

Mobile Web Server: Dynamic Modules

Version 1.0; May 29, 2007

Mobile Web Server

NOKIA

Copyright © 2007 Nokia Corporation. All rights reserved.

Nokia and Forum Nokia are registered trademarks of Nokia Corporation. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. Other product and company names mentioned herein may be trademarks or trade names of their respective owners.

Disclaimer

The information in this document is provided “as is,” with no warranties whatsoever, including any warranty of merchantability, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample. This document is provided for informational purposes only.

Nokia Corporation disclaims all liability, including liability for infringement of any proprietary rights, relating to implementation of information presented in this document. Nokia Corporation does not warrant or represent that such use will not infringe such rights.

Nokia Corporation retains the right to make changes to this specification at any time, without notice.

License

A license is hereby granted to download and print a copy of this specification for personal use only. No other license to any other intellectual property rights is granted herein.

Contents

1	Introduction.....	5
2	Handling dynamic modules in MWS.....	5
2.1	Building modules.....	5
2.2	Apache import libraries.....	6
2.3	Enabling import.....	6
3	Terms and abbreviations.....	7
4	Evaluate this resource.....	7

Change history

May 29, 2007	Version 1.0	Initial document release

1 Introduction

This document describes the usage of dynamic modules in the Mobile Web Server (MWS).

The document is intended for readers who wish to use additional dynamic modules with the Mobile Web Server.

The Mobile Web Server is an http server for mobile phones. The server can be used to share data over the Internet for various clients, such as Web browsers and other applications using the HTTP protocol.

2 Handling dynamic modules in MWS

The MWS is compiled with the ARM RealView® RVCT compiler because the dynamic modules will not work with an MWS that has been compiled with GCC. In most existing modules it is too large and hence those modules must be built using RVCT. There are a number of issues that have to be taken care of in order to enable dynamically loadable Apache modules.

2.1 Building modules

If a module is linked into Apache itself, the module structure is listed in the `ap_prelinked_modules`, and `ap_preloaded_modules` in `httpd-2.0/os/symbian/modules.c`, the module can be built as a static library and subsequently linked in with the actual Symbian OS server in whose context Apache is run. Now, if you want to be able to dynamically load the module, it has to be built into a DLL, which has a number of implications:

- The module must have an UID.
- The module must be linked against Apache import libraries.
- The module must export one specific function.

Basically, what earlier looked something like:

```
TARGET          mod_xyz.lib
TARGETTYPE     LIB
SYSTEMINCLUDE  \epoc32\include
SYSTEMINCLUDE  \epoc32\include\libc
SYSTEMINCLUDE  \epoc32\include\apache\httpd\include
SYSTEMINCLUDE  \epoc32\include\apache\httpd\srclib\apr\include
SYSTEMINCLUDE  \epoc32\include\apache\httpd\srclib\apr-util\include
SYSTEMINCLUDE  \epoc32\include\apache\httpd\os\symbian
USERINCLUDE    ..\http
SOURCEPATH     .
SOURCE         mod_xyz.c
```

Must be turned into

```
TARGET          mod_xyz.so
TARGETTYPE     DLL
UID            <KSharedLibraryUid> <a-module-specific-unique-uid>
#if defined(WINS)
deffile ..\bwins.def
#else
deffile ..\eabi.def
#endif

CAPABILITY     LocalServices NetworkServices ReadUserData WriteUserData
UserEnvironment
```

```

SYSTEMINCLUDE  \epoc32\include
SYSTEMINCLUDE  \epoc32\include\libc
SYSTEMINCLUDE  \epoc32\include\apache\httpd\include
SYSTEMINCLUDE  \epoc32\include\apache\httpd\srclib\apr\include
SYSTEMINCLUDE  \epoc32\include\apache\httpd\srclib\apr-util\include
SYSTEMINCLUDE  \epoc32\include\apache\httpd\os\symbian
USERINCLUDE    ..\http
SOURCEPATH     .
SOURCE         mod_xyz.c

LIBRARY        httpd.lib
LIBRARY        aprutil.lib
LIBRARY        apr.lib
LIBRARY        estlib.lib
LIBRARY        euser.lib
LIBRARY        ...

```

Because Apache does not have a predefined UID value, `κSharedLibraryUid` is used. The *a-module-specific-unique-UID* is a UID reserved for that particular module and *capabilities-of-apache* are at least the capabilities of the Web server executable.

2.2 Apache import libraries

Modules use functionality that resides in Apache. Consequently, when linking those modules into DLLs, they must have some libraries to link against.

In practice this means that instead of building the static libraries that `httpds.exe` is linked against, you must build 3 DLLs; one for *apr*, one for *apr-util*, and one for *httpd*, the generic functionality of Apache.

Effectively you will have three DLLs with the accompanying import libraries `httpd.lib`, `apr.lib`, and `aprutil.lib`, and one static library `apache.lib`, containing the entry point of Apache. The dynamically loadable modules link against the DLL import libraries, and the Symbian OS server in whose context Apache is run links against all four.

2.3 Enabling import

Apache loads the module structure by name. That is, when you enable a module in `httpd.conf` with a line like

```
LoadModule auth_module mod_auth.so
```

Apache loads the DLL named `mod_auth.so` and then attempts to load the symbol with the name *auth_module*, which it assumes to be of struct module type.

Symbian OS does not support importing by name, so that is not possible. Consequently, some manual assistance is needed. This is handled so that every module exports one single function, using ordinal 1, whose prototype is:

```
EXPORT_C int ap_lookup_symbol(const char* name, void** symbol)
```

That is, when you give a name of a symbol as input, the function puts the address of that symbol in **symbol* and returns `APR_SUCCESS`, otherwise it returns `APR_ENOENT`.

Doing all that manually would be very inconvenient, so the necessary infrastructure has been built so that the **only** modification an existing module needs is the addition of the following line to its implementation:

```
AP_EXPORT_1_SYMBOL(name_of_module)
```

So, apart from a few exceptions, the `.def` file of all modules looks like:

```
EXPORTS
    ap_lookup_symbol @ 1 NONAME
```

3 Terms and abbreviations

Term or abbreviation	Meaning
MWS	Mobile Web Server

4 Evaluate this resource

Please spare a moment to help us improve documentation quality and recognize the resources you find most valuable, by [rating this resource](#).