Documentation

HTTP_Protokoll_Vxxx.inc

This file implements all functions of the HTTP protocoll. If data was received from a client, it has to be interpreted. In case of the HTTP request, this file do fine work. Please look up detailled information about the HTTP protocoll itself before trying to understand the following functions. Here are listed the most important funtions of this file:

searchEmptyLine

Parameters:

s$:	Text to search in
check:	return value (TRUE/FALSE)

This method is searching for an empty line. An empty line in a Text is a double carriage return line feed, because every line ends with a carriage return line feed. The return value of this functions is TRUE if an empty line was found, else it is FALSE.

seperateHTTPRequest

Parameters:

request$:	HTTP-request
header$:	here is saved the header of the request / the first request
entity$:	here is saved the entity / the rest of the data
check:	see, if an error has occured

This function seperates the most important part for the moment from the rest of the HTTP request(s). They are always seperated by an empty line. Between 2 HTTP requests there is an empty line and also between the header and the entity. In a request, there rarely is an entity, eg. after sending a formular. In the string header$, the actual request is saved. This request has to be analyzed.

request_line

parameters:

- header$: head of HTTP request
- flag_tab$: a string of size 256(dummy)[because of stack size]
- result_check_keyword$: " (128 Byte) "
- check: result
  - CASE: GET
    - ->vars: URI$, HTTPVersion$, check = dGET
  - CASE: HEAD
    - ->vars: " check = dHEAD
  - CASE: POST
    - ->vars " check = dPOST
- URI$: here the adress (URI/URL) is saved
- HTTPVersion$: This is the HTTP-Version

Structure of the command line in a HTTP request:

<command(GET,HEAD,POST)><Space><URI><Space><HTTP-Version><CrLf>

The several options/parameters of the HTTP request are saved in detailled variables. The most important information are transmitted in the request line. The request line is the first line in the request. This line is also very important for the answer, because nearly all information about the answer are stored here. The first keyword concludes the kind of answer. In the URI

is saved the name of the requested document. This document will be load and transmit to the client. The HTTP version has to be checked, too.

getSendDocument

Parameters:
- s$: here the document will be saved (return value)
- uri$: URI (URL) of the document
- content_type$: the content type of the document is saved in this variable
- part: var is needed for documents, which are bigger then the buffer
- check = TRUE -> all done successfully
- check = FALSE -> File not found
- check = err_peek_flash -> error while reading flash memory(file found but not saved in string)

This function implements the the analysis of thr URI/URL and is searching for the right document in flash memory. The filenames are saved in a string in the same order as the files are saved in flash. For more information about this topic, please look at the structure of the files in the chapter about DATAFILES.
The URI has to be compared with the keyword in the string. The Index of the keyword in the string is the index of the Array with saved adresses in flash memory. The file from this adress is read out from memory and saved in RAM. If the RAM is too small, only a part is saved in RAM and a flag is set.

sendHTTP

Parameters:
- document$: the answer document to send in entity
- content_type$: what kind of document
- typ: what kind of request:
  1) dGET --> complete answer
  2) dHEAD --> only header will be send
- socket: socket number, to which socket must this answer be send
- check: was it successful?

This function also make use of global variables!!!!

This function implements the transmission of the answer to the HTTP request. This could be an error message or a valid answer. The structure always is the same. The variables will be analysed and the answer will be build.

There are some static elements like the HTTP version, which are saved in defines. They are always the same, but are probably important for the client, so they have to be specified.

All the saved information are insert to the structure of the protocoll, that the client is able to understand the data. Behind the information about the document, we will append an empty line and poss. the document itself.

Documentation

Datafiles Vxxx.inc:

This file manage the strucure of the saved files in flash memory. They have to be saved in a special way, that the processing of the answer is faster.

A String is needed with thename of the files saved in flash memory, also in the same order as in flash.The separator character between the names is a blank. If a path is needed,it's nearly possible. You only have to save this path within the name, eg.applPath/demoFile.xxx"

What to do, if you want to append a new file in flash memory:

1) copy file in directory of this programm
2) increase anzFiles
3) add a DATALABEL with a predicative name
4) append name of file to files$
 important:
    1) between every name of file has to be 1 blank (and only 1)
    2) at the end of the string has to be 2 blanks
5) if length of files$ is too short, increase fileStringLength
6) insert following lines before the line "ende::":
    1) [name of datalabel]::
    2) DATA FILE "[name of file]"
    3) filesAdr(x) = [name of datalabel]
    4) x = x + 1
    5) [empty line]

What to do, if you want to delete a file in flash memory:

1) decrease anzFiles
2) delete name of the file from files$
    important:
    1) between every name of file has to be 1 blank (and only 1)
    2) at the end of the string has to be 2 blanks
3) search for DATA FILE "[name of file]"
4) memorize the name of the datalabel above and delete this line
5) delete the 2 line under DATA FILE "[name of file]"
6) delete the line with DATA FILE "[name of file]"
7) delete the declaration of the memorized datalabel

The values of the Datalabels are saved in an Array, so that you can compute the adresses of files saved in the String.

Documentation

cgi_Vxxx.inc:

This file shows some examples of cgi-programms. First we have to manage the cgi-command. We have to know, which cgi programm we have to start. This is implemented in a switch case instruction. The name of the URI is compared with all cgi functions. So the gadget function is called. Here are some short description of the cgi relevant function in this file:

startCGI

Parameters:
- uri$: URI of HTTP request
- retval: succes of function (TRUE,FALSE)

This function analyzes the URI and branch to the requested function.

generateHTMLCode

This is one example of generating dynamic HTML sites. There is one 'standard' site, that we will change. In this example, there are some information, which depends from the settings of the user. If you want to change something in an webpage, you first have to search the right location in the source code, where you can insert or change the code. HTML sites always are source code, you can't compile them. So in this case, it is very easy.

search

Parameters:
- doc$: document to search in
- search$: text to find in doc$
- erg: return value, is position of the text or -1 if the text was not found

This functions search for string in another. The found position is saved in the return value erg. If there is no match, the return value is -1.

blinkLeds

This sub is one example of a cgi programm. The Client sends a HTTP request with the name of this cgi function to this server, and this programm is startet. In this case, the URI also consists of a parameter. This parameter is the parameter for this cgi function. This sub controls the LEDs on this board. So it is possible to control the LEDs outside the Tiger. Everybody with a browser and an active Connection to the Tiger is able to control all features on Board. So it's possible to control your Tiger from nearly everywhere in the world. You only need an Internetconnection.