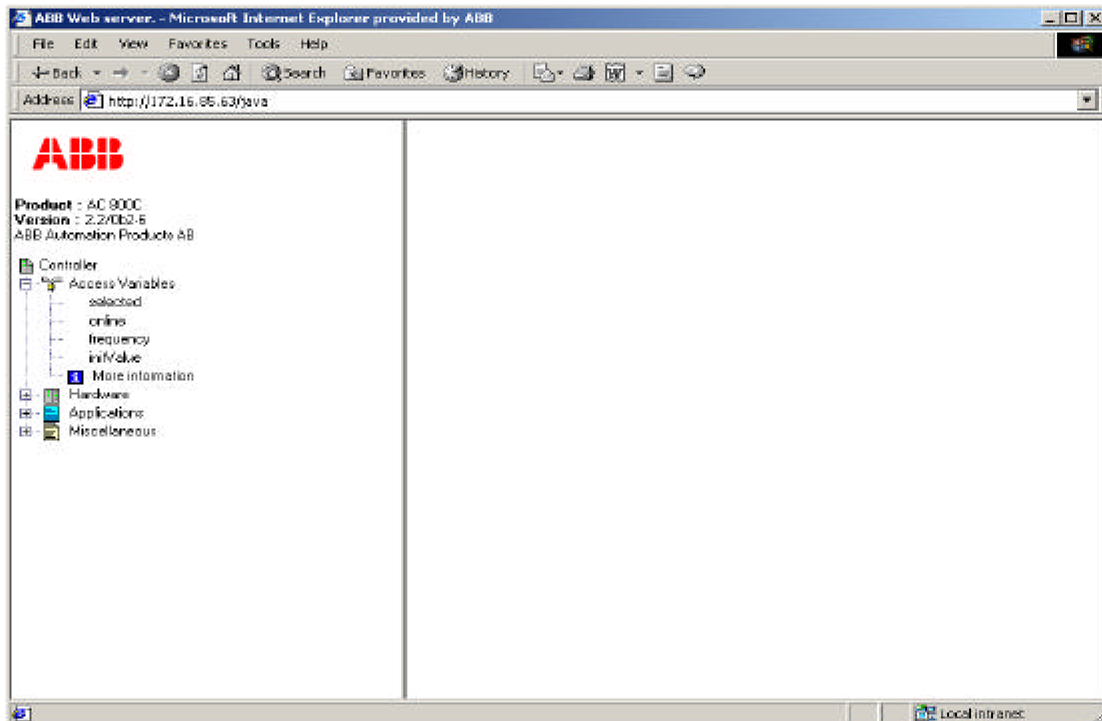


Figure 10.2 The plus sign next to Access Variables has been clicked.

If more information about the access variables is desired the link "More information" must be clicked. In Figure 10.3 this has been done. A table containing information about the access variables occurs in the right frame. Every access variables name, type, value and if it is writeable are presented. If it is writeable a column there the new value can be written is placed as the rightmost column in the table. In this example two different data types are used, boolean (Bool) and double integer (DInt).

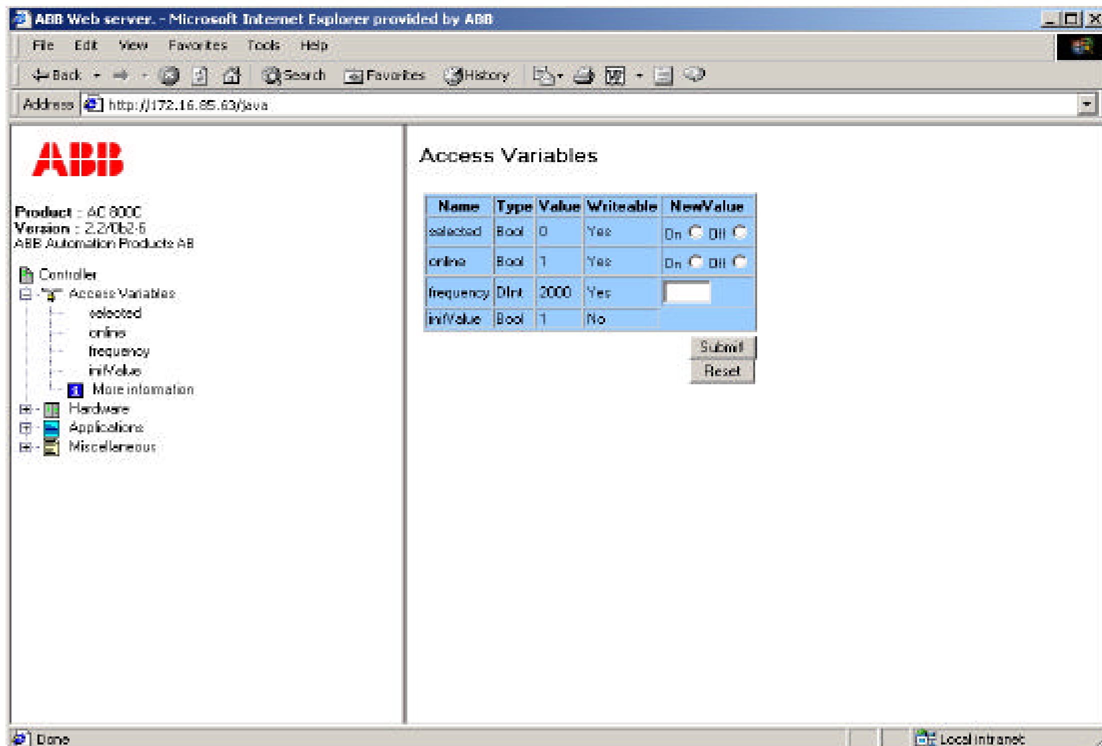


Figure 10.3 More information under Access Variables has been clicked.

If the variable is of type boolean a radio button is placed in the column “New Value” and if it is a double integer a text field is placed there. If the user wishes to change the value, the value must be entered and then the “Submit” button must be pressed.
The next category is “Hardware”. When the plus sign next to that is clicked the present hardware modules occurs under “Hardware”. A “Firmware” link is also presented. When that link also been clicked the browser looks like Figure 10.4. The firmware information consists of the corresponding modules firmware name, date, version and if it is upgradeable. More advanced information like “Major”, “Minor”, “Subversion” and “Working Version” are also presented.

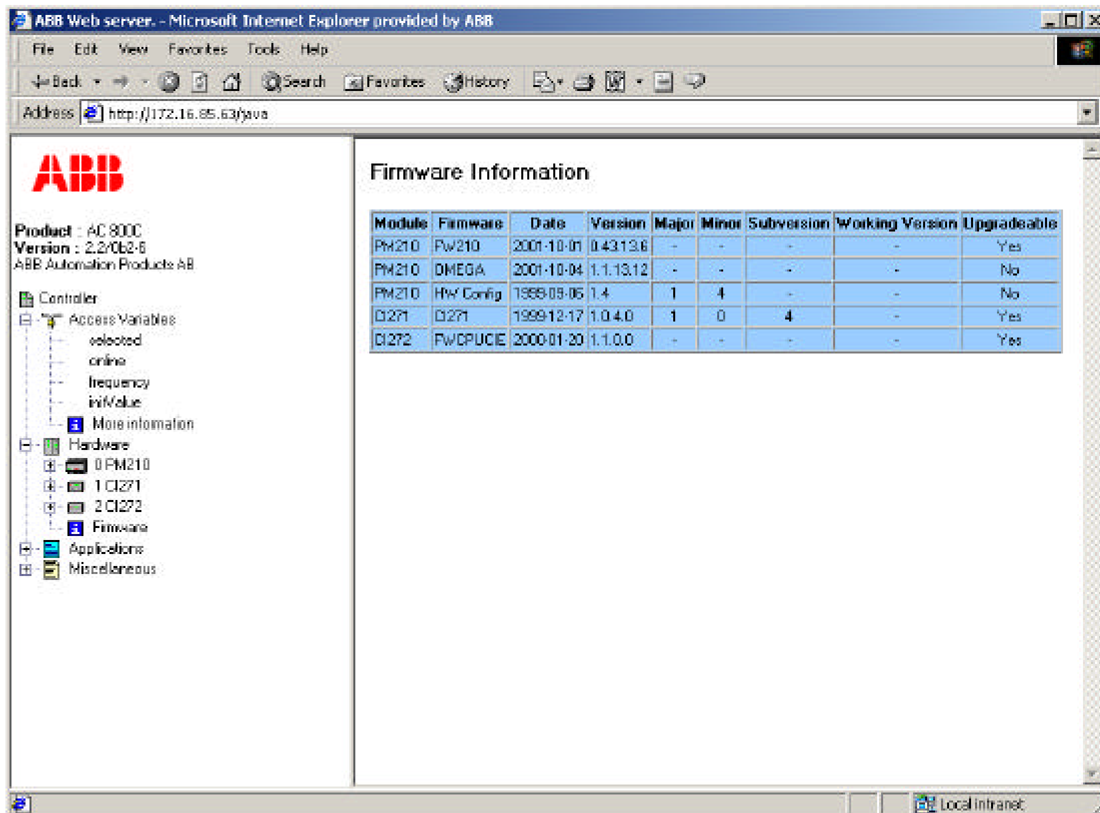


Figure 10.4 The plus sign next to Hardware and the Firmware link has been clicked.

The hardware modules present in this example are “PM210”, “CI271” and “CI272”. PM210 is the controller AC 800C’s CPU unit, CI271 is a serial module and CI272 is an Ethernet module. The numbers in front of the modules (0, 1 and 2) are the module’s place on the controller. There are plus signs to the left of the modules, which indicate that there is more information about the modules to display in the tree. In Figure 10.5 the plus signs next to CI271 and CI272 have been pressed. When the tree expands a more detailed explanation to CI271 and CI272 is shown directly below these abbreviations. CI271 means “Communication interface RS232” (serial) and CI272 means “Communication interface Ethernet”. Below “Communication interface RS232”, is written “0 Com” and “1 Com”. This means that on the module CI271 it exists two com-ports, 0 and 1. Below “Communication interface Ethernet” is written “0 Ethernet” which means that on the Ethernet module it exists one Ethernet connection. Below “0 Ethernet” there is a status link. If the link is clicked information about the Ethernet connection is displayed in the right frame, see Figure 10.5. The information consists of Transmitted packets, Missed packets and Received broadcasts, etc.

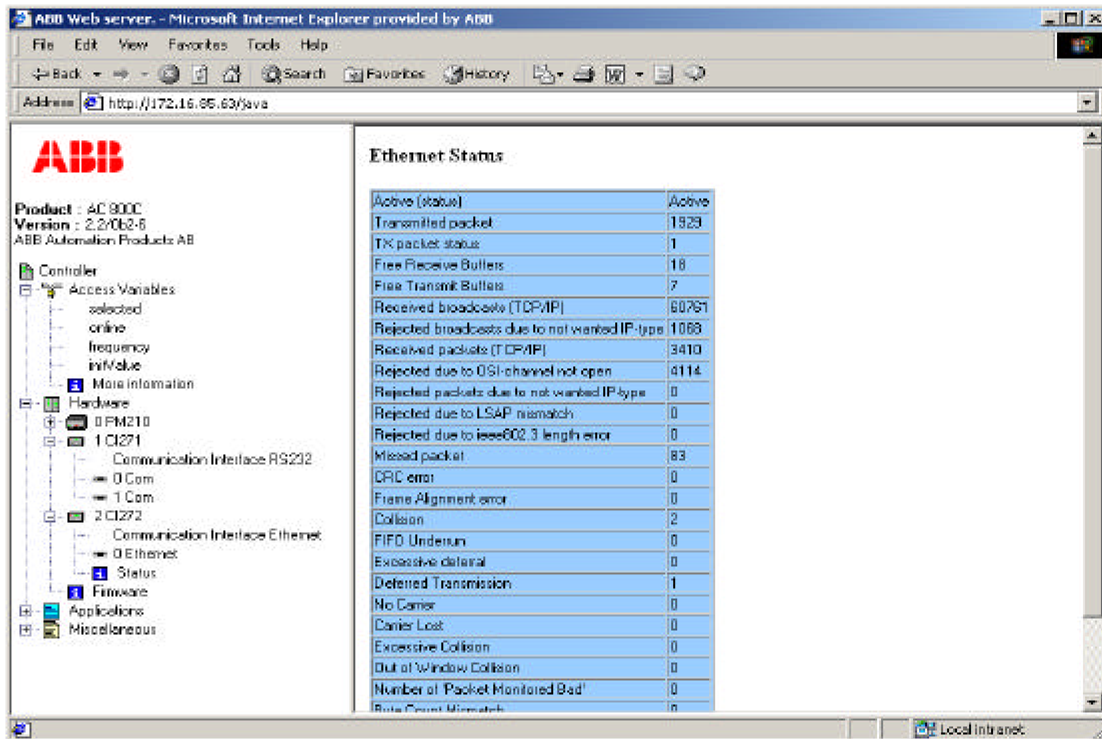


Figure 10.5 The plus signs next to CI271 and CI272 and the status link under CI272 have been pressed.

The next category is "Applications". If the plus sign next to that is pressed the names of the executing applications is presented below. The applications in this example are called "Application_1" and "Tank_Process_". If the link "More information" is clicked, a table appears in the right frame, see Figure 10.6. Every applications task connection, priority, cycle time, and if it executes cyclic are presented.

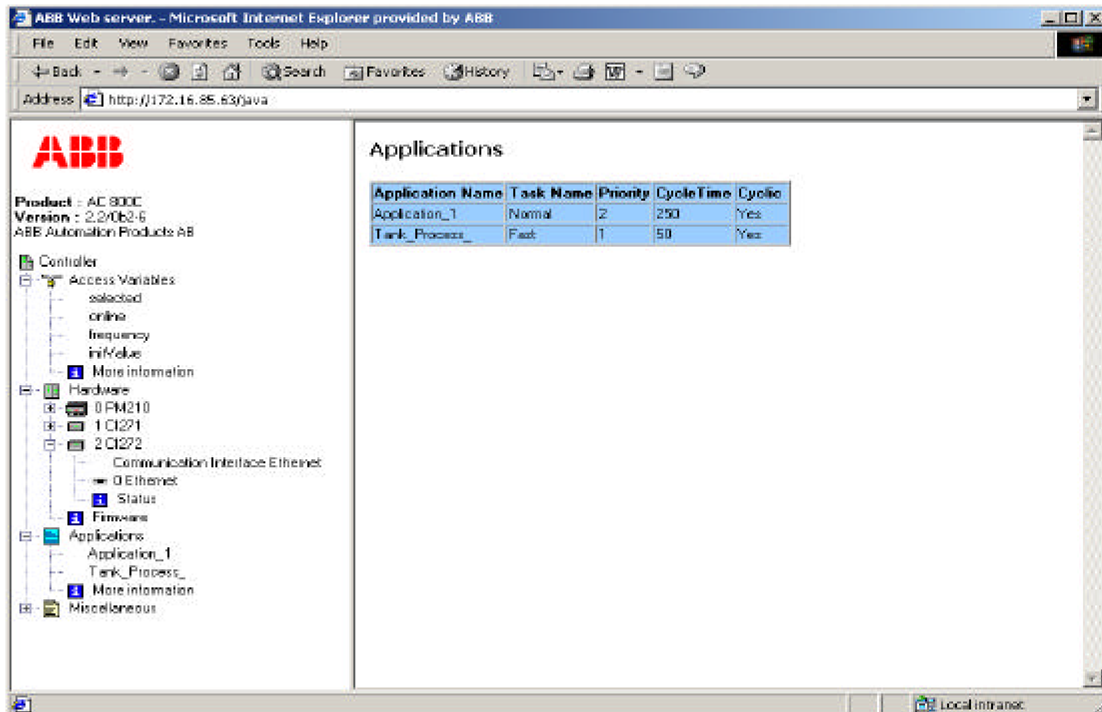


Figure 10.6 The plus sign next to Applications and "More information" have been pressed.

The last category is “Miscellaneous”. It has two sub categories, “Heap information” and “Controller log”. In Figure 10.7 the “Heap information” link has been clicked and a table containing heap data are presented in the right frame. It is possible to reset the heap peak counter by pressing the “Reset!” button, i.e., “Max memory allocated (from reset)” is recalculated. On this page there are also a tool for checking the controller program for memory leaks. If the button “Set” is pressed, the current heap usage will be stored. After performing the operations that should be checked for memory leaks the button “Calculate” should be pressed. The program will again check the heap usage and compare it with the stored value. The difference between the current value and the stored value is calculated and presented.

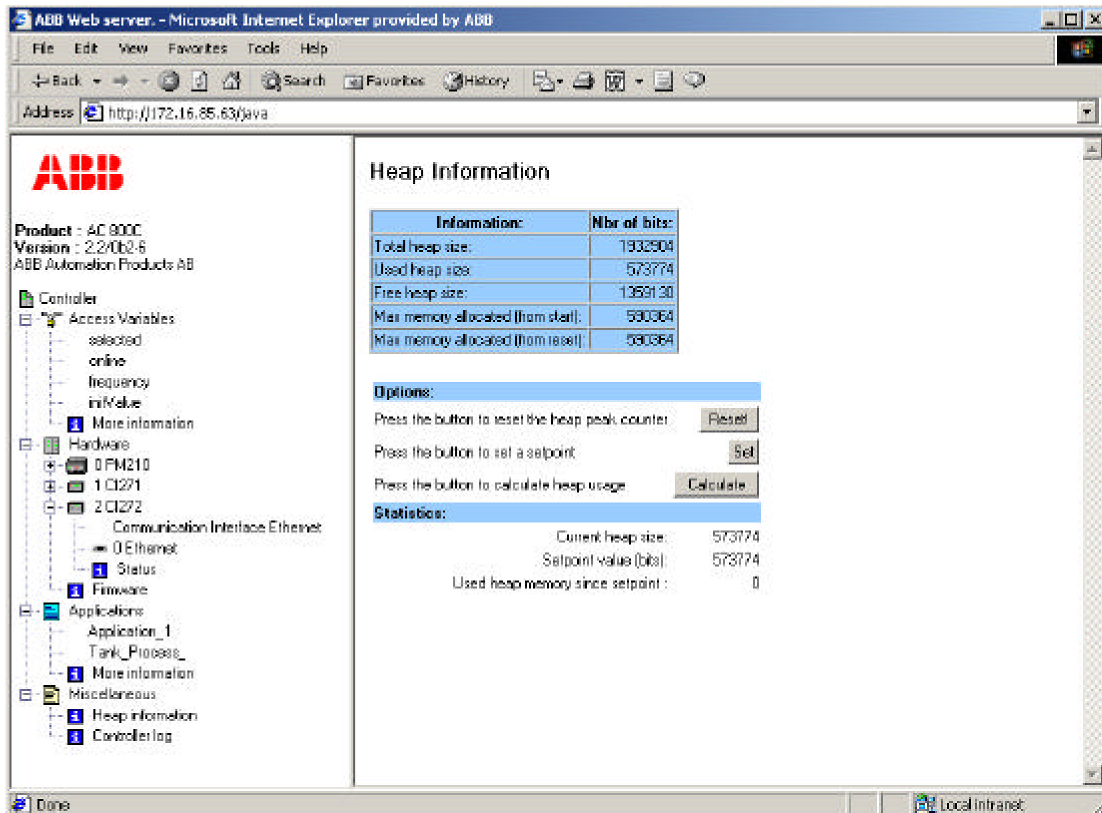


Figure 10.7 The plus sign next to Miscellaneous and the Heap information link have been clicked.

In Figure 10.8 the “Controller log” link has been clicked and the controller log is presented in the right frame. The controller log contains all product printouts to inform about warnings or errors. For example, if a task is aborted or if an overload in the controller occurs, then this is written to the controller log. If the controller crashes the controller log is a useful tool to investigate why the crash occurred. The controller log is available after a controller crash, i.e., the controller log memory is not destroyed in a crash.

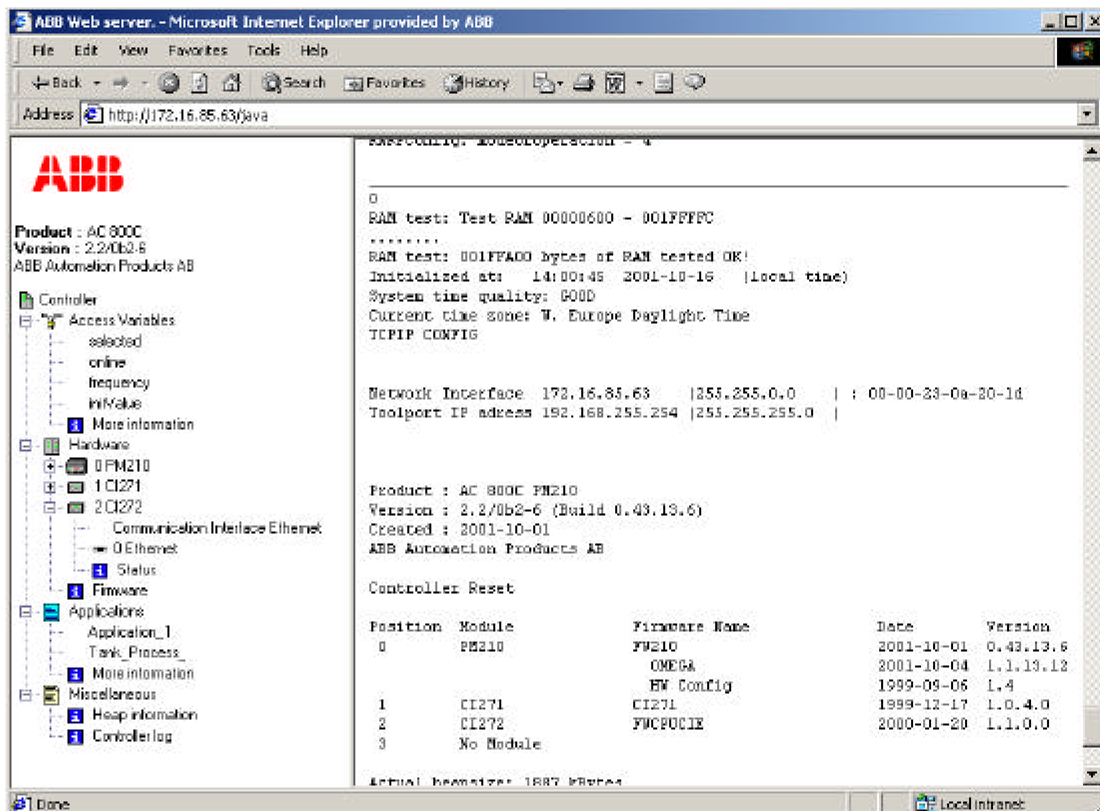


Figure 10.8 The plus sign next to Miscellaneous and the Controller log link have been clicked.

11 References

[1] ABB Automation Products AB, Control Builder Beginner's handbook, October 2001.

[2] Wind River Systems, www.windriver.com.

[3] W. Richard Stevens, "TCP/IP Illustrated, Volume 3", Addison Wesley, July 1996.

[4] World Wide Web Consortium, "Extensible Markup Language (XML) 1.0 (Second Edition)", W3C Recommendation 6 October 2000, www.w3c.org.

[5] Birbeck Mark, Jon Duckett, Oli Gauti Gudmundsson, Pete Kobak, Evan Lenz, Steve Livingstone, Daniel Marcus, Stephen Mohr, Jonathan Pinnock, Keith Visco, Andrew Watt, Kevin Williams, Zoran Zaev, Nikola Ozu, "Professional XML 2nd Edition", Wrox Press Ltd, May 2001.

[6] World Wide Web Consortium, "XSL Transformation (XSLT) Version 1.1", W3C Working Draft 24 August 2001, www.w3c.org.

[7] CGI Programming 101, www.cgi101.com

[8] Netscape, www.netscape.com

[9] GoAhead Software Inc, www.goahead.com.

[10] ABB Automation Products AB, Functional Specification: Atlas Execution Model, LACB-9912-05 (05), September 2001.

[11] Integrated Systems Inc., pSOSystem System Calls, February 1996.

[12] OmenSoft, omensoft.home.ml.org.