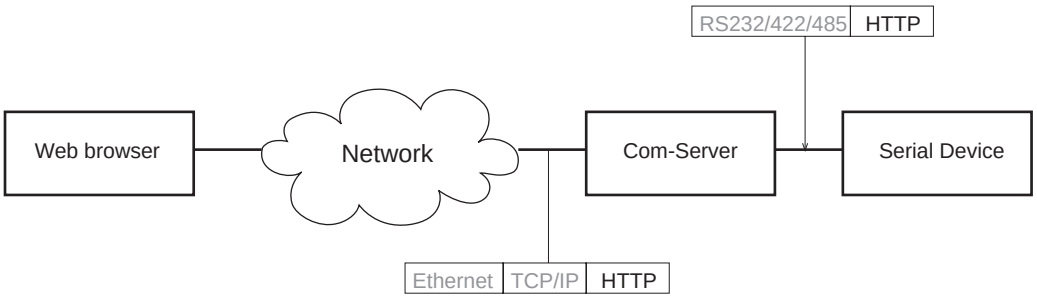
COM servers used as Web servers:

Just a few ASCII strings put RS232 devices on the Web

Applications in the field of remote control and maintenance increasingly use the infrastructure and methodology of the Internet. But many devices in an industrial environment are supplied only with RS232 orf RS422 serial interfaces. Still there is the desire to fully exploit the possibilities of the Internet and be able to display the configuration and operating data of the serial terminal device in any normal Web browser and change the data from there.

All you need to accomplish this is the following:

1. Any usual Web browser.
2. A W&T Com-Server.
3. A network (Internet or Intranet) connecting the Web browser to the Com-Server.
4. A device which makes its data available at the serial port.



This document explains in what form the serial device needs to provide its data and how it has to interpret the requests coming to it on the serial interface.

To represent the data we will of course use the Web standard format HTML (*H*yper*t*ext *M*arkup *L*anguage). But have no fear: HTML is no more than simple ASCII text which has had a few simple strings added for format control. If you don't require a sophisticated layout, this format is quite easy to master. But even if you require a more sophisticated design, no need to rack your brains – there is a great variety of lstandard software for creating Web sites which you can also use for the purpose shown here.

Whereas HTML takes care of data representation, the communication itself is controlled by HTTP (*H*yper*t*ext *T*ransport *P*rotocol) . This is done using short „ASCII headers" which are placed before the actual data.

The division of labor is distributed perfectly between the respective devices:

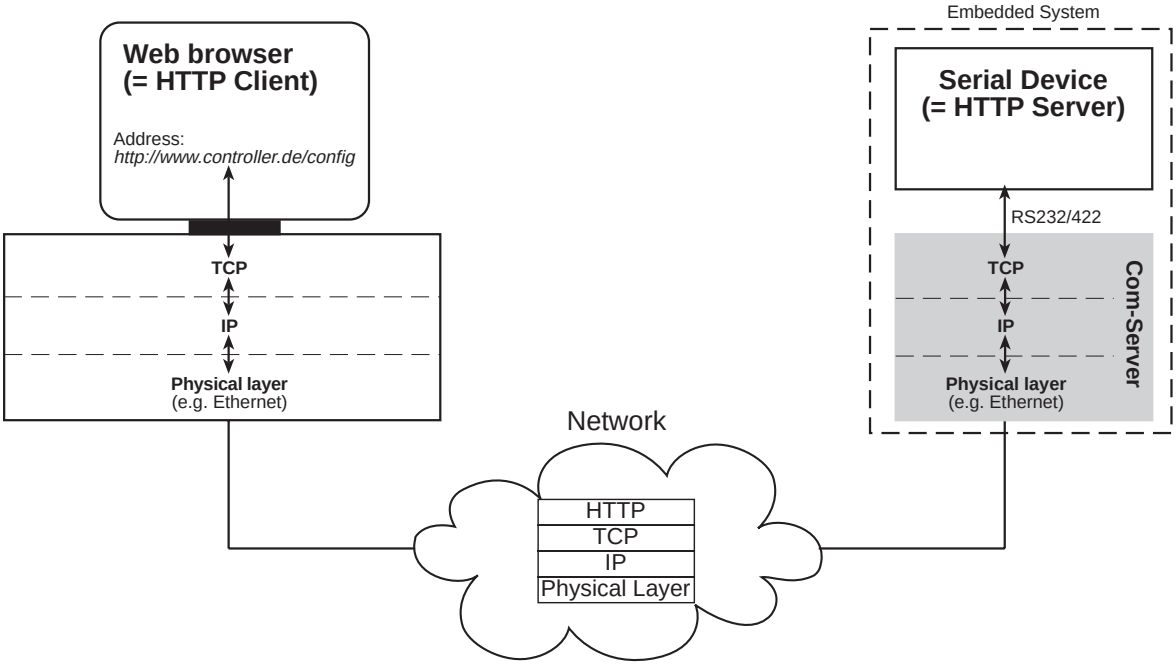
- The serial device is responsible for representing, storing and administering its data. All it needs to know is a few additional ASCII strings.
- The Com-Server takes over the complex data transport through the networks and makes sure the data arrive in the Web browser. On the data level it operates completely transparently, so that any kind of data traffic can be handled from a plain ASCII representation to a sophisticated presentation using Java code.

And all this using just a few simple ASCII strings!
Of course there are a few things you'll need to know: On the following pages we'll explain just enough about how HTTP protocol works, since it is the basis of our procedure:

Sequence of an HTTP connection

HTTP is an ASCII protocol which is built on a secure TCP connection and which is used for the overwhelming majority of Internet accesses. Port 80 is specified in the corresponding RFCs as the standard TCP port for the HTTP service; by means of an appropriate entery in the command line of the browser however (e.g. *http://www.device.de:8000*) other ports can be used as well. Communication takes place using the request/response principle: after a TCP/IP connection is established the Client (=browser) typically uses an HTTP request to get a particular file from the HTTP server (*GET [filename]*). If successful, the server sends the desired file back as reply and then closes the TCP connection. In order for the browser to represent the data received from the server on the screen, these data must be in HTML format.

The following illustration shows a serial terminal device connected to the network through a W&T Com-Server. To enable access using a standard browser, the firmware needs only to have added an interpreter for the very simply constructed GET command of the HTTP specification. Administration of the much more complex aspects of TCP and IP are handled completely by the Com-Server.



In response to the GET command the requested data are put in an HTML frame and sent back to the browser. Any additional HTTP options sent along with the request are ignored. This behavior corresponds to the original HTTP 0.9 Standard, out of which the expanded but downward compatible Versions 1.0 and 1.1 were subsequently derived. Those who are interested in going deeper into this subject are referred to RFC 1945 and 2068, in which the full scope of HTTP is specified.

The HTTP GET request

The request generated by the browser always consists of the so-called request line, which may be followed by additional option lines. The end of an HTTP header is indicated by a CRLF, which allows an HTTP server to check for a double CRLF as a protocol end condition.

The entry *http://www.device.de/documentname* in the command line of the browser creates the following ASCII string as an HTTP request on the serial side of the COM server:

```
GET<Space>/document_name<Space>HTTP [0.9|1.0|1.1]<CRLF>
[Option 1]<CRLF>
[Option 2]<CRLF>
...
<CRLF>
```

The HTTP GET reply

By processing the variable *document_name* the HTTP server can generate the desired reply.

The HTTP specification allows you to skip generating and sending back an HTTP header at this point and instead to immediately send the requested data. To enable representation in the browser, these data must be sent in HTML format. A simple reply might therefore be constructed as follows:

```
<html>
[documentname]
</html>
```

A particular feature of HTTP is the way the TCP connection is ended. The usual rule that a TCP client is responsible both for establishing and ending the connection is not applied here. Once the requested data have been sent the HTTP server must end the TCP connection. The Com-Server allows this either through use of a level change at one of its handshake inputs (CTS or DSR) or by means of a timeout.

Name or IP Adresse?

It is entirely up to you which way you want to contact the COM server. On the network protocol level it makes no difference whether an IP address or a hostname is given in the browser as a target system. Using an IP address does offer the advantage that the computer's TCP/IP stack can immediately establish the connection – whereas if a device name was specified, the associated IP address must be determined via a DNS system or a HOSTS file before the connection can be established. Maintenance of these databases is the responsibility of your network administrator; always contact him if you have questions.

- Example specifying an IP address: `http://172.20.20.1/testfile`
- Example specifying a name: `http://www.device.de/testfile`

Uploading information to the serial device

If the serial device has only limited configuration options or messages available, it is entirely sufficient to define a few keywords for the variable *document_name*, which cause certain actions or replies to follow when they are received. Of course you can also use the same simple mechanism to control devices with a wider range of configuration possibilities. The solution can be found in the HTML submit forms, which you will have often encountered in the Internet (for example our forms for ordering samples and requesting information). If the *Submit* or *Send* button on one of these forms is clicked, the values contained in the individual form fields are sent to the HTTP server. Analogous to the arguments for running a program, these are then sent in the command line of the HTTP header in ASCII format and separated from the document name by a question mark (0x26, dec 63). Multiple options are each separted by an ampersand („&“) (0x26, dec 38):

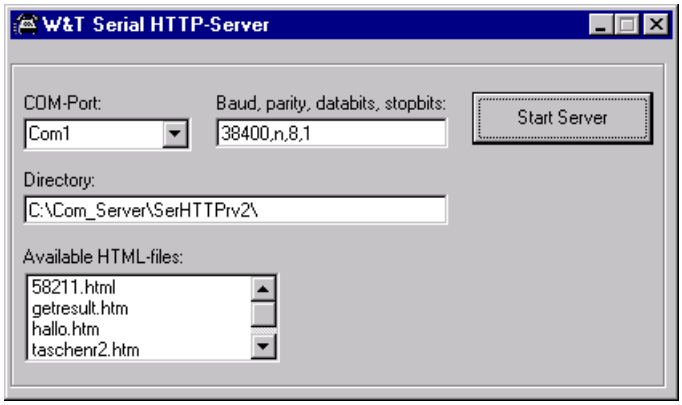
```
GET<space>/documentname?Option1=Value1&Option2=Value2&...
```

As the example shows, the effort required to implement the serial HTTP service even when using this user-friendly form technique is limited strictly to operations with ASCII strings. You can forget about cumbersome acknowledgement-based communication with its associated timeouts.

Visual Basic Demo „Serial HTTP Server”

The demo program written in VB 5.0 clarifies once again the division of labor described at the outset and in principle emulates a serial terminal device connected to the Com-Server.

In addition the demo explains the basic procedure for working with forms. The document *calculator.htm* contains a form for multiplying two numbers. Clicking the button *Get Result* generates a GET command for the file *getresult.htm*; the HTTP server receives the values to be multiplied as options. Once the result has been calculated and written to the *getresult.htm* file, the upload to the requesting browser follows.



Flow chart for the serial HTTP server