POD Translation
by *pod2pdf*

ajf@afco.demon.co.uk

Html2Wml Documentation

Table of Contents
Html2Wml Documentation

NAME

Html2Wml — Program that can convert HTML pages to WML pages

SYNOPSIS

Html2Wml can be used as either a shell command:

```
$ html2wml file.html
```

or as a CGI:

```
/cgi-bin/html2wml?url=/index.html
```

In both cases, the file can be either a local file or a URL.

DESCRIPTION

Html2Wml converts HTML pages to WML decks, suitable for being viewed on a Wap device. The program can be launched from a shell to statically convert a set of pages, or as a CGI to convert a particular (potentially dynamic) HTML resource.
Althought the result is not guarantied to be valid WML, it should the case in most common cases. Good HTML pages will most probably produce valid WML decks.To check and correct your pages, you can use W3C's softwares: the *HTML Validator*, available online at http://validator.w3.org and *HTML Tidy*, written by Dave Raggett.
Html2Wml provides the following features:

- translation of the links
- limitation of the cards size by splitting the result into several cards
- inclusion of files (similar to the SSI)
- compilation of the result (using the WML Tools, see *"LINKS"*)
- a debug mode to check the result using validation functions

OPTIONS

Please note that most of these options are also available when calling Html2Wml as a CGI. In this case, boolean options are given the value "`yes`" or "`no`", and other options simply receive the value they expect. For example, `–ascii` becomes `?ascii=yes`. See the file ***t/form.html*** for an example on how to call Html2Wml as a CGI.

Conversion Options

—ascii

When this option is on, named HTML entities are converted to US-ASCII characters using the same 7 bit approximations as Lynx (the famous text-mode browser). For example, `&copy;` is translated to "(C)", and `&szlig;` is translated to "ss". By default, this option is off, so that the entities are converted to numeric entities: `&copy;` becomes `&#169;`, and `&szlig;` becomes `&#223;`.

—collapse, —nocollapse

This option tells html2Wml to collapse redundant whitespaces, tabulations, carriage returns, lines feeds and empty paragraphs. The aim is to reduce the size of the WML card as much as possible. Collapsing empty paragraphs is necessary for two reasons. First, this avoids empty screens (and on a device with only 4 lines of display, an empty screen is quickly made). Second, Html2wml creates many empty paragraphs when converting, because of the way I programmed the syntax reconstructor. Deleting these empty paragraphs is necessary like cleaning the kitchen :-)
If this really bother you, you can desactivate this behaviour with the **—nocollapse** option.

—compile

Setting this option tells Html2Wml to use the compiler from WML Tools to compile the WML deck. If you want to create a real Wap site, you should seriously use this option in order to reduce the size of the WML decks. Remember that Wap devices have very little amount of memory. If this is not enought, use the splitting options.

—hreftmpl=*TEMPLATE*

This options sets the template that will be used to reconstruct the `href`-type links. See *"LINKS RECONSTRUCTION"* for more information.

—linearize, —nolinearize

This option is on by default. This makes Html2Wml flattens the HTML tables (they are linearized), as Lynx does. I think this is better than trying to use the native WML tables. First, they have extremely limited features and possibilities compared to HTML tables. In particular, they can‘t be nested. In fact this is normal because Wap devices are not supposed to have a big CPU running at some zillions-hertz, and the calculations needed to render the tables are the most complicate and CPU-hogger part of the HTML.

Second, as they can‘t be nested, and as typical HTML pages heavily use imbricated tables to create their layout, it‘s impossible to decide which one could be kept. So the best thing is to keep none of them.

[Note] Although you can desactivate this behaviour, and although there is internal support for tables, the unlinearized mode has not been heavily tested with nested tables, and it may produce unexpected results.

—nopre

This options tells Html2Wml not to use the `<pre>` tag. This option was added because the compiler from WML Tools 0.0.4 doesn‘t support this tag.

—srctmpl=*TEMPLATE*

This option sets the template that will be used to reconstruct the `src`-type links. See *"LINKS RECONSTRUCTION"* for more information.

Splitting Options

—max-card-size=*SIZE*

This option allows you to limit the size (in bytes) of the generated cards. Default is 1,500 bytes, which should be small enought to be loaded on most Wap devices. See *"DECK SPLITTING"* for more information.

—card-split-threshold=*SIZE*

This option sets the threshold of the split event, which can occur when the size of the current card is between `max-card-size - card-split-threshold` and `max-card-size`. Default value is 50. See *"DECK SPLITTING"* for more information.

—next-card-label=*STRING*

This options sets the label of the link that points to the next card. Default is "[>>]", which whill be rendered as "[]".

Debugging Options

—debug

This option activates the debug mode. This prints the output result with line numbering and with the result of the XML check. If the WML compiler was called, the result is also printed in hexadecimal an ascii forms. When called as a CGI, all of this is printed as HTML, so that can use any web browser for that purpose.

—xmlcheck

When this option is on, it send the WML output to XML::Parser to check its well-formedness.

DECK SPLITTING

The *deck splitting* is a feature that Html2Wml provides in order to match the low memory capabilities of most Wap devices. Many can‘t handle cards larger than 2,000 bytes, therefore the cards must be sufficiently small to be viewed by all Wap devices. To achieve this, you should compile your WML deck, which reduce the size of the deck by 50%, but even then your cards may be too big. This is where Html2Wml comes with the deck splitting feature. This allows you to limit the size of the cards, currently only *before* the compilation stage.

Currently, Html2Wml estimates the size of the card on the fly, by summing the length of the strings that compose the WML output, texts and tags. I say "estimates" and not "calculates" because computing the exact size add as many calculations. One may object that there are only additions, which is correct, but knowing the *exact* is not necessary. Indeed, if you compile the WML, most of the strings of the tags will be removed, but not all.

For example, take an image tag: `<img src="images/fog.jpg" alt="Photo of a dog">`. When compiled, the string `"img"` will be replaced by a one byte value. Same for the strings `"src"` and `"alt"`, and the spaces, double quotes and equal signs will be stripped. Only the text between double quote will be preserved.

As you see, it doesn‘t matter to know exactly the size of the textual form of the WML, as it will be always by far superior to the size of the compiled form. That‘s why I don‘t count all the characters that may be actually written.

Also, it‘s because i‘m quite lazy ;-)

Why compiling the WML deck?

If you intent to create real WML pages, you should really consider to always compile them. If you‘re not convinced, here‘s is an illustration.

Take the following WML code snipet:

```
<a href="http://www.yahoo.com/">Yahoo!</a>
```

It‘s the basic and classical way to code an hyperlink. It takes 42 bytes to code this, because it is presented in a human-readable form.

The WAP Forum has defined a compact binary representation of WML in its specification (this is what we called "compiled WML"): its binary, so you (human), can‘t read that, but your computer can. And it‘s much faster.

The previous example would be, once compiled (printed here as hexadecimal):

```
1C 4A 8F 03 y a h o o 00 85 03 Y a h o o ! 00 01
```

This only takes 20 bytes. Half the size of the human-readable form. And a Wap device can read this way faster! And of course, smaller document means less time to download.

There is a last argument, and not the less important: most Wap devices only read binary WML.

ACTIONS

Actions are a feature similar to (but with far less functionalities!) the SSI (Server Side Includes) available on good servers like Apache. In order not to interfere with the real SSI, but to keep the syntax easy to learn, it differs in very few points.

Syntax

Basically, the syntax to execute an action is:

```
<!-- [action param1="value" param2='value'] -->
```

Note that the angle brackets are part of the syntax. Except for that point, Actions syntax is very similar to SSI syntax.

Available actions

Currently, only two actions are available, but if I can implemented more on request.

include

Description

Includes a file in the document at the current point. Please note that Html2Wml doesn‘t check nor parse the file, and if the file cannot be found, will silently die (this is the same behavior as SSI).

Parameters

`virtual=url` — The file is get by http.
`file=path` — The file is read from the local disk.

Notes

If you use the file parameter, an absolute path is recommend.

fsize

Description

Returns the size of a file at the current point of the document.

Parameters

`virtual=url` — The file is get by http.
`file=path` — The file is read from the local disk.

Notes

If you use the file parameter, an absolute path is recommend.

Examples

If you want to share a navigation bar between several WML pages, you can include it this way:

```
<!-- [include virtual="nav.wml"] -->
```

Of course, you have to write this navigation bar first :-)

LINKS RECONSTRUCTION

The links reconstruction engine is IMHO the most important part of Html2Wml, because it‘s this engine that allows you to reconstruct the links of the HTML document being converted. It has two modes, depending upon whether Html2Wml was launched from the shell or as a CGI.
When used as a CGI, this engine will reconstructs the links of the HTML document so that all the urls will be passed to Html2Wml in order to convert the pointed files (pages or images). This is completly automatic and can‘t be customized for now (but I don‘t think it would be really useful).
When used from the shell, this engine reconstructs the links with the given templates. Note that absolute URLs will be left untouched. The templates can be customized using the following syntax.

Templates

HREF Template

This template controls the reconstruction of the `HREF` attribute of the `A` tag. Its value can be changed using the **—hreftmpl** option. Default value is
`"{FILEPATH}{FILENAME}{$FILETYPE =~ s/s?html?/wml/o; $FILETYPE}"`.

Image Source Template

This template controls the reconstruction of the `SRC` attribute of the `IMG` tag. Its value can be changed using the **—srctmpl** option. Default value is

```
"{FILEPATH}{FILENAME}{$FILETYPE =~ s/gif|png|jpe?g/wbmp/o;
$FILETYPE}"
```

Syntax

The template is a string that contains the new URL. More precisely, it‘s a Text::Template template. Parameters can be interpolated as a constant or as a variable. The template is embraced between curcly bracets, and can contain any avlid Perl code.
The simplest form of a template is `{`*`PARAM`*`}` which just returns the value of *PARAM*. If you want to do something more complex, you can use the corresponding variable; for example `{"foo $`*`PARAM`*` bar"}`, or `{join ‘_‘, split ’ ‘, `*`PARAM`*`}`.
You may read *Text::Template* for more information on what is possible within a template.
If the original URL contained a query part or a fragment part, then they will be appended to the result of the template.

Available parameters

URL

This parameter contains the original URL from the `href` or `src` attribute.

FILENAME

This parameter contains the base name of the file.

FILEPATH

This parameter contains the leading path of the file.

FILETYPE

This parameter contains the suffix of the file.

Examples

To add a path option:

```
{URL}$wap
```

Using Apache, you can then add a Rewrite directive so that URL ending with `$wap` will be redirected to Html2Wml:

```
RewriteRule  ^(/.*)\$wap$  /cgi-bin/html2wml.cgi?url=$1
```

To change the extension of an image:

```
{FILEPATH}{FILENAME}.wbmp
```

Note that `FILETYPE` contains all the extensions of the file, so its name is ***index.html.fr*** for example, `FILETYPE` contains "`.html.fr`".

CAVEATS

Currently, only the well-formedness of the resulting WML can be tested, not its validity.
Inverted tags (like "bold italic") may produce unexpected results. But only bad softwares do bad stuff like this.

LINKS

Resources

The WAP Forum

This is the official site of the WAP Forum. You can find some technical information, as the specifications of all the technologies associated with the WAP.
[http://www.wapforum.org/]

WAP.com

This site has some useful information and links. In particular, it has a quite well done FAQ.
[http://www.wap.com/]

The World Wide Web Consortium

Altough not directly related to the Wap stuff, you may find useful to read the specifications of the XML (WML is an XML application), and the specifications of the different stylesheet languages (CSS and XSL), which include support for low-resolution devices.
[http://www.w3.org/]

MobiliX

This web site is dedicated to Mobile UniX systems. It leads you to a lot of useful hands-on information about installing and running Linux and BSD on laptops, PDAs and other mobile computer devices.
[http://www.mobilix.org/]

Programmers utilities

HTML Tidy

This is a very handful utility which corrects your HTML files so that they conform to W3C standards.
[http://www.w3.org/People/Raggett/tidy]

WML Tools

This is a collection of utilities for WML programmers. This include a compiler, a decompiler, a viewer and a WBMP converter.
[http://pwot.co.uk/wml/]

WML browsers and Wap emulators

wApua

wApua is a WML browser written in Perl/Tk. I use it for most of my tests as it is the fatest and the simplest one to install.
[http://fsinfo.cs.uni-sb.de/~abe/wApua/]

Tofoa

This is a Wap emulator written in Python. I don‘t use it as its installation was quite painful for me (I mean, really painful compared to some other softwares which were not easy to install), and it produces strange results, even with valid WML files.
[http://tofoa.free-system.com/]

Deck-It

This Wap emulator is available for Windows and Linux/Intel only. Too bad, because I can‘t use it for my tests. Another bad point is that it can‘t read local WML files. Apart that, it‘s could be the best software in this section if it was available on more platforms.
[ ]

WinWAP

This is a commercial Wap browser, available for Windows only.
[http://www.winwap.org/]

WML Browser

Its name tells what it is. In practice, I‘m waiting for a message other than "segmentation fault"...
[http://www.wmlbrowser.org/]

ACKNOWLEDGEMENTS

Werner Heuser - for his numerous ideas, advices and his help for the debugging

AUTHOR

Sébastien Aperghis-Tramoni <maddingue@free.fr>

COPYRIGHT