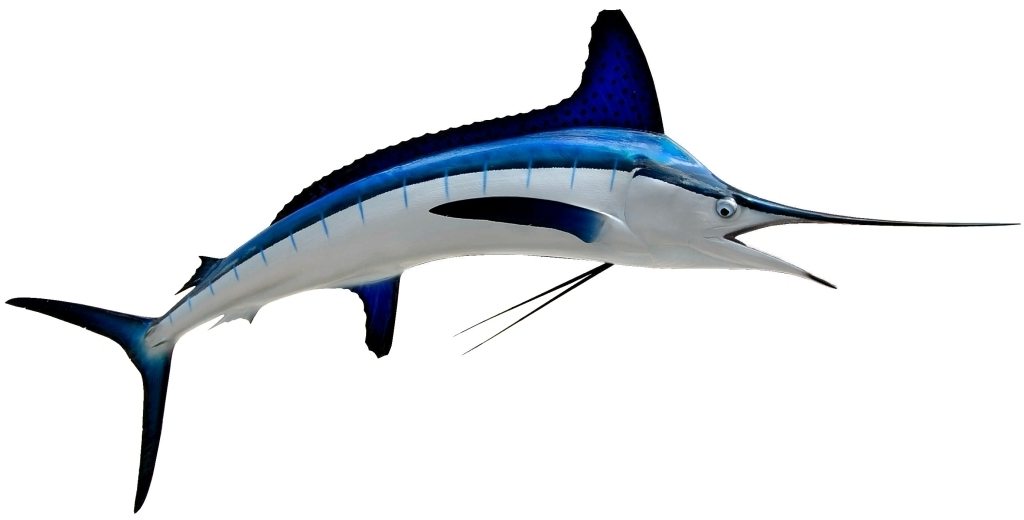
**MARLIN**

Webserver and web client

Version 4.0



Copyright © 2012 - 2017 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.

**VERSION 4.0**

**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> and on the base website of Marlin on <https://github.com/edwig/Marlin>

This version was created with one purpose in mind: the integration of the Marlin framework on the standard IIS infrastructure of MS-Windows. Though the original server was completely standalone and was – and still is – much faster when not running inside IIS, there was a large demand to make the move to IIS. Website administrators and technical personnel are much more comfortable in their ‘known surroundings’.

The good news: in version 4 you have a choice between three models:

1. The standalone webserver model, as it was in the previous versions
2. Running inside IIS on the IIS infrastructure
3. Running as a ‘Hosted Web Core’ for developers

As a result of this operation, the usage of one “web.config” file for the whole marlin server has changed, as the core server must read and use the configuration files of the IIS infra-structure. The ‘applicationHost.config’ file of IIS on the Windows system directory will be read and also the ‘web.config’ files on the ‘inetpub’ directory are used.

**Bootstrapping the webserver**

Marlin 4 is implemented as a ‘Native module’ in IIS. As a result of this choice, the bootstrap process of IIS is used to startup the Marlin 4 server module (“MarlinModule.cpp”). This module is not a class, but a file with a number of small classes and a loose service routine (“RegisterModule”). IIS only requests that you export this service routine through the DLL export mechanism. The service routine starts in its turn two factories (The ‘GlobalFactory’ and the ‘ModuleFactory’). These factories start the classes that handle the calls. In fact there are two factories:

* MarlinGlobalFacotry (derived from ‘GlobalFactory’)
* MarlinModuleFactory (derived from ‘ModuleFactory’)

These factory classes handle all the global and HTTP events. It’s the ‘OnBeginRequest’ of the MarlinModuleFactory where all the web calls come in.

The strategic choice of using the OnBeginRequest event only was made because it is after all the authentication and security (SSL/TLS) handling have been done, and just before the choice will be made which handler to call.

The original decision to go for a Microsoft webserver and not extending a BSD like approach was that we did not want to handle the authentication and security encryption issues, but that we sure wanted to handle all the webmethods.

Figure 1 on page 3 shows the order of initialization between these modules and how we get to the ‘OnBeginRequest’ event. Events 1 to 6 show the process up to the first HTTP call. Events 7 & 8 show how the server is stopped.

Please be advised that version 4 also has a separate document “Setup for IIS.docx”. This document has a ‘walk-through’ for the standard testset of the Marlin server for the IIS configurations. As the testset includes every feature like SSL/TLS and certificates, client certificates, authentication and the like, you will touch every aspect of those features in the IIS-Admin application.



**General changes in the main classes**

To be able to make a choice and not lose the original functionality of the system, it has been made possible to run in one of two modes (with or without IIS). As a result a few classes have now two derived implementations (IIS and Marlin). These classes are:

* The HTTPServer. Instantiating and calling “HTTPServerMarlin” will get you the standalone implementation. And instantiating and calling “HTTServerIIS” will be needed for IIS;
* The HTTPSite. In the IIS situation, calling “CreateSite” of the server will return you a ‘HTTPSiteIIS’. A ‘HTTPSiteMarlin’ is what you get in the standalone version;
* The WebConfig class also has a WebConfigIIS override

Normally when you are building a webserver with the standalone version, you would build an \*.exe application and somewhere call the “Run” method of the webserver. In IIS there is no executable. The module must be a DLL. So the starting is done by the “OnGlobalApplicationStart”. The IIS server has a “ServerApp” class on board from which you derive your application. To do this you \*\*MUST\*\* write a ‘InitInstance’ and ‘ExitInstance’ method for this class that will be called on starting and stopping the server.

This is where we register our sites and handlers and where we do the stopping logic (if needed).

**Usage of the IIS logfiles**

Due to the different workings of IIS, if you want to run in that mode, the web.config of Marlin will not be read. Instead the "ApplicationHost.Config" (on C:\Windows\System32\inetsrv\config) and the web.config (in C:\inetpub\wwwroot\YourSite\) will be read.

These are the main variables that are read by the WebConfigIIS class and the HTTPSiteIIS.

Table 1: Used config file nodes

|  |  |
| --- | --- |
| **Functionality** | **Where to find** |
| **Location of the log file** | <log><centralW3CLogFile><directory> |
| Logging is on or off | <log><centralW3CLogFile><enabled> |
| Site's root directory | <site><VirtualDirectory><physicalPath> |
| Protocol binding | <site><binding><bindingInformation> |
| **Protocol** | <site><binding><protocol> |
| Secure site (HTTPS) | <site><binding><sslFlags> |
| Streaming limit (in bytes) | <system.WebServer><requestLimits><maxAllowedContentLength> |
| Basic authentication | <location><authentication><basicAuthentication><enabled> |
| Basic realm | <location><authentication><basicAuthentication><realm> |
| Basic domain | <location><authentication><basicAuthentication><defaultLogonDomain> |
| Digest authentication | <location><authentication><digestAuthentication><enabled> |
| Digest realm | <location><authentication><digestAuthentication><realm> |
| Windows authentication | <location><authentication><windowsAuthentication><enabled> |
| NLTM caching | <location><authentication><windowsAuthentication>  <authPersistSingleRequest> |
| NTLM authentication | <location><authentication><windowsAuthentication><providers><NTLM> |
| Negotiate authentication | <location><authentication><windowsAuthentication><providers>  <Negotiate> |
| Kerberos authentication | <location><authentication><windowsAuthentication><providers>  <Negotiate:Kerberos> |

**CAVEAT:** Many options of the Marlin web.config are \*\*NOT\*\* supported in IIS mode. To use these options, incorporate them in the config file of your product!! And let the product set the options on the site by use of "Set...." methods.

**WSDL Checks and datatypes**

The WSDL (Web Service Definition Language) checking of incoming SOAP services are now extended with all XSD datatypes as defined by the W3C standard for XML schema's <http://www.w3.org/TR/xmlschema11-2/> Version 3 had only 6 basic datatypes. In this version there are a lot of newly understood datatypes. A complete list of all datatypes now understood is presented in table 2.

Table 2: All WSDL datatypes

|  |  |
| --- | --- |
| **XML Schema data type** | **Implementation** |
| **anyURI** | Any valid 'crackable' URL |
| date | Gregorian dates between 1-1-1 and 31-12-9999 |
| dateTimeStamp | Gregorian date + time + optional timezone |
| decimal | Any decimal number (without exponent) |
| long | 32 bits signed integer |
| int | 32 bits signed integer |
| short | 16 bits signed integer |
| byte | 8 bits signed integer |
| nonNegativeNumber | 32 bits positive integer or 0 (zero) |
| positiveNumber | 32 bits positive integer (> 0) |
| unsignedLong | 32 bits unsigned integer |
| unsignedInt | 32 bits unsigned integer |
| unsignedShort | 16 bits unsigned integer |
| unsignedByte | 8 bits unsigned integer |
| nonPositiveInteger | 32 bits negative integer or 0 (zero) |
| negativeInteger | 32 bits negative integer (< 0) |
| duration | Interval string (+/PnYnMnDTnHnMn.nS) |
| dayTimeDuration | Interval string day to second |
| yearMonthDuration | Interval string year to month |
| float | IEEE floating point with 7 decimals |
| gDay | Gregorian calendar day in the month (1-31) |
| gMonth | Gregorian month in the year (1-12) |
| gMonthDay | Gregorian month and day in the year |
| gYear | Gregorian year |
| gYearMonth | Gregorian year and month |
| hexBinary | Hexadecimal representation of a binary |
| NOTATION | Enumeration of XML entities |
| QName | Full qualified name with namespace (one colon!) |
| normalizedString | String without red space (cr-lf-tab) |
| token | Identifier name |
| language | Language code |
| name | Name with check on first char |
| NCName | Non-colonized name |
| ENTITY | Non-colonized name, no special implementation! |
| ID | Non-colonized name, no special implementation! |
| IDREF | Non-colonized name, no special implementation! |
| NMTOKEN | Token, no special implementation! |
| time | Time string as in "HH:MM:SS" |
| ENTITIES | List of ENTITY, seperated by one space |
| IDREFS | List of IDREF, seperated by one space |
| NMTOKENS | List of NMTOKEN, seperated by one space |
| anyType | Any type in this list |
| anySimpleType | Any type in this list |
| anyAtomicType | Any type in this list |
|  |  |
| **PREVIOUSLY ALREADY IMPLEMENTED IN MARLIN 3 and earlier** | |
| string | String of characters |
| integer | 32 bits signed integer |
| **boolean** | Boole logic as in: true or false, 1 or 0 |
| double | IEEE floating point with 15 decimal places |
| base64Binary | Base64 representation of binary |
| dateTime | "YYYY-MM-DDTHH:MM:SS.S" timestamp string + timezone |

The following WSDL checks have also been implemented:

- whitespace Preserve/replace/collapse rules

- pattern Regular expression matching

- totalDigits Max total digits of a number

- fractionDigits Max total fractional digits of a number

- minInclusive For all relevant datatypes

- minExclusive For all relevant datatypes

- maxInclusive For all relevant datatypes

- maxExclusive For all relevant datatypes

In order to fullfill the min/max in- and exclusive rules, a new XMLTemporal class has been added. This class contains a partial implementation of date, time, timestamp and durations. The implementation only calculates a ordinal number to parse min/max values.

**Command bus**

On top of the existing threadpool a ‘CommandBus’ class has been implemented. The class implements a publish/subscribe pattern. Functions, classes and modules can register to a command with the “SubscribeCommand” method. More than one subscriber can subscribe to a command. Whenever a functionality in the server publishes a command with the “PublishCommand” method, all subscribers will get called on their own callback function with their own thread from the threadpool.

The inverse of the subscribe is the “UnSubscribe” method. The unsubscribe does not remove anything from the threadpool queue, so all previously registered published commands will still execute.

**Cleanup jobs**

The threadpool has been extended with so called 'cleanup jobs'. One or more cleanup jobs can be registered that will be called after the threadpool has been drained and stopped. The cleanup jobs will be called from the main thread of the threadpool, in order of registration. These cleanup jobs can handle any functionality for the application in the closing phase of that application.

**Running in the ‘Hosted Web Core’**

For developers it can be quite cumbersome to develop an application while running in IIS. This originates from the fact that one simply cannot press ‘run’ in Visual Studio in order to run and debug the server app. What normally must be done is going to the IIS admin app, starting the application pool, then attaching the debugger of Visual Studio to the application pools process (one of the w3wp.exe processes) and then waiting for a breakpoint to occur.

To lighten the developers burden, Microsoft has created the so called ‘hosted web core’.