HtmlParse—An HTML preprocessor

Chris Rutter

April 10, 1997
Version 1.10

Contents

1 Introduction

HtmlParse is a utility designed to save time managing a web site. It provides useful utilities for dealing with page headers, footers and most importantly, links between pages. The idea of it is to prevent duplication of material which may later need changing throught the entire site. Before using HtmlParse, you must setup a *script* (see section 3) which HtmlParse uses to process your website. Each time you want to process the web site, you rerun the HtmlParse program which reads in the options from the script, and processes the files into a separate output directory.

1.1 The structure of each page

Each page that HtmlParse will process starts with the *preamble*. The preamble contains a series of directives describing the page (e.g. what footer it should have and what reference number to give the page). As soon as HtmlParse encounters a line that does not start with `#`, it considers the preamble over, and the document body starts. The document body can contain directives, also. The final output page consists of five sections: the preamble file, the title, the header, the body (including any text inserted by in-line directives) and the footer.

1.2 How to invoke HtmlParse

The HtmlParse program itself (i.e. `!HtmlParse.!RunImage` on RISC OS machines) can be invoked simply by running it with parameters

`HtmlParse` <*script file*>

However, under RISC OS HtmlParse has been setup to save time. As soon as HtmlParse has been 'seen', all files of type `0x109` (or `HPScript`) will invoke HtmlParse automatically when run.

1.3 The recommended script directory structure

In the script file you have to supply HtmlParse with the paths of several directories containing various bits and pieces that HtmlParse uses when processing the pages. It is more convienient if all these directories are subdirectories of a central 'web site' directory. Therefore, the recommended directory structure layout is shown in figure 1. The script file should be the actual script file which you want HtmlParse to look at (see section 3 for more information on script files). It is suggested that you create an obey file called 'RunScript' (or a similar name) containing the following:

```
Set WWWRoot$Path <Obey$Dir>.
Run WWWRoot:Script
```

which will (providing HtmlParse has been seen) start up HtmlParse in a taskwindow, working on your script. Then, whenever you need to specify the name of a directory in the script, you can simply use `WWWRoot:`<*directory name*>.

Figure 1: Recommended directory structure

1.4 Directives

Directives can be placed on each page that HtmlParse will process. There are two different types of directives: *in-line* and *preamble*. *In-line* directives can be placed anywhere on the page, so long as they are at the start of a line. Typically, in-line directives insert a bit of HTML at the point where they are placed on the page. An example would be the `#links` directive, which inserts a set of links to other pages wherever you place the command. An example of a preamble directive is the `#page` directive, which gives the page a reference number. HtmlParse considers it has found the end of the directive when it reaches either a `;` character or a newline character. For example:

```
...some text</p>
#links pr,main
<p>The main point...
```

HtmlParse will process in-line directives found in the preamble as though they were placed at the top of the document body. If you want to insert a `#` character itself at the beginning of a line, use the string `##`, otherwise just use `#` in the middle of a line – directives can only be placed at the beginning of a line.

1.5 How HtmlParse scans the input directory

HtmlParse processes the input directory in two stages: first, it goes round all the pages copying each page's preamble into memory, but ignoring the document body, so that it can form a hierarchy of pages internally. Then, it goes round again, processing the document body of each page.

2 Features

2.1 The $ character

At any point in the script (apart from when specifying ADFS pathnames), the $ character can be used to mean 'the root of the output directory'. As an example, if you placed a $ character in an HTML file two directories down from the root input directory, it would be replaced with `../../` (meaning go up two directories). This way, you can always specify a path relative to the root directory without worrying how many levels deep the file is, as HTMLParse will automatically reprocess the $ character if the directory moves. If you want to insert a dollar character into the file, you must use the sequence `$$`. If the $ character is used in either a link definition or the control directory specifier, then needless `../` sequences will be avoided. For example, if a link was defined as `$progs/off/g.html` and it was referenced from directory `/progs/off` then only `g.html` would be inserted into the output, not `../../progs/off/g.html` as in other cases. Full support of this kind for all uses of the $ character will be a forthcoming improvment in HTMLParse.

2.2 Links

Typically, at some place on every page, you will want some kind of standard graphic(s)/text that links to the pages around and above. For example, a page two levels deep describing the features of a program might link to the central program page above, and also to the main page two levels above. Obviously, if you change the name or position one of these pages that several other pages link to, it will mean a time-consuming search-and-replace throughout all the pages in your web site to make all the pages point to the new location. Equally, if you wanted to change the format in which the links where presented (e.g. from text into images, or a different layout of text) the job would involve manually going round every single page that you wanted to change (*very* time consuming). HTMLParse includes provision for defining a series of commonly-used links in the script file (see section 3.2 to find out how). On the page, you can use the in-line directive `#links` to include a set of those links, formatted according to the links format file.

2.2.1 Link format files

In the script, you specify a directory that contains all the link format files. (Each link will use the format file called "default" if no other is specified.) For each link format, you need to define

- the beginning string of HTML that goes before any of the links
- the format of a linked-to item
- the string of HTML that goes between multiple links
- the closing string of HTML that goes at the end of all the links.

Each link that you define in the script file has two mandatory properties and two optional ones. The properties are

Path — This is the destination URL of the page that the link links to

Textual description — This is the textual description of the link

Image file — This is the image file to use for the link (*optional*).

ALT tag contents — This is the text to put in the `ALT=` tag inside the image tag. It is necessary when you are using image links, because when the page is viewed on a text-only system the image is not displayed. In most cases, you will omit this property of the link, and simply use the *textual description* instead for the `ALT=` tag contents.

When specifying your format string, you can use the four control sequences (`%p`, `%t`, `%i` and `%a`) at any point. HTMLParse will replace the control sequence with either the path (`%p`), the textual description (`%t`), the image file path (`%i`) or the `ALT=` tag contents (`%a`).

2.2.2 The `#links` directive

The `#links` directive is an in-line directive. Its format is

> `#links` <*comma-separated links list*> [*optional alternative format file*]

The links in the comma-separated links list must all be defined in the links section of the script file (see section 3.2 for further details). An example directive would be

```
#links progs,main program
```

which would insert links to the 'progs' and 'main' pages, using the format file 'program'.

2.2.3 Examples of link setups

Here are some examples of various format setups:

1. *If you want to use simple text-based links ...*
   In this example the link *main* links to `$index.html` and is called 'main page' and the link *progs* links to `$info/progs.html` and is called 'programs page'. If your links format file was like this:

   ```
   Click to go up to the
   <a href="%p">%t</a>
   ,␣
   .
   ```

   then the instruction to include links *main* and *progs* would produce the following output:

   ```
   Click to go up to the <a href="$info/progs.html">prog-
   rams page</a>, <a href="$index.html">main page</a>.
   ```

2. *Another variant of text-based links ...*
   In this example, you would want to use the name of the page first, and the link second.

```
Links:
[%t -- <a href="%p">here</a>]
,␣
blank line
```

   would produce the following output (with the links from the previous example):

```
Links:  [programs page -- <a href="$info/progs.html"-
>here</a>], [main page -- <a href="$index.html">here-
</a>]
```

   or as far as the final formatted HTML is concerned, it would look like this:

   Links: [programs page – *here*], [main page – *here*]

3. *An example with images ...*
   In this example, link *main* links to `$index.html`, its textual description is 'main page' and the image property is set to `$images/links/main.gif`.

```
[
<a␣href="%p"><img␣src="%i"␣alt="%t"></a>
,␣
]
```

   when told to produce a link to *main* produces the following output:

```
[<a href="$index.html"><img src="$images/links/main.-
gif" alt="main page"></a>]
```

2.3 Controls

The popular LaTeX2Html package splits a LaTeX file up into several HTML pages, each page with a row of 'video-control'[1] style buttons which link to different sections. This behaviour can be duplicated with HtmlParse. First, each page must be given a reference number (if a page is *not* given a reference number then it is not included in the hierarchy of pages that the control buttons use[2]). Then, HtmlParse will automatically link together all pages given a reference number, placing the controls at a user-defined place on each page.

[1] By 'video-control' I mean back, forward, etc.
[2] Hereafter referred to as the 'controls hierarchy'.

2.3.1 Reference number scheme

All pages are given an maximum eight-digit long reference number. Each digit can range from 0–9, A–Z (A following 9). The main page is defined as having reference number 0. Each page can have 35 other pages on the same 'level' as itself. When the digits are read from left-to-right, the digits leading up to the final digit indicate which pages are 'above' the page in the hierarchy. For example, page 1 would be a page on the same level as the main page, probably linked to via the main page. Page 04 would be a page linked to from the main page, where the main page would be one level above it. Similarly, 0B would be a page linked to from the main page, where the main page would also be one level above it. Page 0B4 would be a page linked to from page 0B, where page 0B would be one level above and the main page two levels above.

Figure 2: An example hierarchy of pages

2.3.2 Page-linking method

So, how does HTMLParse actually link the pages together? Well, it puts four control buttons at the specified place on the page: *previous*, *next*, *back* and *main*. The rules for each control button on any page are as follows (HTMLParse tries out each rule in the order they are listed here, until it gets a rule that produces a page that is different to the current page and is valid):

Next
1. Pick the page one level down the hierarchy from this page with the smallest last digit
2. Pick the page with the next highest last digit on the same level and branch[3]
3. Go up the hierarchy of pages that lead to this one, until a level is found where rule 2 applies
4. Discard the next link on this page.

Previous
1. Locate a page that has the *next* pointer set to this page
2. Discard the previous link on this page.

Back
1. Pick the page one level up the hierarchy
2. Discard the back link on this page.

Main
1. Link to page 0
2. Discard the main link on this page.

Figure 3: An example showing the linking together of the pages from the previous example

So for our previous diagram the pages would be linked together as in figure 3.

The image used to draw each icon is found in a directory you specify as a subdirectory of the web site[4]. The image files that need to be present are:

`next.gif` This is used for an active 'next' link

`nextg.gif` This is used for an inactive 'next' link (i.e. where HTMLParse couldn't find a rule which produced a next page different to itself).

`prev.gif` This is used for an active 'previous' link

`prevg.gif` This is used for an inactive 'previous' link

`back.gif` This is used for an active 'back' link

`backg.gif` This is used for an inactive 'back' link

`main.gif` This is used for an active link to the main page

`maing.gif` This is used for an inactive link to the main page.

Also, there should be a text file (specified in the script) which contains eight lines, each containing the `ALT=` tag contents for each image (in the same order as in the above list).

2.3.3 The `#controls` directive

The `#controls` directive is an in-line directive. Its format is

`#controls` [*optional format string*]

By default, the controls are placed in the order *previous*, *next*, *back*, *main*. However, you are not always going to want the links in this order. Therefore, you can specify a *format string* after the `#controls` directive. The format string is composed of the initials of the controls you want to include, in the order they should appear. For example

```
#controls PBN
```

[3] i.e. identical first $(n-1)$ characters where n is the length of the string.

[4] You should specify *this* directory in URL format, e.g. `$images/misc/controls/`.

would include the controls *previous*, *back* and *next*, in that order. However, sometimes, you would prefer an inactive link icon (see above) *not* to appear if a suitable page cannot be found. Any link which should simply disappear instead of placing an inactive icon should be specified with a lower-case letter. This is useful on the main page, for example, where it would be preferable to have the 'main page' icon simply disappear rather than display greyed out. An example of this would be

```
#controls PbNm
```

where if a suitable *back* or *main* link is not found, the icon is deleted. However, if a suitable *previous* link is not found, an inactive icon is placed in its place instead. You can specify your default controls configuration by using the `DefaultControls` script element (see section 3.3 for more information).

2.4 Headers, footers and titles

HTMLParse has the option to insert a header and footer onto every page. If no other header or footer is chosen on the page itself, the header and footer 'default' is chosen, but the header used on an individual page can be changed by using the page directives (see later). The header comes before everything on the page apart from the title, and the footer comes after the page body (see section 1.1 for more information on the page structure). Note that the header file should not contain the opening `<html>`, `<head>` or `<title>` tags, as those are already inserted before the header, by the `#title` directive. The beginning of the header file is inserted after the closing `</title>` tag, with the `<head>` element open – it is up to you to close it, using `</head>`. This allows you to insert `META` information into the header. So, the first HTML tag in the header file should be `</head>`. An example header file would be:

```
</head> <!-- Page designed by Chris Rutter -->
<body><h1 align=left><img src="$images/logos.gif"></h1>
```

The footer file is appened after the links. In this example, an opening `<p>` tag has been used in the links, and so the footer must use a closing `</p>` tag. Note that the header and footer *can* contain in-line directives (e.g. `#controls`) which would be useful if you want the header and footer at a standard place on every page.

```
<hr>
<address><a href="$about.html">Sibelius Software</a>
<a href="mailto:info@sibelius.demon.co.uk">
<img src="$images/post.gif" alt="[mail]" border=0>
</a></address></p></body></html>
```

2.4.1 The `#title` directive

The `#title` directive is a preamble directive. Its format is simply `#title` <*title string*>. It inserts the string

`<title>` <*title string*> `</title>`

after the end of the preamble file on the output page.

2.4.2 The `#header` directive

The `#header` directive is a preamble directive. Its format is `#header` <*header file*>. It inserts the specified header file, processing as it were part of the file it being attached to. If no `#header` directive is specified on a page, HTMLParse inserts the header file called 'default'. The file is inserted *after* the title. The header can contain all normal in-line directives (e.g. `#controls`) which is useful if you want standard links (e.g.) at a standard place on every page.

2.4.3 The `#footer` directive

The `#footer` directive is a preamble directive. Its format is `#footer` <*footer file*>. It inserts the specified footer file, processing as it were part of the file it being attached to . If no `#footer` directive is specified on a page, HTMLParse inserts the footer file called 'default'. The file is inserted at the very end of the output page. The file is inserted *after* the title. The footer can contain all normal in-line directives (e.g. `#links`) which is useful if you want standard links (e.g.) at a standard place on every page.

2.4.4 The `#scaffold` directive

Putting the preamble directive `#scaffold` <*file*> in an input file has the same effect as putting two separate `#header` <*file*> and `#footer` <*file*> directives (i.e. it is a command simply to save on typing).

2.5 The `#noscaffold` directve

If you want to prevent a header and footer from being added, put `#noscaffold` in the preamble to a page.

3 Scripts

HTMLParse infers all its setup from the script it is supplied. The script is simply a human-readable textfile, with several lines setting various options. The script parser will ignore blank lines and lines starting with a '`#`' character. The script should contain various sections[5], each started by having a line blank except for `[section name]`. The basis of every HTMLParse command is '*variable=contents*', where *variable* might be something such as '`Source`' (the source directory from which to read the files) and *contents* might be '`<WebRoot$Dir>.Files.Input`'[6].

3.1 The Setup section

The setup section is present to tell HTMLParse where to find the input directory and where to put the files once they are processed and what access permission to give them. The variables that must be defined are:

[5] Script commands have different effects depending on their section.

[6] Note that all RISC OS pathnames are parsed via *OS_GSTrans* and therefore can contain system variables and paths.

```
# Our example HtmlParse script
# Version 1.00

[Setup]
Source=ADFS::4.$.Foo.Bar
Destination=WWWRoot:<Output$Dir>.pages
Access=wr/WR

...
```

Figure 4: An example fragment from a setup script

Source The input directory (containing all the unprocessed files) should be specified as `Source=`<*input directory*>

Destination The destination directory (which once the files have been processed contains every file in the input directory except in a processed state) should be specified as `Destination=`<*output directory*>

Access The access permission set on all the processed files should be of the standard Acorn form (e.g. `wr/WR` would set public read and write access). It should be specified as `Access=`<*access permission*>

3.2 The Links section

The format to define a link is *linkname=path,textual description*[,*image file path,ALT tag contents*]. An example would be `main=$index.html,main page`, or perhaps `progs=$progs/index.html,programs page,$images/links/progs.gif` if you were using image links. Note the use of the $ character (section 2.1). The format at which links are placed at the bottom of the can be defined in the links directory (see section 3.3).

3.3 The Scaffold section

The Scaffold section is present to tell HTMLParse where to find certain commonly-used resources. The variables to define are:

Headers The contents of this should be the base directory in which all the header files can be found.

Footers The contents of this should be the base directory in which all the footer files can be found.

Links The contents of this should be the base directory in which all the link format files can be found.

Preamble The contents of this should be a text file which contain data to put at *very* beginning of a file. Typical contents would be

```
<!doctype html public "-//W3C//DTD HTML 3.2//EN">
<html><head>
```

but the only requirement is that you have an opening `<html>` and `<head>` tag in this file.

Controls The contents of this should be the base directory in which all the controls images can be found. (This is taken to be a URL-type pathname, e.g. `$images/misc/controls/`.)

ControlsAlt This should be the (RISC OS) path of a file which contains the `ALT=` tag text for the control images (see section 2.3.2 for more information).

DefaultControls DefaultControls sets the default setting for the controls format (see section 2.3.3 for information on the format of this string).

4 Quick directive reference

4.1 Preamble directives

Command	Format	Decscription
`#header`	`#header` <*file*>	Use a particular header
`#footer`	`#footer` <*file*>	Use a particular footer
`#scaffold`	`#scaffold` <*file*>	Use a header and footer
`#title`	`#title` <*title*>	Specify the page title
`#page`	`#page` <*ref no*>	Include a page in the controls hierarchy
`#noscaffold`	`#noscaffold`	Prevent addition of header, footer or links

4.2 In-line directives

Command	Format	Decscription
`#links`	`#links` <*linknames*> [*format file*]	Insert links
`#controls`	`#controls` <*format*>	Insert control buttons

5 Future improvements

- Automatically include *width=*, *height=* tags to improve page-loading times (and disimprove the look of the page on certain browsers)
- Support for `#include` directive
- Create user-defined macros
- Date-stamp mechanism so only changed files are reprocessed
- WIMP filter that traps loading of HTML (`0xfaf`) files and reprocesses that file only, and then passes the file on to the loading application (e.g. HTMLEdit)

6 History

1.02beta–1.05beta
- Made HTMLParse much less likely to trash confusing setups involving "!HtmlParse" files
- Restructured HTMLParse internally to make it nicer to maintain

	– Fixed incorrect usage of `fgetc()` and `fgets()`
1.05beta–1.06beta	– Fixed bug where HTMLParse got confused with directory pointers when scanning directories containing "!HtmlParse" files (previously would have most often deleted all directories after the first)
	– If HTMLParse detects a too long command name it will stop reading in and cause an error
	– Turned source tree into multiple-file arrangement where all are compiled separately but linked in one pass (saves time)
	– Added limits checking for everything that needed limits checking
1.06beta–1.07beta	– Ensured that empty lines in links format files are read in as blanks, not as junk
	– Set type of resulting files to HTML (`0xfaf`) instead of text
	– Added $ charcter collapsing for link statements
1.07beta–1.08beta	– Tidied up the `pages_scan()` routines
	– Fixed a rather nasty bug which caused heap overwriting
	– Fixed a bug which got the levels of files wrong when binary files were present
	– Fixed a bug which would have prevented one worded header commands from being recognised (e.g. `#noscaffold`)
	– Restructed output directory creating system, so that binary directories do not have to be in the top-level directory
1.08beta–1.09beta	– Marginally tidied up a few loose statements
	– Fixed a tedious bug which churned out rubbish into links directives
1.09beta–1.10	– Shunted source files around a lot in my new format
	– Moved lots of functions about
	– Tidied memory structure slightly
	– Fixed a few bugs (uninitialised local variables)
	– Added `Preamble=` scaffold field, and stopped closing `<head>` tag after title

7 Contacting the author

The author may be contacted

- *by snail mail ...*
  Old Laceys, St John's Street, Duxford, Cambridge, CB2 4RA.
- *by email ...* as chris@fluff.org
- *by phone ...* on 01223 832474
- *by ArcadeMail ...* as Chris Rutter