AN4965
Application note

WebServer on SPWF04S module

Introduction

The HTTP protocol is an excellent solution for communication between humans and embedded devices because of the ubiquitous presence of browsers.

This SPWF04S HTTP server application note is for developers seeking to implement Webservices and Webpages with static or dynamic content in their embedded products.

It includes:

- an introduction to SPWF04S HTTP server, including server capabilities
- server side information
- built-in functionality information

Contents

List of tables

1 Features overview

The SPWF04S HTTP server features:

- HTTP 1.1 support
- HTTPS support (on top of TLS 1.2)
- HTTP 1.0 methods GET, POST (RFC 1945)
- HTTP 1.1 method OPTIONS (RFC 2616)
- Delivering static responses based on filesystem access
- Ability to deliver dynamic, application-generated responses based on server-side-includes and fully application-generated content

2 Introduction into SPWF04S HTTP server

This chapter describes some of the product packages and protocols supported by the SPWF04S HTTP Server. It includes a basic setup of protocols, which should be sufficient for an embedded device to provide Webservices capability via HTTP.

2.1 Using multiple connections

The HTTP server is able to manage up to four multiple HTTP connections in parallel.

2.2 Supported HTTP methods

The HTTP supports the following HTTP methods:

- GET: to request a resource from the HTTP server
- POST: to deliver data to the HTTP server
- OPTIONS: to request the HTTP header only from the HTTP server.

2.3 Generating HTTP responses

When delivering web pages over HTTP, there are two main ways to generate responses for a request:

- With a static resource (a file).
- With a program the HTTP server calls after receiving the request. On larger web servers, this is usually done by spawning an additional process that handles the request by executing a program written in a scripting language like PHP, Perl or ruby.

Both methods are supported by the SPWF04S HTTP server, but the generation of dynamic responses requires server side include (SSI) tags (see).

2.4 HTTP protocol 1.1

With HTTP 1.1, the SPWF04S HTTP server:

- Can parse out the content of the request when it is transferred in chunks (chunked encoding)
- Can send the content of the response in chunks (chunked encoding)
- Doesn't close the TCP connection immediately after a sent response, but after a timeout when no further request is received over the TCP connection

2.5 Get HTTP header content

The following HTTP header fields are evaluated internally:

- **Content-Length**: length of received HTTP content (to determine the end of the request for HTTP content).
- **Content-Type**: type of HTTP content (e.g., multipart/form-data or application/xwww-form-urlencoded).
- **If-Modified-Since**: whether the browser should only receive modified files, in order to determine whether to transfer the file or the status code 304 (Not Modified) in the response.
- **User-Agent**: name of user agent which sent the HTTP request; the name is saved and used internally.
- **Content-Transfer-Encoding**: encoding of received HTTP content to determine whether the content is encoded in chunks ("chunked encoding").

2.6 Receiving POST data

Data that are received with a POST request are normally delivered to the filesystem, with two exceptions:

- When the content type is "application/x-www-form-urlencoded", the content contains name/value pairs and every pair is notified.
- When the content type is "multipart/form-data", an uploaded file is embedded in one of the multiple parts and the content of the received file is notified.

See *Section 4: "Built-in functionalities"* for a detailed description of available built-in actions.

2.7 Selecting the homepage

If the user of a HTTP client (browser) only requests the host name (or IP address) of the device with the HTTP server without specifying a page name, the HTTP server returns a default page, usually index.html.

For example, if the user requests http://device77, the server returns http://device77/index.html.

2.8 Favicon-support

Most browsers request a file named "favicon.ico" which represents a Favicon, a small logo displayed left in the address row of the browser.

If such a file is found in the local file system, the content of the file is returned in the HTTP response, with Content-Type "image/x-icon".

Normally, if a requested file is not found, the SPWF04S HTTP server returns a page with status code 404 (File Not Found) unless it is a Favicon file, in which case an empty file is returned with status code 200 (OK) and Content-Type "image/x-icon".

2.9 Using MIME types

For every HTTP response, the type of the returned file has to be declared so the browser knows how to handle/display the file.

These are declared as Multipurpose Internet Mail Extensions (MIME) types: a set of strings, primarily declared for email exchange, in the Content-Type HTTP header field.

Since the content of the file to be returned in the HTTP response is read via filesystem, the associated MIME type is also returned from filesystem (as index) when opening the file.

The following MIME types are supported by the SPWF04S HTTP server:

- application/font-woff
  - application/json
  - application/vnd.ms-fontobject
  - application/x-java-applet
  - application/x-javascript
  - application/x-raw-stuff
  - application/x-shockwave-flash
  - application/x-www-form-urlencoded
- audio/midi
  - audio/mpeg
  - audio/mpeg3
  - audio/wav
  - audio/x-ms-wma

- image/bmp
  - image/gif
  - image/jpg
  - image/png
  - image/tiff
  - image/x-icon
  - image/x-pcx
- multipart/form-data
  - multipart/x-mixed-replace
- text/css"video/mpeg
  - text/html
  - text/plain
  - text/xsl
  - text/xml
  - video/x-ms-asf
  - video/x-ms-wmv
  - video/x-msvideo

2.10 Compressing resources (GZIP)

If a requested file from a HTTP client cannot be found via filesystem, the server also checks for a corresponding gzip compressed file by inserting ".gz" before the last dot in the name of the requested file.

Resources containing SSI (refer to *Section 4: "Built-in functionalities"*) cannot be compressed.

Example:

When a user requests "http://device77/large.txt" with a browser and the SPWF04S HTTP server cannot large.txt via filesystem, it checks for large.gz.txt and, if found, transfers it in its response. In this case, the Content-Transfer-Encoding header field is also delivered with the value "gzip" to alert the browser that it is receiving a gzip compressed file.

2.11 Controlling the browser cache (date/time)

The SPWF04S HTTP server can control the browser cache by sending the Expires field with an appropriate date/time stamp:

- in the future (year 2100) if a file must be cached
- in the past (year 2000) if a file must not be cached

The caching property is defined per MIME type. The default values are chosen so that files of all types can be cached except for:

- text/html (when the content is dynamic HTML)
- text/plain
- text/xml
- application/json
- application/x-raw-stuff

3 Server-side-includes (SSI / Dyn-HTML)

Server-side-Includes files are dynamically generated from a template file.

A template file is a file that is post-processed by the SPWF04S HTTP server. To indicate that a HTML source file contains dynamic HTML data, the HTML file must have a .FHTML extension instead of HTML.

It contains special markers that are replaced dynamically with data coming from the application.

These markers are HTML comments with the following format:

<!--|code|parameter1|parameter2|parameter3|parameter4|-->

The code determines the resulting string type and parameters 1 to 4 may be used to define certain substrings of that tag; mostly attributes. Not all possible types use all the parameters; only the necessary parameters need to be provided. See *Table 2: "List of the possible dynamic marker types and their parameters in templates"* for a complete code list and corresponding parameters.

Using the following tokens buffer, filled by AT+S.INPUTSSI command (Ref. to UM2114):

|aName|y| |aValue|someText|y|aName|aValue|someText| |

some short examples of marker replacement are following:

Table 1: Example of marker replacement

Template text	Generated HTML code
<!--|00|SelStart|0|-->	<select name=”aName” MULTIPLE>
<!--|01|Select|3|-->	<option SELECTED value=”aValue”>someText
<!--|02|SelEnd|-->	</select>
<!--|03|CheckBox|2|-->	<input type=”checkbox” name=”” value=”aValue” CHECKED>someText
<!--|03|CheckBox|6|-->	<input type=”checkbox” name=”aName” value=”aValue”>someText
<!--|06|Input|4|-->	SomeText
<!--|06|Peers|rx_rssi|0|-->	-57

No close tag </option> is generated for <option> filed, as these are not mandatory for HTML syntax.

The other marker types follow along these lines.

Table 2: List of the possible dynamic marker types and their parameters in templates

Code	Description	Parameters
00	Start tag of a select box	Parameter 1: Must be SelStart Parameter 2: Starting token index inside buffer filled from application via AT+S.INPUTSSI command. Ref. to UM2114. Tokens must be set in the following order: Name of the select box If given (any value), multiple selections are allowed.
01	Option tag	Parameter 1: Must be Select Parameter 2: Starting token index inside buffer filled from application via AT+S.INPUTSSI command. Ref. to UM2114. Tokens must be set in the following order: Value of the option Label of the option If given (any value), this options is marked as Selected.
02	End tag of a select box	Parameter 1: Must be SelEnd
03	A check box	Parameter 1: Must be CheckBox Parameter 2: Starting token index inside buffer filled from application via AT+S.INPUTSSI command. Ref. to UM2114. Tokens must be set in the following order: Name of the check box Value of the check box Label of the check box If given (any value), the check box is initially checked.
04	A radio button	Parameter 1: Must be RadioButton Parameter 2: Starting token index inside buffer filled from application via AT+S.INPUTSSI command. Ref. to UM2114. Tokens must be set in the following order: Name of the radio button (radio buttons with the same name cannot be checked at the same time). Value of the radio button Label of the radio button If given (any value), the radio button is initially checked.
05	A text area start tag	Parameter 1: Must be TextArea Parameter 2: Starting token index inside buffer filled from application via AT+S.INPUTSSI command. Ref. to UM2114. Tokens must be set in the following order: Name of the text area Number of columns of the text area Number of rows of the text area

Code	Description	Parameters
06	Raw text	Parameter 1: Must be one of the following, to recall proper SPWF04S Web Server actions: Input – replace with raw text (use Parameter 2) DevConf – replace with configuration variable (use Parameter 2) DevSts – replace with status variable (use Parameter 2) Peers – replace with peer status variable (use Param. 2 and 3) ADC – replace with ADC value (use Parameter 2) GpioR – replace with GPIO status (use Parameter 2) Parameter 2: Token index inside buffer filled from application via AT+S.INPUTSSI command. Ref. to UM2114. Configuration variable Status variable Peer status variable ADC number GPIO number Parameter 3: Peer number
07	A text area end tag	Parameter 1: Must be TextEnd
08	A text field	Parameter 1: Must be TextField Parameter 2: Starting token index inside buffer filled from application via AT+S.INPUTSSI command. Ref. to UM2114. Tokens must be set in the following order: Name of the text field The initial value of text field Size of text field
09	A submit button	Parameter 1: Must be Button Parameter 2: Starting token index inside buffer filled from application via AT+S.INPUTSSI command. Ref. to UM2114. Tokens must be set in the following order: Name of the submit button Value (displayed text) of the submit button
10	A reset button	Parameter 1: Must be Reset Parameter 2: Starting token index inside buffer filled from application via AT+S.INPUTSSI command. Ref. to UM2114. Tokens must be set in the following order: Name of the reset button Value (displayed text) of the reset button
11	A password field	Parameter 1: Must be Password Parameter 2: Starting token index inside buffer filled from application via AT+S.INPUTSSI command. Ref. to UM2114. Tokens must be set in the following order: Name of the password field The initial value of password field Size of password field

For the <select> element, currently the argument "size = x" is not supported. So actual browsers will not display the option elements in a multiline rectangle, but in a dropdown-box only.

4 Built-in functionalities

The single or multipart form data in the HTTP POST stream trigger specific actions when they are read, as described in the following table.

Table 3: List of built-in functions

POST information name	POST information value	HTTP server action(1)
Output	SomeText	+WIND:57: Output from remote:%u:%s
Key	Must be user_desc configuration variable to allow remote configuration	+WIND:64: Remote configuration:%s:%s (only shown if it is wrong)
Ssid	SSID	+WIND:64: Remote configuration:%s:%s (only allowed if Key was previously accepted)
Pwd	WPA PSK	
WpaeId	WPA-E identity	
WpaeAnonId	WPA-E anonymous identity	
WpaeType	WPA-E EAP type	
Ip	IPv4 static address	
Netmask	IPv4 static netmask	
Gateway	IPv4 static gateway	
Dns1	IPv4 static primary DNS	
Dns2	IPv4 static secondary DNS	
Dhcp	DHCP mode (0: static, 1: dynamic, 2: Auto IP)	
IbssAuth	Authentication type (0: shared, 1: open)	
Auth	Authentication mode (0: open, 1: WEP, 2: WPA, 3: WPA-E)	
Mode	SPWF04S mode (0: idle, 1: STA, 2: IBSS, 3: miniAP)	
Scfg_*	Value of the variable to be set	
GpioC_*	Value of the GPIO to configure (out, in, in_r, in_f, in_b)	
GpioW_*	Value of the GPIO to set (0: low, 1: high)	
DAC	Value of the DAC to set on GPIO15 (0 to 3300mV)	
PWM_*	Value of the PWM and duty cycle to set (KHz_dc) on GPIO2 or GPIO4	
Sleep	Enter or Exit (1 or 0) sleep power mode	

POST information name	POST information value	HTTP server action(1)
Reboot	-	Save the configuration set to the Flash, and reboot the SPWF04S module (only allowed if key was previously accepted)

Notes:

(1)(ref. to UM2114 on www.st.com)

5 Revision history

Table 4: Document revision history

Date	Revision	Changes
17-Jan-2017	1	Initial release.
17-Nov-2017	2	Updated *Section 2.9: "Using MIME types"*. Updated *Section 4: "Built-in functionalities"*. Minor text changes.

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.