NAME

webasm — Web Assembler creates complete HTML or XML documents from reusable pieces.

SYNOPSIS

webasm [options] command [*file ...*]
webasm **-source** path/to/source **-output** path/to/output **build**
webasm **-config** webasm.xml **build**

DESCRIPTION

Web Assembler creates complete HTML documents from reusable document fragments, the output of scripts, evaluated JavaScript, rendered text, and other information. It may be loosely thought of as a compiler for HTML and other text-based document formats. Web Assembler can be used to eliminated redundant markup and to assist with common but time consuming tasks. Unlike server-side web application platforms, however, Web Assembler does this work only once - when a project is built - rather than for every request.

For more on Web Assembler's grand scheme and design, see `http://woltergroup.net/webasm`.

GLOBAL OPTIONS

Global options apply to all commands. Most options that can be set on the command line can also be specified in a configuration file (often, more easily). See the configuration section for more.

-quiet Be less verbose. Only error output is displayed. Mutually exclusive with verbose (**-verbose**) and debugging (**-debug**) modes.

-verbose
Be more verbose. Additional nonessential output is displayed. Overrides quiet mode (**-quiet**).

-debug Be extremely verbose. A huge amount of information is displayed. Generally, debugging mode is not useful unless you are trying to diagnose a problem with Web Assembler. Implies verbose mode (**-verbose**) and overrides quiet mode (**-quiet**).

-config *config path*
Specify the path to a configuration file to be used. When absent, only command line arguments are considered.

-source *source root*
Specify the root source path. This is the path under which source files to assemble are located. Only files with the extensions selected by types (**-types**) or paths matching a pattern (**-match**) are evaluated.

-output *output root*
Specify the root output path. This is the path under which processed files are placed. Processed files are saved a the same relative location under the *output root* as the source root (**-source**).

-types *a,b,c*
Specify a list of file type extensions to match under the source root (**-source**) when looking for files to assemble. More than one file type can be specified by defining a list of extensions (e.g. "html,htm,xml"). Hidden files will be ignored (even if they match these extensions) unless **-include-hidden** is set. If this is omitted, the types "html,htm" are used. For more control over which files are processed, match files by pattern (**-match**).

-match *pattern*
Process only files that match the Perl compatible regular expression *pattern* under the source root (**-source**). For example, the pattern '(?<!.frag).html$' could be used to process only files ending in ".html" but not ".frag.html" in order to exclude fragment HTML files that are included into other documents, but shouldn't be processed on their own. Selecting files to process by matching is mutually exclusive with selecting by file extensions (**-types**).

-define *name=value*
Define a property which will be made globally accessible to directives. The property is expressed as a key/value pair in the form: "name=value" (e.g., **-define** greeting="Hello, there!"). A property defined on the command line overrides a property with the same name that was defined in a configuration file.

-include-hidden
Don't ignore hidden files under the *source root* (**-source**). By default hidden files are not processed even if their extensions match the types set.

-collapse-white
Collapse whitespace in verbatim content. *This feature is experimental and may be removed in a later release*. When enabled, whitespace in processed files is reduced. This can be useful in generating more compact HTML at the expense of legibility.

-help Display help information and quit.

COMMANDS

build Assemble project files. By default, the entire project hierarchy under the source root (**-source**) is traversed and any files matching filtered file type extensions are processed and output to the corresponding path under the output root (**-output**).

plan Display a listing describing which files would be processed if **build** were to be run, but don't actually process any files. This can be used to confirm that **webasm** will process the files you expect.

parse Parse project files and display a listing of their structure. This is mainly only useful for debugging.

DIRECTIVES

The following is a comprehensive list of built-in directives. You may also create your own custom directives, which are **execute** directives mapped to a special tag name.

insert Insert the entire contents of another file into the document. Any directives in the inserted file are processed first and the inserted file has the same runtime state (i.e., properties, JavaScript objects) as the calling **insert** directive itself.

resource Path to the resource (file) to insert. The path is expressed as relative to the file in which the directive is found.

Supports nested directives: **define.**

import Evaluate JavaScript in another file. The file is evaluated immediately and any objects created are subsequently available to other directives. This directive is similar to **script** with JavaScript source in an external file.

resource Path to the JavaScript resource (file) to import. The path is expressed as relative to the file in which the directive is found.

script Evaluate inline JavaScript. JavaScript source nested in this directive is evaluated immediately and any objects created are subsequently available to other directives. This directive is similar to **import** with JavaScript source in the same file.

value The JavaScript expression to evalute.

property
Print a property. If no such property is defined, this directive is replaced with nothing.

name The name of the property to print.

print Print the result of a JavaScript expression. If the expression evaluates to undefined, "undefined" is printed.

value The JavaScript expression to evalute. If this attribute is missing, the content of the tag is evaluated in its place.

if,elseif,else
Conditional output. The directives **if**, **elseif**, and **else** are *cooperating* directives; that is, they work with each other and must be declared relative to each other. These directives work in the same way conditional statements work anywhere else. Only the content of the directive that evalutes to true (or failing that, the **else** directive, when present) is output.

The directives **if** and **elseif** require a test expression.

test The JavaScript expression to evalute. If this expression evaluates to true, the contents of the tag is output.

Any directive may be nested in **if**, **elseif**, and **else**.

escape Escape content. Any non-alphanumeric characters nested in **escape** are replaced with HTML entity-escaped equivalents. For example, the copyright character is replaced with "©"

Any directive may be nested in **escape**.

execute Execute another script or program. The content of the directive (if any) is written to the standard input of the command and the standard output of the command is output in place of the directive.

command Path to the command to execute. The path is expressed as relative to the file in which the directive is found.

arg Any number of *arg* attributes may be used to provide command-line arguments which are used to invoke the command. Arguments are passed to the command in the order their attributes appear in the **execute** tag.

In addition to command-line arguments, the following variables are put in the environment of the command:

WEBASM_SOURCE_ROOT
The source root directory defined with **-source** or in a configuration file.

WEBASM_OUTPUT_ROOT
The output root directory define with **-output** or in a configuration file.

WEBASM_QUIET
Whether or not **webasm** is running in quiet (**-quiet**) mode. The value will be "true" or "false".

WEBASM_VERBOSE
Whether or not **webasm** is running in verbose (**-verbose**) mode. The value will be "true" or "false".

WEBASM_DEBUGGING
Whether or not **webasm** is running in debugging (**-verbose**) mode. The value will be "true" or "false".

WEBASM_FORCE
Whether or not **webasm** should force evaluation of directives (**-force**) and not rely on any cached content. The value will be "true" or "false".

WEBASM_INCLUDE_HIDDEN_FILES
Whether or not hidden files should be ignored when traversing the source root. The value will be "true" or "false".

*WEBASM_TAG_ATTR_***name**
Where **name** is the name of an attribute defined in the **execute** directive. For example, if the attribute *length* were defined with the value "100", the environment variable *WEBASM_TAG_ATTR_length* will be declared with the value "100".

Attribute names that contain non-alphanumeric characters are mangled: non-alphanumeric characters are replaced with a single underscore. For example, the attribute *no-line-breaks* will result in the environment variable *WEBASM_TAG_ATTR_no_line_breaks*.

The *command* attribute is declared in this manner as well, but *arg* attributes are not.

*WEBASM_PROPERTY_***name**
Where **name** is the name of a property. For example, if the property *greeting* were defined with the value "Hello", either via the (**-define**) flag, in a configuration file, or in the environment when invoking **webasm**, the environment variable *WEBASM_PROPERTY_greeting* will be declared with the value "Hello".

Property names that contain non-alphanumeric characters are mangled: non-alphanumeric characters are replaced with a single underscore. For example, the property *the-greeting* will result in the environment variable *WEBASM_PROPERTY_the_greeting*.

text Set text in an image. This directive can be used to render an image of text; for example, to display header text set in an unusual font that cannot be relied upon to be present on every user's machine. Text is rendered via Apple's excellent CoreText.

path The path to the file the image should be written to. The path is expressed as relative to the output counterpart of the file in which the directive is found.

font-name The name of the font in which to set type.

font-size The size of the text to set, in pixels.

foreground
The foreground color to set text in, expressed as a hex triplet with an optional leading '#' (e.g. "#ff0000" for red.)

background
The background color to set text in, expressed as a hex triplet with an optional leading '#' (e.g. "#ff0000" for red.) When not specified, a transparent background is used.

width The width of the image to generate. When omitted, the generated image will have the smallest width which can fit the text without breaking lines. When specified, text will be broken as necessary to fit the width.

height The height of the image to generate. When omitted, the generated image will have the smallest height which can fit the text.

padding The amount of padding to surround the text content with, in pixels. Padding is expressed in CSS style where 4, 2, or 1 values separated by spaces may be used to denote the the directions each value corresponds to. Four values refer to: *top*, *right*, *bottom*, *left*; in that order. Two values refer to: *top*/*bottom*, *right*/*left*. One value refers to all four directions.

For example, the padding "10 4 12 12" means: *top* = 10, *right* = 4, *bottom* = 12, and *left* = 12. The padding "10 4" is the equivalent of "10 4 10 4"; and "10" is the equivalent of "10 10 10 10".

BUILT-IN CUSTOM DIRECTIVES

Web Assembler includes some built-in custom directives. These are just **execute** directives which are mapped to a tag name and included in the **AssemblerCore** framework bundle.

markdown

Format text with Markdown, John Gruber's text-to-HTML processor. You can define *arg* attributes to pass arguments to the underlying script and modify it's handling.

For more on Markdown, refer to:
`http://daringfireball.net/projects/markdown/`

highlight

Format syntax-highlighted source code with Pygments.

language The programming language source code should be interpreted as. Any language supported by Pygments can be used. For example, to format as C source, the language "c" is used.

To insert a file of syntax-highlighted source, you might do something like:
<w:highlight language="c"><w:insert resource="hello.c" /></w:highlight>

For more on Pygments, refer to:
`http://pygments.org/`

lipsum Insert placeholder text (a.k.a. "Lorem Ipsum"). The text will be broken into sentences and paragraphs randomly in an attempt to minic actual content.

words The number of words to generate. If omitted, 50 words are output.

USING WEB ASSEMBLER DIRECTIVES IN YOUR HTML

Web Assembler processes a set of input source files and generates a set of output processed files by interpreting zero or more *directives* interspersed in the source (a source file with zero directives is output unchanged).

The underlying concept behind directives is very much similar to that of *JSP* or *PHP* markup. Directives are simply snippets of XML which are placed in source HTML to denote some action be taken at that point in the file. The rest of the content in source files (anything other than Web Assembler directives) is copied into the processed files verbatim.

Web Assembler uses a specialized XML parser to interpret files so that normal HTML or XML is not parsed. Directives use the XML namespace **w** to distinguish themselves from other markup.

For example, a simple directive which will replace itself by copying the contents of another file into its location would be declared as follows:

```
<w:insert resource="html/file.html" />
```

Tags can be nested, in which case an enclosing tag will operate on the processed content of any enclosed tags. For example, the entire contents of an inserted file can be escaped as follows:

```
<w:escape>
  <w:insert resource="html/file.html" />
</w:escape>
```

In this case, the file "file.html" is processed via the **insert** directive first, and then the contents of the **escape** directive is processed.

ABOUT

Web Assembler is designed and engineered by Wolter Group in New York City and made available free of charge under the terms of the Wolter Group End User License Agreement.

Web Assembler is provided by Wolter Group on an âAS IS" basis. Wolter Group MAKES NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THE WGNY SOFTWARE OR ITS USE AND OPERATION ALONE OR IN COMBINATION WITH YOUR PRODUCTS.