OpenWorld

user's and developer's guide

to my family ...

CONTENT

USER'S GUIDE

Introduction 5
Installation 6
The reload function 6
OpenWorld internal commands 7
OpenWorld plug-ins 9
PIXO support 12
Tips 12

DEVELOPER'S GUIDE

Introduction 14
OpenWorld API 15
AppleScript plug-in 17

Acknowledgments 18
Contact information 19

USER'S GUIDE

1

INTRODUCTION

OpenWorld is a plug-in for any HTTP server compatible with the W*API 1.1 (like WebSTAR, AppleShare IP, WebTen or Quid Pro Quo). It adds server side include capabilities to your server and also allows you to extend the HTML language by writing your own tags in C, in AppleScript or in Perl so you can report every kind of dynamic information in your pages.

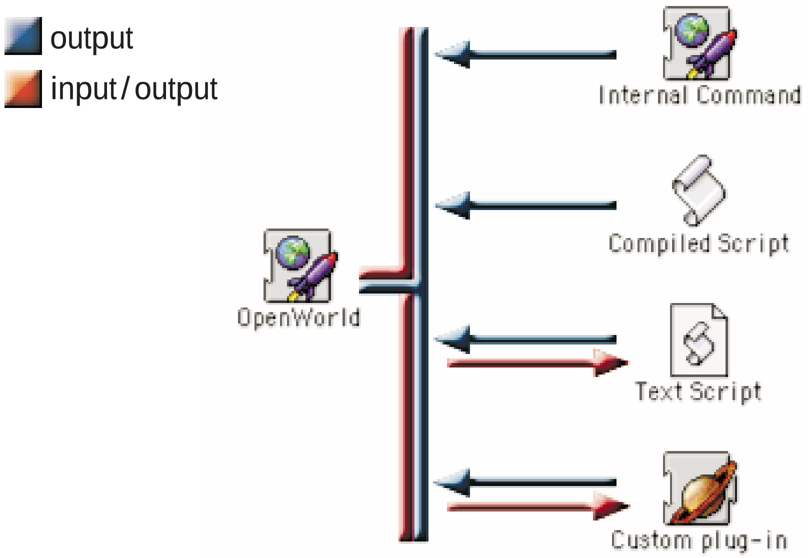
During the startup process OpenWorld loads all the plug-ins located inside the OpenWorld plug-ins folder and whenever an user requests an “.ow” page, OpenWorld loads the page, executes all the plug-ins encountered inside it and returns the resulting page to the client.

You can create many kinds of custom plug-ins. You can write plug-ins using the C language by following the simple API provided in this documentation and you can also write plug-ins using AppleScript, saving them as a compiled script if you don’t need dynamic information from the server or as an uncompiled script (text) if you do need such dynamic information. In addition to this, OpenWorld provides a number of internal commands (that you can write directly into the HTML pages) which are described in this guide.

In order to execute a plug-in you have to insert its name into your HTML source like a conventional HTML tag. For example this package includes a plug-in named “File Size“; to execute it you have to insert the <File Size> tag into your HTML source.

The OpenWorld internal commands, the AppleScript plug-in and some custom plug-ins are "single tag" plug-ins such as for example the <Date> tag, which simply performs some operation and returns the text to display. But there are some custom plug-ins that can work like "double tag" such as the <Upcase> </Upcase> tag that needs the text between two tags to perform the operation. For example, if you write a source like <UPCASE> welcome <OpenWorld?IP>, have fun! </UPCASE>, OpenWorld will first resolve all the nested tags and then send to the client a text that will read like: "WELCOME 195.120.139.130, HAVE FUN!.

Remember that you can pass dynamic information to your AppleScript scripts, so you have a complete new way to easily develop powerful new HTML tags.



INSTALLATION

To install OpenWorld, follow these steps:

1) Quit your WebSTAR (or compatible) server application.
2) Copy the file "OpenWorld" into the Plug-Ins folder in the same folder as your WebSTAR server application.
3) Copy the folder "OpenWorld plug-ins" into the Plug-Ins folder in the same folder as WebSTAR.
4) Relaunch your WebSTAR server.

To install an OpenWorld plug-in:

1) Copy the plug-in or the AppleScript script into the OpenWorld plug-ins folder inside the WebSTAR's Plug-Ins folder.
2) Send a "Reload" command to OpenWorld.

Once OpenWorld is installed, all the files with the suffix ".ow" will be filtered.

THE RELOAD FUNCTION

To install new OpenWorld plug-ins you don’t have to quit and relaunch WebSTAR, simply place the new plug-in(s) inside the “OpenWorld plug-ins" folder and then send a reload command to OpenWorld.

To send this command, simply create an empty file named “reload.ow” and then use a Web Browser to request this file from your server. OpenWorld will intercept the request and will unload and then reload all its plug-ins so you don’t need to interrupt your web service.

I suggest you save the "reload.ow" file in a protected directory to prevent unauthorized users from accessing it and needlessly sending the reload message to your server.

OPENWORLD INTERNAL COMMANDS

OpenWorld has a set of predefined internal commands not build as a plugins that can be very useful to obtain dynamic information from the server. These commands can be used to pass information either to other custom plug-ins or to an AppleScript script.

The following is a complete list of available internal commands, the type of the command is selected through the search argument.

<OpenWorld?path>	Path argument portion of URL
<OpenWorld?search>	Search argument portion of URL
<OpenWorld?username>	Username for current authentication scheme
<OpenWorld?password>	Password for current authentication scheme
<OpenWorld?from>	"From" header field
<OpenWorld?client_domain>	Domain name of client
<OpenWorld?server_domain>	Domain name of server
<OpenWorld?port>	Port number server is listening on
<OpenWorld?refer>	"Referer" HTTP header field value
<OpenWorld?agent>	"User-Agent" HTTP header field value
<OpenWorld?ip>	Client's IP address
<OpenWorld?http_request>	Entire HTTP request as received from the client
<OpenWorld?mime>	MIME type of requested URL as determined by server
<OpenWorld?header>	Preconstructed response header field for server name
<OpenWorld?server_path>	Local file system path to the server's document tree

<OpenWorld?file_path>	Local file system name for the requested file
<OpenWorld?mod>	True / false if file has been modified since last client request
<OpenWorld?date>	If-modified-since date string from HTTP header field
<OpenWorld?security>	Name of security realm (if any) protecting requested file
<OpenWorld?connection_id>	Unique ID for this HTTP connection
<OpenWorld?execution_id>	Unique ID for this execution of the server

EXAMPLES

You can use the OpenWorld internal commands both directly in your HTML source or in your AppleScript plug-in.

HTML example:

```
...
<P>
Return to the <A HREF=”<OpenWorld?refer>”>previous page</A>
</P>
...
```

in this example, the link previous page points back to the last visited page.

AppleScript example:

```
if “[OpenWorld?ip]“=”195.120.139.130” then
        return “Welcome Marco Bambini“
else
        return “Welcome [OpenWorld?ip] from [OpenWorld?address]“
end if
```

By writing an AppleScript such as this and saving it as an uncompiled script you can build your own customized HTML tag. For example, if you name this script “Welcome”, you can then use the tag ... <Welcome> ... directly in your HTML source and whenever it is encountered OpenWorld will execute the AppleScript and return the result directly to the HTML page.

OPENWORLD PLUG-INS

This chapter describes you how to use the simple plug-ins included in this version of OpenWorld. Keep in mind that the plug-ins included in this package are only examples of what you can do with this powerful new way to build HTML tags, I encourage you to try to write your own plug-ins.

Examine the file "unregistered.ow" included in this package to better understand the plug-ins usage.

This version of OpenWorld comes with the following 16 plug-ins:

Single tag plug-ins:

<Author>
Simply returns the OpenWorld's author name (Marco Bambini).

<Modification date?format>
Returns the modification date of the requested file.
The parameter “format“ is used to compile the output, “format“ can be:

short: example 10/1/98
long: example Monday, February 1, 1998
abbr: example Mon, Feb 1, 1998

If no parameter if specified, the default format is the short date.

<Modification Time>
Returns the modification time of the requested file.
The parameter “format“ is used to compile the output, “format“ can be:

sec: example 11:40:52 (include seconds)
or nothing to not include seconds

<File Size?option>
Returns the size of the requested file.
If option is not the string “null“, it returns the size of the file converted into bytes, or Kbytes or Mbytes.

example: 2 Kbytes (without the option) or 2024 if option is null

<Memory?selector>
Returns memory related information.
The parameter “selector” is used to select the type of information to return, “selector“ can be:

physical: size of the physical ram installed in the computer
logical: logical memory, physical plus virtual memory
available: size of the system available memory

If no parameter if specified, returns the available memory in the current heap (the server heap).

<Counter?page>
“page“ is the name that identifies the file whose access you want to count.

<System>
Returns the version of the operating system used on your server.

<Include?filename>
Copies the content of the file "filename" into your HTML source.
“filename“ can be the file path (relative to the WebSTAR root folder or absolute to your disk)

<Date?argument>
Returns the current date.
The parameter “format“ is used to compile the output, “format“ can be:

short: example 10/1/98
long: example Monday, February 1, 1998
abbr: example Mon, Feb 1, 1998

If no parameter if specified, the default format is the short date.

<Time?argument>
Returns the current time.
The parameter “format“ is used to compile the output, “format“ can be:

sec: example 11:40:52 (include seconds)
or nothing to not include seconds

<Server?argument>
Returns information directly from the server, argument is a four character identifier specified into the “W*API 1.1 Reference.pdf” file.
You can obtain this file directly from Starnine at the address:
http://dev.starnine.com/devresources.html

<Extract Header?HeaderName>
This plug-in enable you to extract fields directly from the Header field sent by the browser.

Example:
...
<Extract Header?User-Agent> returns “Mozilla/4.51 (Macintosh; I; PPC)“

Double tag plug-ins:

<UPCASE>...</UPCASE>
Transforms all the text included between the two tags to capital letters.

<APPLESCRIPT>...</APPLESCRIPT>
You can include AppleScript commands between these two tags, the AppleScript plug-in will compile and execute it and report the result into your pages.
For example you can write commands like these:

```
<APPLESCRIPT>
if "<OpenWorld?ip>"="someIPaddress" then
return "Welcome (whatever text you want)"
else
return "Welcome unknown user"
end if
</APPLESCRIPT>
```

<FRONTIER>...</FRONTIER>
You can include usertalk commands between these two tags, the Frontier plug-in will compile and execute it and report the result into your pages.
For example you can write commands like these:

```
<Frontier>
return 5*7
</Frontier>
```

<TOKEN?param>...</TOKEN>
This powerful plug-in enable you to extract specific information from text fileds.
It works like the strtok C function, param is in the form: number,separator

where number is the number of times to repeat the strtok and separator is a string that contains the separator’s characters

HTML example:

```
…
Browser: <token?1,/><Extract Header?user-agent></token><BR>
Version: <token?2, /><Extract Header?user-agent></token><BR>
Platform: <token?2,(;><Extract Header?user-agent></token></B>
Processor: <token?4, )><Extract Header?user-agent></token>
…
```

You’ll have:

```
Browser: Mozilla
Version: 4.5.1
Platform: Macintosh
Processor: PPC
```

Note that the command <Extract Header?user-agent> returns a string like “Mozilla/4.51 (Macintosh; I; PPC)“

PIXO SUPPORT

The PIXO ("Plug-In Cross-Over") API is a 'standardized' way for Web server plug-ins to communicate and cooperate for better performance, advanced integration, and extra Web site coolness.

In the beginning, there were no WebSTAR plug-ins; all 'external' processing was called through standardized AppleEvents. The server sent the AppleEvent, and the CGIs responded.

Sometimes it was useful for one CGI to call another, and this was fairly straightforward: the 'caller' CGI sent an AppleEvent to the 'callee,' and the callee returned its results to the caller. For example, the NetCloak CGI was famous for being able to call the Tango CGI with an AppleEvent, and process any 'special' tags found in the Tango output. Webmasters could build interesting and powerful sites using various combinations of CGIs.

Then plug-ins came along. The server calls each plug-in directly and the plug-in returns the results directly to the server. Since plug-ins are all loaded directly into the server, there has not been any way to combine the features of plug-ins the way CGIs could be combined. Now that there are more plug-ins than ever and they're doing more and more interesting things, it has once again become critical that plug-ins have some way of calling each other.

PIXO ("plug-in cross-over") is an API that allows plug-ins loaded into the same server to call each other just as CGIs could but without the overhead of AppleEvents or TCP/IP. Using PIXO, modern Web server plug-ins can now call upon each other's services directly, using a simple, high-performance API, and can be combined to create extremely powerful Web sites, services, and applications.

OpenWorld acts as both a PIXO SOURCE and PROCESSOR, its action name is OPENWORLD_PIXO.

TIPS

All the messages reported by OpenWorld are located inside its resource fork in resources of type TEXT so you can customize them.

ID 12000	- File not found error
ID 13000	- Low memory error
ID 14000	- General error
ID 15000	- Reload command
ID 16000	- Unauthorized access

DEVELOPER'S GUIDE

2

INTRODUCTION

The OpenWorld API implements a new technique for developers to extend the HTML language. OpenWorld scans all the HTML source and when it encounters a TAG it calls the plug-in with the same name.

OpenWorld passes to the plug-in a pointer to a structure like this:

```
typedef struct
{
     long            message;
     long            version;
     long            customData;
     char            *search;
     const char      *inputText;
     char            *outputText;
     long            dim;
     FSSpec          myFile;
     short           fRef;

     /* Reserved parameters, don't use them */
     ...
} data;
```

Fields

message	type of operation to perform
version	the plug-in version (to set in the init message)
customData	used to store static data
search	this is the search argument of the tag (tag?search) (null terminated)
inputText	this is the text passed by OpenWorld to the double tag plug-in (null terminated)
outputText	the text returned from the plug-in
dim	on input it contains the size of the inputText buffer, on output you have to set this field with the size of the outputText buffer
myFile	this is the complete FSSpec record of the open file
fRef	this is the field reference number of the open file

How plug-ins are invoked

When OpenWorld is loaded, it sends an init message (in the message field) to all the plug-ins inside the “OpenWorld plug-ins” folder. At this time the custom plug-ins have to set the version field with the costant kVersion defined in the “OpenWorld_plugin.h“ file and optionally can set the customData field. This field will be saved by OpenWorld and passed to the plug-in at every call.
Every time OpenWorld reaches a tag with the same name as the plug-in, it sends a run_message. This is the core function of the plug-in which is responsible for setting the resulting text to be displayed in the HTML. After the run_message OpenWorld sends a stop_message that you can use to dispose of all the temporary memory used by your plug-in.
At the end, when the user quits WebSTAR, OpenWorld sends an end_message in order to allow plug-ins to dispose of all the remaining buffers or to close all open files.
Remember that the init and the end messages are sent only one time, while the run and the stop messages are sent every time the plug-in is invoked.

The structure of a plug-in looks like this:

```
Boolean main (dataPtr param)
{
      switch (param->message)
      {
            case init_message:
            {
                  param->version=kversion; // initialize the version field
                  param->customData=kUnused; // optional custom static data
            }
            break;

            case run_message:
            {
                  //
                  // write here your core function
                  //
            }
            break;

            case stop_message:
            {
                  //
                  // free all the buffers allocated in the run message
                  //
            }
            break;

            case end_message:
            {
                  // deallocate all the buffers
                  // close all open files
                  // stop all the others operations
            }
            break;
      }

      return kSingleTag; //or kDoubleTag
}
```

As you can see, the structure is very simple and intuitive, so it's easy to develop OpenWorld's plug-ins. The only thing you have to observe is that the retun value of a single tag plug-in must be the costant kSingleTag, or kDoubleTag for a double tag plug-in (these costants are defined in the “OpenWorld_plugIn.h“ file).

Custom plug-ins are code resources of type mPPC (for PowerPC processor) or m68k (for 680x0 processor, not supported in this version), the AppleScript plug-ins can be either compiled scripts or text scripts. The name of the plug-in is the name of the TAG they implement (excluded the characters after the dot).

You can pass any kind of data to the custom plug-in using the search argument (example: TAG?data).

void *OW_NewPtr (dataPtr param, long size);

Allocates and returns a pointer to "size" bytes of data. Returns NULL if memory allocation fails. DON'T use any other techniques to allocate memory within your plug-in code if you can avoid it. All memory should be allocated using this call and disposed of using the corresponding OW_DisposePtr call.

param -> the parameter passed to your main entrypoint function.
size -> integer which represents the requested number of bytes to allocate

Returns a pointer to the requested memory block or NULL if the request couldn't be satisfied

void OW_DisposePtr (dataPtr param, void *ptr);

Deallocates a block of memory originally allocated with OW_NewPtr. Do NOT mix and match OW_NewPtr and OW_NewHandle or vice versa!!! OW_DisposePtr checks for NULL pointer before it executes the memory deallocation.

param -> the parameter passed to your main entrypoint function.
ptr -> a pointer to a block of memory allocated by OW_NewPtr that will be deallocated.

Returns nothing

void *OW_NewHandle (dataPtr param, long size);

Allocates and returns an O/S-specific memory accessor (e.g., a Handle under Mac O/S) to "size" bytes of data. Returns NULL if memory allocation fails. Do NOT use any other techniques to allocate memory within your plug-in code. Memory allocated using this call should be disposed of using the corresponding OW_DisposeHandle call.

param -> the parameter passed to your main entrypoint function.
size -> integer which represents the requested number of bytes to allocate

Returns an handle to the requested memory block or NULL if the request couldn't be satisfied

void OW_DisposeHandle (dataPtr param, void *ptr);

Deallocates a block of memory originally allocated with OW_NewHandle. Do NOT mix and match OW_DisposePtr and OW_DisposeHandle or vice versa!!!
OW_DisposeHandle checks for NULL pointer before it executes the memory deallocation.

param -> the parameter passed to your main entrypoint function.
data -> a pointer to an handle to the memory block to deallocate.

Returns nothing

char *OW_GetServerInfo (dataPtr param, WSAPI_ParamKeywords key, unsigned long *dim);

Extracts information from the server, it can allocate a block of memory.

param -> the parameter passed to your main entrypoint function.
key -> the type of parameter to extract (you can see all the available parameters in the W* API 1.1 Manual, that you can download from: **http://dev.starnine.com/devresources.html**)
dim -> the lenght of the returned string

Returns a pointer to a string (null terminated) that contains the requested information.

see the W API Documentation for the returned type

void DisposeServerInfo (dataPtr param, void *ptr);

Disposes the information allocated with the OW_GetServerInfo routine.

param -> the parameter passed to your main entrypoint function.
data -> a pointer to a block of memory allocated by OW_GetServerInfo that will be deallocated.

Returns nothing.

char *OW_TextToHTML(dataPtr param, char *s, long *dim);

Convert the string s of lenght dim to an HTML string, it can allocate a block of memory.
This function replaces all the occurences of ASCII characters like “ì“, or “à“ to the corrisponding HTML strings &igrave; or &agrave;

param -> the parameter passed to your main entrypoint function.
s -> the string to convert
dim -> on input is the lenght of the string to convert, on output is the lenght of the converted string

Returns the converted string (null terminated) or NULL if an error occurs.

Remember to set the dim parameter before use this function.

APPLESCRIPT PLUG-IN

With the advent of MacOS 8.5 AppleScript has been written in PowerPC native code, rapresenting a new way to easily create fast and powerful pieces of software.
This is why I have adopted it as a language to develop new and exciting OpenWorld plug-ins.

The easiest way to write an AppleScript plug-in is to save it as a compiled script. In this format, your script can't have access to dynamic information.

Another possibility is to save it as a text script (not compiled). In this way your script can execute all OpenWorld plug-ins so it can have access to all the dynamic information it needs.
In order to execute an external plug-in (or an internal command) inside an AppleScript script, you have to write the plug-in's name (or the internal command's name) inside the brackets '[]' characters.

EXAMPLES

If you need to report the client's IP address (an internal command) you can write a script like:

```
return "Your IP address is [OpenWorld?ip]"
```

The script must be saved as text and report its name into the HTML source to be able to execute it.

Otherwise, if you want to return information about the size of the requested file (an OpenWorld plug-in called "File Size"), you can write:

```
set mySize to [File Size?null]

if mySize>1024 then
        return "Wow! this is a big file, bigger then 1K"
else
        return "This is a small file"
end if
```

Remember that you have to save your script as text and report its name into the HTML source to be able to execute it.

ACKNOWLEDGMENTS

A very special thanks to:

Chuck Shotton
Clearway Technologies
Grant Hulbert

Special thanks to my friend Julius Caesar Toledo

Special thanks to my family: Pietro, Donata and Monica

Special thanks to my spiritual guide zTom

A big greeting to my friends: Cinghio, Vito, Devis, Sara, Ivana, Tox, Ruben, Spaggio, Gallo, Monia, Matteo, Leona, Paolo, Wren and so on.

Special thanks to my ...**Landy** (my personal star)

and a big thanks to all the people who believe in me.

CONTACT INFORMATION

Marco Bambini
Via Piave n.21
46015 Cicognara (MN)
ITALY

e mail: **marco.bambini@kagi.com**
web site: **http://www.geocities.com/SiliconValley/Network/7185/**

A REAL BIG THANKS TO ALL THE REGISTERED USER.