Web Server for Embedded Systems

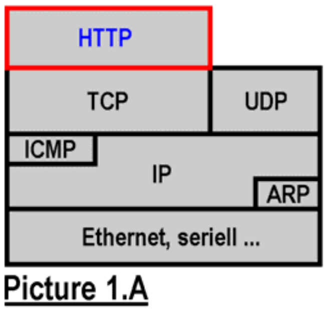
After the "everybody-in-the-Internet-wave" now obviously follows the "everything-in-the-Internet-wave". The most coffee, vending and washing machines are still not available about the worldwide net. However the embedded Internet integration for remote maintenance and diagnostic as well as the so-called M2M communication is growing with a considerable speed rate.

Just the remote maintenance and diagnostic of components and systems by Web browsers via the Internet, or a local Intranet has a very high weight for many development projects. In numerous development departments people work on completely Web based configurations and services for embedded systems. The remaining days of the classic user interface made by a small LC-display with front panel and a few function keys are over. Through future evolutions in the field of the mobile Internet, Bluetooth-based *PAN*s (Personal Area Network's) and the rapidly growing *M2M* communication (M2M=Machine-to-Machine) a further innovating advance is to be expected.

The central function unit to get access on an embedded system via Web browser is the Web server. Such Web servers bring the desired HTML pages (HTML=Hyper Text Markup Language) and pictures over the worldwide Internet or a local network to the Web browser. This happens HTTP-based (Hyper Text Transfer Protocol). A TCP/IP protocol stack –that means it is based on sophisticated and established standards– manages the entire communication. Web server (HTTP server) and browser (HTTP client) build TCP/IP-applications. HTTP achieved a phenomenal distribution in the last years. Meanwhile millions of user around the world surf HTTP-based in the *World Wide Web*. Today almost every personal computer offers the necessary assistance for this protocol. This status is valid more and more for embedded systems also. The HTTP spreads up with a fast rate too.

1. TCP/IP-based HTTP as Communication Platform

HTTP is a simple protocol that is based on a TCP/IP protocol stack (picture 1.A). HTTP uses TCP (Transmission Control Protocol). TCP is a relative complex and high-quality protocol to transfer data by the subordinate IP protocol. TCP itself always guarantees a safeguarded connection between two communication partners based on an extensive three-way-handshake procedure. As a result the data transfer via HTTP is always protected. Due to the extensive TCP protocol mechanisms HTTP offers only a low-grade performance.



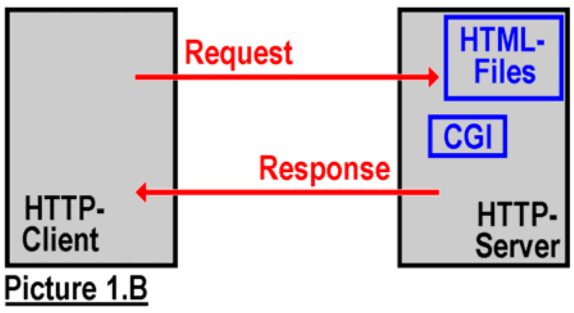


Figure 1: TCP/IP stack and HTTP programming model

HTTP is based on a simple client/server-concept. HTTP server and client communicate via a TCP connection. As default TCP port value the port number 80 will be used. The server works completely passive. He waits for a request (order) of a client. This request normally refers to the transmition of specific HTML documents. This HTML documents possibly have to be generated dynamically by CGI. As result of the requests, the server will answer with a response that usually contains the desired HTML documents among others (picture 1.B).

```
GET /test.htm HTTP/1.1
Accept]: image/gif, image/jpeg, */*
User selling agent: Mozilla/4.0
Host: 192.168.0.1
```

Listing 1.A: HTTP GET-request

```
HTTP/1.1 200 OK
Date: Mon, 06 Dec 1999 20:55:12 GMT
Server: Apache/1.3.6 (Linux)
Content-length: 82
Content-type: text/html

<html>
<head>
<title>Test-Seite</title>
</head>
<body>
Test-Seite
```

```
</body>
</html>
```

Listing 1.B: HTTP response as result of the GET-request from listing 1.A

HTTP requests normally consist of several text lines, which are transmitted to the server by TCP. The listing 1.A shows an example. The first line characterizes the request type (GET), the requested object (/test1.htm) and the used HTTP version (HTTP/1.1). In the second request line the client tells the server, which kind of files it is able to evaluate. The third line includes information about the client-software. The fourth and last line of the request from listing 1.A is used to inform the server about the IP address of the client. In according to the type of request and the used client software there could follow some further lines. As an end of the request a blank line is expected.

The HTTP responses as request answer mostly consist of two parts. At first there is a header of individual lines of text. Then follows a content object (optional). This content object maybe consists of some text lines –in case of a HTML file– or a binary file when a GIF or JPEG image should be transferred. The first line of the header is especially important. It works as status or error message. If an error occurs, only the header or a part of it will be transmitted as answer.

2. Functional principle of a Web Server

Simplified a Web server can be imagined like a special kind of a file server. Picture 2.A shows an overview. The Web server receives a HTTP GET-request from the Web browser. By this request, a specific file is required as answer (see step 1 into picture 2.A). After that, the Web server tries to get access on the file system of the requested computer. Then it attempts to find the desired file (step 2). After the successful search the Web server read the entire file (step 3) and transmit it as an answer (HTTP response comprising of header and content object) to the Web browser (step 4). If the Web server cannot find the appropriate file in the file system, an error message (HTTP response which only contains the header) is simply be send as response to the client.

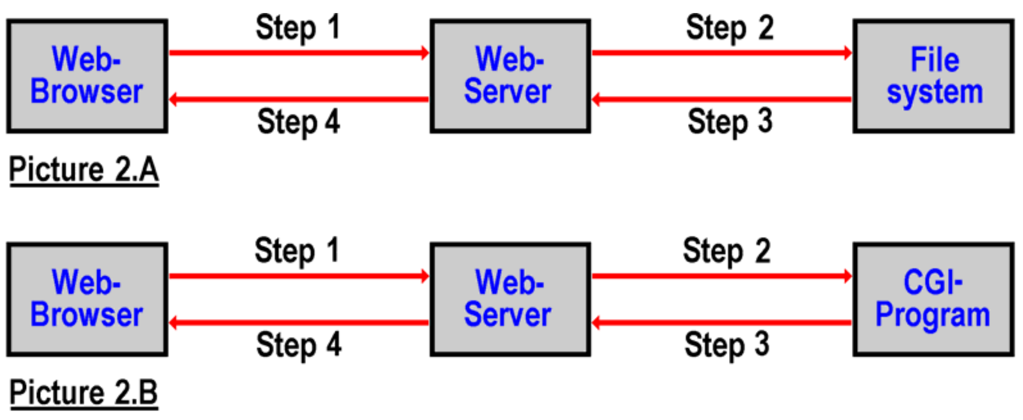


Figure 2: Functional principle from Web server and browser

The web content is build by individual files. The base is build by static files with HTML pages. Within such HTML files there are references to further files embedded –these files are typically pictures in GIF or JPEG format. However, also references to other objects, for example Java-Applets, are possible. After a Web browser has received a HTML file of a Web server, this file will be evaluated and then searched for external references. Now the steps 1 to 4 from picture 2.A will run again for every external reference in order to request the respective file from the corresponding Web server. Please note, that such a reference consists of the name or IP address of a Web server (e.g. "dilnetpc.com"), as well as the name of the desired file (e.g. "picture1.gif"). So virtually every reference can refer to another Web server. In other words, a HTML file could be located on the server "ssv-embedded.de" but the required picture -which is external referenced by this HTML file- is located on the Web server "dilnetpc.com". Finally this (worldwide) networking of separate objects is the cause for the name World Wide Web (WWW). All files, which are required by a Web server, are requested from a browser like the procedure shown on picture 2.A. Normally these files are stored in the file system of the server. The Webmaster has to update these files from time to time.

A further elementary functionality of a Web server is the *Common Gateway Interface* (CGI) -we have mentioned before. Originally this technology is made only for simple forms, which are embedded into HTML pages. The data, resulting from the padding of a form, will be transmitted to a Web server via HTTP-GET or POST-request (see step 1 into picture 2.B). In such a GET- or POST-request the name of the CGI program, which is needed for the evaluation of a form, is fundamentally included. This program has to be on the Web server. Normally the directory "/cgi-bin" is used as storage location.

As result of the GET- or POST-request the Web server starts the CGI program located in the subdirectory "/cgi-bin" and delivers the received data in form of parameters (step 2). The outputs of a CGI program are guided to the Web server (step 3). Then the Web server sends them all as responses to the Web browser (step 4).

3. Dynamic generated HTML Pages

In contradiction to a company Web site server, which informs people about the product program and services by static pages and pictures, an embedded Web server has to supply dynamically generated contents. The embedded Web server will generate the dynamic pages in the moment of the first access by a browser. How else could we check the actual temperature of a system via Internet? Static HTML files are not interesting for an embedded Web server. The most information about the firmware version and service instructions are stored in HTML format. All other tasks are normally made via dynamic generated HTML.

There are two different technologies to generate a specific HTML page in the moment of the request: First the so-called *server-side-scripting* and second the *CGI* programming. At the server-side-scripting, script code is embedded into a HTML page. If required, this code will be carried out on the server (server-sided).

For this, there are numerous script languages available. All these languages are usable inside a HTML-page. In the Linux community PHP is used mostly. The favourite of Microsoft is VBScript. It is also possible to insert Java directly into HTML pages. Sun has named this technology *JSP* (*Java Server Pages*). The HTML page with the script code is statically stored in the file system of the Web server. Before this server file is delivered to the client, a special program replaces the entire script code with dynamic generated standard HTML. The Web browser will not see anything from the script language.

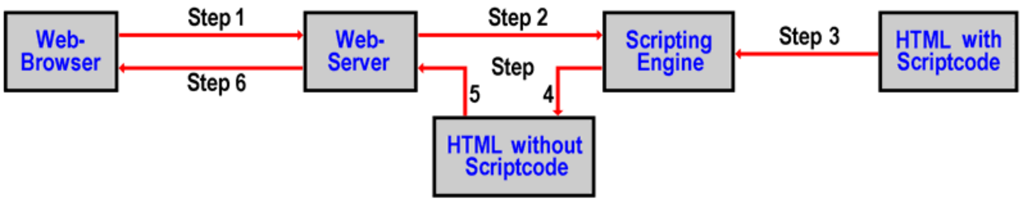


Figure 3: Single steps of the Server-Side-Scripting

Picture 3 shows the single steps of the server-side-scripting. In step 1 the Web browser requests a specific HTML file via HTTP GET-request. The Web server recognizes the specific extension of the desired file (for example *.ASP or *.PHP instead of *.HTM and/or *.HTML) and starts a so-called *scripting engine* (see step 2). This program gets the desired HTML file including the script code from the file system (step 3), carry out the script code and make a new HTML file without script code (step 4). The included script code will be replaced by dynamic generated HTML. This new HTML file will be read by the Web server (step 5) and send to the Web browser (step 6). If a server-sided scripting is supposed to be used by an embedded Web server, so you have to consider the necessary additional resources. A simple example: In order to carry out the embedded PHP code into a HTML page, additional program modules are necessary for the server. A scripting engine together with the embedded Web server has to be stored in the Flash memory chip of an embedded system. Through that, during run time more main memory is required.

4. Web Server running under Linux

Once spoken about Web servers in connection with Linux most people immediately think of *Apache*. After investigations of the Netcraft Survey this program is the mostly used Web server worldwide. *Apache* is an enhancement of the legendary NCSA server. The name *Apache* itself has nothing to do with Red Indians. It is a construct from "A Patchy Server" because the first version was put together from different code and patch files.

Moreover there are numerous other Web servers - even for Linux. Most of this are standing under the GPL (like *Apache*) and can be used license free. A very extensive overview you can find at "http://directory.google.com/". Every Web server has his advantages and disadvantages. Some are developed for specific functions and have very special qualities. Other distinguishes at best through their reaction rate at many simultaneous requests, as well as the variety of their configuration settings. Others are designed to need minimal resources and offer very small setting possibilities, as well as only one connection to a client.

The most important thing by an embedded Web server is the actual resource requirements. Sometimes embedded systems offer only minimal resources, which mostly has to be shared with Linux. Meanwhile there are numerous high-performance 32-bit-386/486-microcontroller or (Strong)ARM-based embedded systems that own just 8 Mbytes RAM and 2 Mbytes Flash-ROM (picture 4). Outgoing from this ROM (Read-only-Memory, i.e. Flash memory chips) a complete Linux, based on a 2.2- or 2.4-Kernel with TCP/IP protocol stack and Web server, will be booted. HTML pages and programs are also stored in the ROM to generate the dynamic Web pages. The space requirements of an embedded system are similar to a little bigger stamp. There it is quite understandable that there is no place for a powerful Web server like *Apache*.

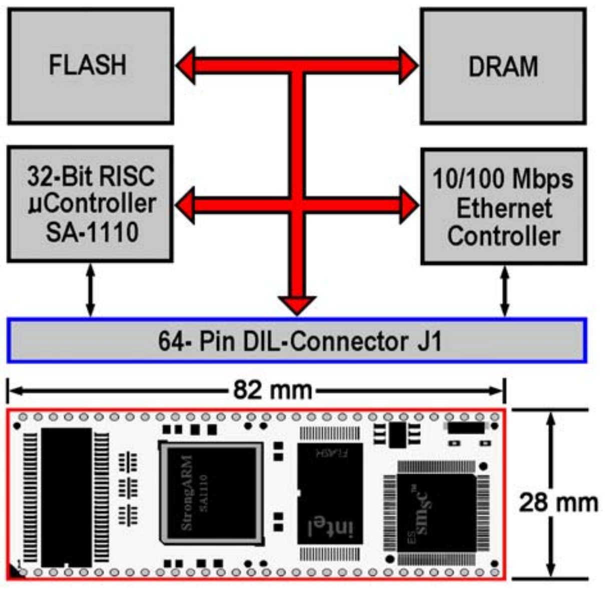

FLASH
DRAM
32-Bit RISC µController SA-1110
10/100 Mbps Ethernet Controller
64- Pin DIL-Connector J1
82 mm
28 mm

Figure 4: Embedded Web Server Module with StrongARM and Linux

But also the capability of an *Apache* is not needed to visualize the counter of a photocopier or the status of a percolator by Web servers and browsers. In most cases a single Web server is quite enough. Two of such representatives are *boa* (www.boa.org) and *thttpd* (www.acme.com). At first, both Web servers are used in connection with embedded systems running under Linux. The configuration settings for *boa* and *thttpd* are poor, but quite enough. By the way, the source code is available to the customer. The practicable binary files for these servers are always smaller than 80 Kbytes and can be integrated in the most embedded systems without problems. For the dynamic generation of HTML pages both servers only offer CGI (Common Gateway Interface) as enlargement. Further technologies, like server-side-includes (SSI) are not available.

The great difference between an embedded Web server and *Apache* is, next to the limited configuration settings, the maximal possible number of simultaneous requests. High performance servers like *Apache* immediately make an own process for every incoming call request of a client. Inside of this process all further steps will then be executed. This requires a very good programming and a lot of free memory resources during run time. But, on the other hand many Web browsers can access such a Web server simultaneously. Embedded Web server like *boa* and *thttpd* work only with one single process. If two users need to get access onto a embedded Web server simultaneously, one of both have to wait a few fractions of a second. But in the environment of the embedded systems that is absolutely justifiable. In this case it is first of all a question of remote maintenance, remote configuration and similar tasks. There are not many simultaneous requests expected.

List of Figures

Listings

Contact

SSV Embedded Systems
Heisterbergallee 72
D-30453 Hannover
Tel. +49-(0)511-40000-0
Fax. +49-(0)511-40000-40
Email: sales@ist1.de
Web: www.ssv-embedded.de

Document History (Sadnp05.Doc)

Revision	Date		Name
1.00	24.05.2002	First Version	KDW