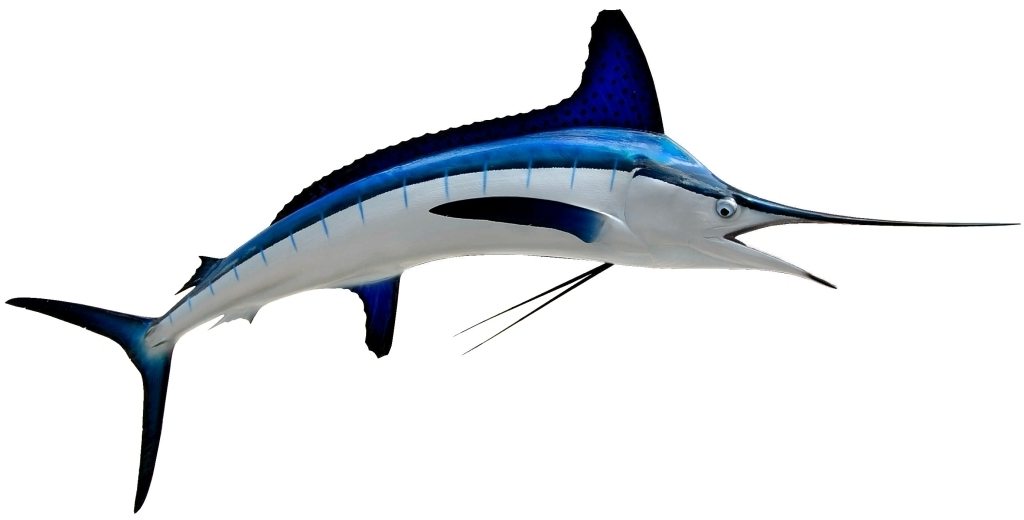
**MARLIN**

Webserver and webclient

Upgrade 3.7



Copyright © 2012 - 2016 ir. W. E. Huisman MSC

This library is released under the MIT license. You are free to use it in any way you like, provided that you leave a reference to the original software. See the file license.txt for further details.

**UPGRADE TO VERSION 3.5**

**Introduction and purpose**

This document is about the Marlin webserver library. Previous documentation and clarification of the programming model and the reasons behind the Marlin webserver can be found on the codeproject article: <http://www.codeproject.com/Articles/1069106/Marlin-Escaping-from-IIS>

This document is about the further developments of Marlin between versions 3.0 and 3.6. It describes the general improvements in terms of function. For a full report of all changes and bug fixes (sure, it’s software – so there’s bound to be little bugs), see the releasnotes of the last version in the source download.

So these are the highlights!

**Version 3.1 WebDAV handling**

Handling of WebDAV HTTP calls has been added. WebDAV is used for document management systems, source versioning systems and OData (Open-data). WebDAV calls include the calls for “MOVE”, “COPY” and “SEARCH” on the general resources (files). But they also include the calls for the properties, like:

* PROPFIND: Finding a resource by finding a property
* PROPPATCH: Patching the property of a resource
* MKCOL: Making a collection of resources (i.e. a directory)
* LOCK: Locking a resource for other users
* UNLOCK: Unlocking a resource after use

Having all those extra HTTP calls can be very fine, but some webservers or proxies might not recognize these calls and even block them. To come by this problem, verb-tunneling has been invented. This involves the translation of these calls to regular HTTP ‘POST’s and then having the ‘real’ call in a http request header (“verb-tunnel:”). Marlin now supports the verb tunneling by default.

The logging subsystem has seriously improved. A lot of bugs where removed in logging strings with ‘%’ signs in them. But the most important improvement was the asynchronous writing of the logging buffer to the filesystem. This removed a serious performance hit in the server. These improvements have spread over several versions after this one, but I mention it here once.

Sleeping and reawakening a thread in the threadpool is now possible. This means that you can grab an extra thread from the threadpool and use it for an extra purpose, other than the webserver. The number of maximum threads get extended by 1 if you do so, and decreased after you stop the thread. You can use this to your advantage to build processing queue’s heartbeats or whatever longer processing that you want.

**Version 3.2 WSDL’s**

Prior to this release, you had to add a lot of SOAP messages, and then let the WSDL cache go on its way and generate a WSDL file from it. This ensures of course that the WSDL is complete and correct. That’s of course very fine for a server, but what about the client?

From this version on you can also read an existing WSDL from a file and use that one for testing your calls. This makes it possible to generate a WSDL with external tools, up front and engineer an interface API solution, working your way toward the agreed interface.

Apart from the ‘loose’ WSDL files, more data testing has been added. Apart from integer sizes, the support of string enumerations is the big change here. You can define which enumerations a SOAP XML field can have. You will see the enumerations on the generated interface HTML pages in the SOAP examples, and you can explicitly test on them.

Here is an example from the test server in the project:

|  |
| --- |
| POST /MarlinTest/TestInterface//MarlinWeb.acx HTTP/1.1  Host: CERT5  Content-Type: application/soap+xml; charset=utf-8  Content-Length: **length**  <?xml version="1.0" encoding="utf-8"?>  <s:Envelope xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:s="http://www.w3.org/2003/05/soap-envelope" xmlns:a="http://www.w3.org/2005/08/addressing">  <s:Header>  <a:Action s:mustUnderstand="1">http://interface.marlin.org/testing/MarlinSecond</a:Action>  </s:Header>  <s:Body>  <MarlinSecond xmlns="http://interface.marlin.org/testing/">  <!--Translation has values: Deutsch (='Die Deutsche sprache') / English (='The English language') / Espagnol (='El lingua franca') / Français (='La plus belle langue')-->  <Translation>**string**</Translation>  </MarlinSecond>  </s:Body>  </s:Envelope> |

Before the node that carries the enumeration (<Translation>) you’ll see a comment node with an explanation as to which enumerations you can use on the following node.

**Version 3.3 JSON roundtrip**

Suppose you have a server with a SOAP interface. But there comes a new paradigm on the browser: javascript. And not just one script, but a barrage of javascript interfaces. Sometimes is seems whole the world is jumping into that. AngularJS to mention as one of the new hypes in these interfaces. But wait: suddenly SOAP is not cool anymore. Everyone wants JSON interfaces. Now how to cope? Writing your server over again. No, roundtrip JSON / SOAP / JSON translation is one of the quicker solutions.

Suppose a web javascript interface want’s to call my SOAP service, it can do so by calling the service with all the parameters as HTTP parameters. So lets suppose I have a SOAP service with a “SiteHandlerJson2Soap” in place, then the following call:

|  |
| --- |
| http://server:7500/Interface/MyCall?one=1&two=2&three=3 |

Will be automatically translated into the following SOAP call.

|  |
| --- |
| <xml>  <Envelope>  <header/>  <Body>  <MyCall>  <one>1</one>  <two>2</two>  <three>3</three>  </MyCall>  </Body>  </Envelope> |

Note here the placement of the name of the call (‘MyCall’) as last part of the resource of the URI, and as the main node of the message. All URL parameters now translate to ‘ordinary’ SOAP parameters in the soap call. This message then can be handled in the ‘old fashioned SOAP way’, as the server used to, and the server will generate a response message in SOAP.

|  |
| --- |
| <xml>  <Envelope>  <header/>  <Body>  <MyCallResponse>  <Counting>true</Counting>  <Sum>6</Sum>  </MyCallResponse>  </Body>  </Envelope> |

The response message then gets translated back to JSON by the SiteHandler ‘Json2Soap’ to a JSON answer for the web interface.

|  |
| --- |
| {  “MyCallResponse” : {  “Counting”: true  ,“Sum”: 6  }  } |

And this is the JSON message that a web interface can easily digest.

Please note that the conversion from JSON into SOAP and SOAP into JSON is done in the server just by constructing a SOAP message from a JSON message and vice versa. So that you can do the same thing also in your code.

Also be warned that not all JSON messages translate into SOAP messages, as the JSON notion of arrays does not exist within the SOAP realm!

Version 3.3.1 Chaining of SiteHandlers

It’s quite possible that you have different ways of handeling a HTTP request for a site. For instance a handler for authorized files and directories, and one without authorization in special cases.

To easily program this, you can now set multiple site handlers on one and the same site, just by calling ‘AddHandler’ more than once for the same HTTPCommand (‘get’, ‘put’ and so on).

The handlers get chained in order of appearance, and if a handler cannot handle the HTTP request by returning ‘false’, the next handler in the chain gets called automatically, until you run out of handlers.

Version 3.3.2 Mandatory namespaces in SOAP

In order for .NET 4.0.0 and higher to consume our WSDL’s it proved to be necessary to add mandatory namespaces to all SOAP messages and in the WSDL. After this action had been taken Visual Studio could create a .NET SOAP client interface directly from our WSDL’s.

**Version 3.4 (More) ODATA**

Multipart buffer posts as the HTML <form> element posts multiple sections of field data and files in one (1) HTTP post to the server. These forms of posts are also in widespread use in the handling of OData (open-data) requests. In order for Marlin to handle OData, it was necessary to fully support serverside and clientside multipart buffer requests.

The requests are gathered in an object called “MultiPartBuffer”. You can add parts to it with a call to “AddPart” and add messages or files to the multipart buffer. After that you can send them through the HTTPClient and receive them in a site handler called ‘SiteHandler­FormData”. This site handler has a different set of extra handlers. These handlers are:

* PreHandleBuffer: Gets called once for each HTTP call.
* HandleData: Gets called for each data part in the call (field etc)
* HandleFile: Gets called for each file part in the call
* PostHandleBuffer: Gets called after all data and file parts have been handled

Client certificates

In order to build secure websites that can only be approached by known clients, a general provision called ‘client certificates’ has been invented. You can specify in the HTTPClient which client certificate from your certificate store to use in a call to a certain website. The Marlin server can request for this client certificate and find it in the local certificate store on the server machine and ascertain itself that it’s the correct client certificate.

In order to build certificates for the development stage, a new directory “Certificate” has been created in this project. Here you can make root CA certificates, SSL/TLS certificates and client certificates for yourself and for the development fazes of your software. Beware. Here there be dragons! Read the extra documentation on how to do this exactly.

Windows 10 hangs

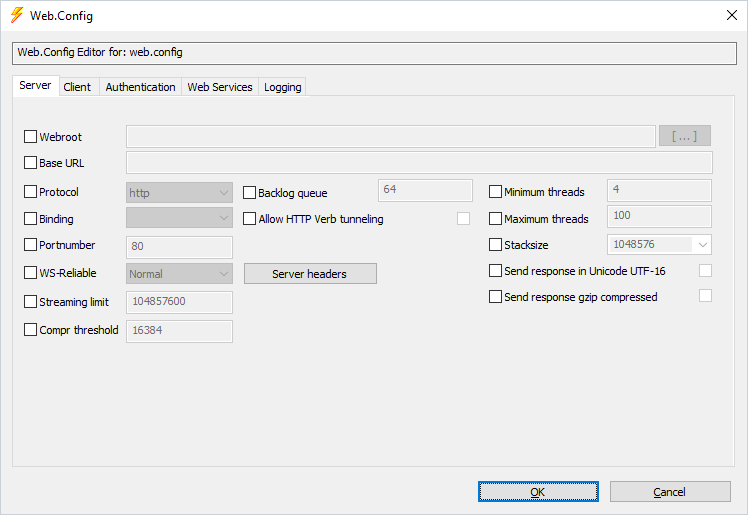
“O everything is better in the next version of MS-Windows”. We keep hearing that from the commercial department of Microsoft. But not in the HTTP Server API 2.0 ! For the first time the contents of the incoming request in “HttpReceiveHttpRequest” has been changed since MS-Windows Vista. And undocumented as well. Yes: your heard me: no extra documentation on MSDN.

There where now undocumented extra requestInfo records in the request. It was only after installing the Windows 10 SDK, and after reading through all the headers from that SDK that the veil was lifted. The extra requestInfo records are carrying the SSL specifics of an incoming “https” call. And as Marlin was not prepared for those, it did hang in reading the request info records. Auch.

After the SDK entries were found, Marlin now logs the extra info in the logging file (if turned on of course). So there is an extra LogSSLInformation method in the server that logs the current SSL/TLS protocol and the cypher strengths of the current connection. But only if run on MS-Windows 10 and Server 2016. Mind you!

Rewrite of the web.config editor.

The web.config editor in the HTTPManager program had become quite bloted, long and less clear as it once was. So I decided to give it a rewrite and create tabs per function. So now you have tabs for the server configuration, the client configuration, authentication, web services and logging. This is what it looks like:



Version 3.4.2 Full UTF-8 / UTF-16 support

Marlin already had UTF-8 and UTF-16 support. So what do you mean by ‘full’? Well, the support of UTF-8 was only for the first 3 code pages and the West-European diacritics. Letters such as à, á, ï and so on. In essence the standard vowels with all the accents on them. But as time progressed, more became necessary. A € euro sign for instance is not on the first three pages of UTF-8, and the UTF-8 was not supported in the URL parameters as well. So I fixed all that. It was ugly, horribly and also behaves different in debug and in release mode.

UTF-8 and UTF-16 are now fully supported on the full URL, the URL parameters, and the URL anchor as well. And also in the SOAPMessage and the JSONMessage and all its parts.

So if I messed up somewhere, let me know and I’ll dive right back in. (and take a deep sigh!)

Version 3.4.3 Extra Tracing in the HTTPClient

Tracing of the traffic in the logfile of the HTTPClient has been rather basic. So now it is possible to “SetTraceData” and “SetTraceRequest”. The first call traces incoming answers and the second traces the state of the HTTP request prior and after the ‘real’ request and response calls. So that we can see what state the WinHTTP engine of Microsoft is in.

Please be prepared for rather big logfiles!

Version 3.4.4 GZIP compression / decompression

Compression and decompression of content was a missing feature in the Marlin server. I added it in this version. Marlin is now using the original ZLIB of Jean-loup Gailly and Mark Adler for RFC 1950/1951/1952 ‘gzip’ compression and decompression. Yes I known: there are a lot more compression/decompression methods to be had in HTTP traffic, but ‘gzip’ is just the basic one that’s most widespread and since 2011 that everybody is using. It’s been tested with all major browsers and web stacks, including Google Chrome, Firefox, the IE’s and Edge.

**Version 3.5 Performance release**

This is a serious performance release. Firstly: the incoming HTTP request is now only recorded in the central server and forwarded through the HTTPMessage and the threadpool to a different thread that's handling the reading of the HTTP body of the message within the HTTPSite. This clears out the responsibility of reading large file bodies from the central server to a threadpool thread and a site. Thus clearing the central processing thread of the server to be on the business again of receiving incoming calls.

One major parameter of course is the number of available threads, and the length of the input waiting queue of HTTP calls to be processed. So keep in mind that after this version you just might want to fiddle with the following web.config parameters:

- <Server><QueueLength> : Multiple of 64

- <Server><Stacksize> : Stack space of the threads

- <Server><MaxThreads> : Should be less than <QueueLength>!

Secondly: I have rewritten the memory model of the complete server. All malloc / calloc and free() calls are now gone from the code. Everything has been allocated with ‘new’ or with ‘new []’ methods and will be freed from the heap with 'delete' or 'delete []'. If you are using FileBuffers straight from the Marlin library, or using the WideString\* conversion calls, retrace your steps in the code and refactor your memory modelling likewise!

Thirdly, and as a consequence of the rewriting of the memory model, I fixed a few serious stack bugs in the sizing of the XML/SOAP messages and the reading of the HTTP buffers, while rewriting the memory model of the server. The server is now less prone to buffer overruns and memory-overflows. All thanks to the sonarsource company in Switzerland and their SonarQube analysis (See: [www.sonarsource.com](http://www.sonarsource.com))

**Version 3.6 Rewriting the get handler**

During the last versions the get handlers has gotten several re-writes. (But we only mention it here) The reason behind these rewrites lies in the way the popular AngularJS framework handles missing files and other internal errors. These are the main rules:

* All missing full resource names ending in ‘/’ fall back to ‘index.html’ in that directory;
* All other missing resources do fall back to the ‘index.html’ on the main page (!!)
* If the site is an application the fallback should do a fallback to the login page.

Another JSON feature is the handling of arrays. The roundtrip conversion from SOAP XML to JSON and back to JSON, did not handle these. Especially if the handler returns a SOAP message where multiple nodes with the same name are returned, it’s handy to have these converted to a JSON array, lest the nodes will overwrite themselves when importing the message into a javascript object. See this example from the test client below:

|  |
| --- |
| <xml>  <Envelope>  <header/>  <Body>  <Applications>  <Datasources>  <Datasource>develop</Datasource>  <Datasource>test</Datasource>  <Datasource>accept</Datasource>  <Datasource>production</Datasource>  <Datasources>  <Applications>  <Commercial>  <Application>Financials</Application>  <Application>Maintenance</Application>  <Application>Rentals</Application>  </Commercial>  <Default>Rentals</Default>  </Applications>  </Body>  </Envelope> |

Two of the nodes here (Datasources & Commercial) contain subnodes of the same name. So they get converted to a JSON array with only the contents of those nodes.

|  |
| --- |
| {  "Applications":{  "Datasources":{  "Datasource":  [ "develop"  ,"test"  ,"accept"  ,"production"  ]  }  ,"Commercial":{  "Application":  [ "Financials"  ,"Maintainance"  ,"Rentals"  ]  }  ,"Default":"Rentals"  }  } |

One of the main reasons the get handlers got re-written so often, was the need to have a websites content somewhere on disk outside of the server’s root directory. To accommodate this better, the virtual directory capability was added. This works by prepending the root directory of the site with the “virtual://” protocal. For instance:

- The server root directory is: "C:\WWW\Server\"

- The site root directory was: "C:\WWW\Server\MySite\"

- The site root directory now is: "C:\Appication\Version12\MySite\"

You can specify this with: site->SetWebroot("virtual://C:\Application\Version12\MySite");

Rewriting of a site handler was also triggered by the need to initialize things and set things up for a handler. That’s why a ‘OnStartSite()’ method was added to the SiteHandlers and the SiteFilters. The starting of the site in HTTPSite::StartSite() calls all the ‘OnStartSite’ methods of all registered filters and handlers. And because the ‘OnStartSite’ method is a virtual method, you can have your own localized initialization for your handler.

A last feature in this release are the handling of media types for the get handler.

To support both Microsoft Office and OpenOffice (or LibreOffice), a great number of content types have been added to the marlin server. These media types are quite hard to come by. In fact you have to gather them from all over the internet. This version adds an extensive list of media types to the default ‘get handler’.

**Version 3.7 Rewriting the testset**

During the development the testset was re-written a lot to accommodate for all the new features. But also the first head-up check of the testset was altered. In the new start screen you can see in 'one go’ that the startup of the server went well. That is, if you look at the column of ‘OK’ statuses after all the tests. And you can pick the interface pages from the start screen and paste them in your favorite browser.

