Clean up your Web pages with HTML TIDY

The maintenance of Tidy has now been taken over by a group of enthusiastic volunteers at Source Forge, see http://tidy.sourceforge.net. There you can find the source code and binaries for a very wide range of platforms. A new version of Tidy is nearing completion which encapsulates Tidy as a library *TidyLib*, and has been designed for easy integration with other software

Introduction to TIDY

When editing HTML it's easy to make mistakes. Wouldn't it be nice if there was a simple way to fix these mistakes automatically and tidy up sloppy editing into nicely layed out markup? Well now there is! Dave Raggett's HTML TIDY is a free utility for doing just that. It also works great on the atrociously hard to read markup generated by specialized HTML editors and conversion tools, and can help you identify where you need to pay further attention on making your pages more accessible to people with disabilities.

Tidy is able to fix up a wide range of problems and to bring to your attention things that you need to work on yourself. Each item found is listed with the line number and column so that you can see where the problem lies in your markup. Tidy won't generate a cleaned up version when there are problems that it can't be sure of how to handle. These are logged as "errors" rather than "warnings".

Dave Raggett has now passed the baton for maintaining Tidy to a group of volunteers working together as part of the open source community at Source Forge. The source code continues to be available under an open source license, and you are encouraged to pass on bug reports and enhancement requests at http://tidy.sourceforge.net.

If you find HTML Tidy useful and you would like to say thanks, then please send me a (paper) postcard or other souvenir from the area in which you live along with a few words on what you are using Tidy for. It will be fun to map out where Tidy users are to be found! My postal address is given at the end of this file.

The W3C public email list devoted to HTML Tidy is: <html-tidy@w3.org>. To subscribe send an email to html-tidy-request@w3.org with the word subscribe in the subject line (include the word unsubscribe if you want to unsubscribe). The archive for this list is accessible online. If you would like to contact the developers, or you just want to submit an enhancement request or a bug report, please visit http://tidy.sourceforge.net.

Tidy can now perform wonders on HTML saved from Microsoft Word 2000! Word bulks out HTML files with stuff for round-tripping presentation between HTML and Word. If you are more concerned about using HTML on the Web, check out Tidy's "Word-2000" config option! Of course Tidy does a good job on Word'97 files as well!

Tidy features in an article by Scott Nesbitt on webreview.com, and more recently on Dave Central's Best of Linux, and as tool of the month on Unix Review by Joe Brockmeier, who writes:

> *"One thing I love about the UNIX philosophy is the idea that each program should do one job and do it really well. There are zillions of small tools for UNIX-type OSes that make life much easier and are hugely useful, but they don't necessarily get written about. They certainly don't receive the same kind of coverage that Apache and Sendmail receive. One of my favorites, HTML Tidy, is a tool for HTML/Web development that I think will interest a lot of folks. HTML Tidy cleans up HTML produced by WYSIWYG editors and such."*

Tidy is available as a downloadable binary, as source code (ANSI C), or as an online service at W3C, Info Network, HTML Help's site Valet and other sites.

Tutorials for HTML and CSS

If you are just starting off and would like to know more about how to author Web pages, you may find my guide to HTML and CSS helpful. Please send me feedback on this, and I will do my best to further improve it.

Examples of TIDY at work

Tidy corrects the markup in a way that matches where possible the observed rendering in popular browsers from Netscape and Microsoft. Here are just a few examples of how TIDY perfects your HTML for you:

- **Missing or mismatched end tags are detected and corrected**

  ```
  <h1>heading
  <h2>subheading</h3>
  ```
  is mapped to
  ```
  <h1>heading</h1>
  <h2>subheading</h2>
  ```
- **End tags in the wrong order are corrected:**

  ```
  <p>here is a para <b>bold <i>bold italic</b> bold?</i> normal?
  ```
  is mapped to
  ```
  <p>here is a para <b>bold <i>bold italic</i> bold?</b> normal?
  ```
- **Fixes problems with heading emphasis**

  ```
  <h1><i>italic heading</h1>
  <p>new paragraph
  ```
  In Netscape and Internet Explorer this causes everything following the heading to be in the heading font size, not the desired effect at all!

  Tidy maps the example to
  ```
  <h1><i>italic heading</i></h1>
  <p>new paragraph
  ```
- **Recovers from mixed up tags**

  ```
  <i><h1>heading</h1></i>
  <p>new paragraph <b>bold text
  <p>some more bold text
  ```
  Tidy maps this to
  ```
  <h1><i>heading</i></h1>
  <p>new paragraph <b>bold text</b>
  <p><b>some more bold text</b>
  ```
- **Getting the <hr> in the right place:**

  ```
  <h1><hr>heading</h1>
  <h2>sub<hr>heading</h2>
  ```
  Tidy maps this to
  ```
  <hr>
  <h1>heading</h1>
  <h2>sub</h2>
  <hr>
  <h2>heading</h2>
  ```
- **Adding the missing "/" in end tags for anchors:**

  ```
  <a href="#refs">References<a>
  ```
  Tidy maps this to
  ```
  <a href="#refs">References</a>
  ```
- **Perfecting lists by putting in tags missed out:**

  ```
  <body>
  <li>1st list item
  <li>2nd list item
  ```
  is mapped to
  ```
  <body>
  <ul>
  <li>1st list item</li>
  <li>2nd list item</li>
  </ul>
  ```
- **Missing quotes around attribute values are added**

  Tidy inserts quote marks around all attribute values for you. It can also detect when you have forgotten the closing quote mark, although this is something you will have to fix yourself.
- **Unknown/Proprietary attributes are reported**

Tidy has a comprehensive knowledge of the attributes defined in the HTML 4.0 recommendation from W3C. This often allows you to spot where you have mistyped an attribute or value.

- **Proprietary elements are recognized and reported as such.**
  Tidy will even work out which version of HTML you are using and insert the appropriate DOCTYPE element, as per the W3C recommendations.
- **Tags lacking a terminating '>' are spotted**
  This is something you then have to fix yourself as Tidy is unsure of where the > should be inserted.

Layout style

You can choose which style you want Tidy to use when it generates the cleaned up markup: for instance whether you like elements to indent their contents or not. Several people have asked if Tidy could preserve the original layout. I am sorry to say that this would be very hard to support due to the way Tidy is implemented. Tidy starts by building a clean parse tree from the source file. The parse tree doesn't contain any information about the original layout. Tidy then pretty prints the parse tree using the current layout options. Trying to preserve the original layout would interact badly with the repair operations needed to build a clean parse tree and considerably complicate the code.

Some browsers can screw up the right alignment of text depending on how you layout headings. As an example, consider:

```
<h1 align="right">
  Heading
</h1>
<h1 align="right">Heading</h1>
```

Both of these should be rendered the same. Sadly a common browser bug fails to trim trailing whitespace and misaligns the first heading. HTML Tidy will protect you from this bug, except when you set the indent option to "yes".

Setting the indent option to yes can also cause problems with table layout for some browsers:

```
<td><img src="foo.gif"></td>
<td><img src="foo.gif"></td>
```

will look slightly different from:

```
<td>
  <img src="foo.gif">
</td>
<td>
  <img src="foo.gif">
</td>
```

You can avoid such quirks by using indent:Âno or indent:Âauto in the config file.

Internationalization issues

Tidy offers you a choice of character encodings: US ASCII, ISO Latin-1, UTF-8 and the ISO 2022 family of 7 bit encodings. The full set of HTML 4.0 entities are defined. Cleaned up output uses HTML entity names for characters when appropriate. Otherwise characters outside the normal range are output as numeric character entities. Tidy defaults to assuming you want the output to be in US ASCII. Tidy doesn't yet recognize the use of the HTML meta element for specifying the character encoding.

Accessibility

Tidy offers advice on accessibility problems for people using non-graphical browsers. The most common thing you will see is the suggestion you add a summary attribute to table elements. The idea is to provide a summary of the table's role and structure suitable for use with aural browsers.

Cleaning up presentational markup

Many tools generate HTML with an excess of FONT, NOBR and CENTER tags. Tidy's *-clean* option will replace them by style properties and rules using CSS. This makes the markup easier to read and maintain as well as reducing the file size! Tidy is expected to get smarter at this in the future.

Some pages rely on the presentation effects of isolated <p> or </p> tags.Tidy deletes empty paragraph and heading elements etc. The use of empty paragraph elements is not recommended for adding vertical whitespace. Instead use style sheets, or the <br> element. Tidy won't discard paragraphs only containing a nonbreaking space &nbsp;

Teaching Tidy about new tags!

You can teach Tidy about new tags by declaring them in the configuration file, the syntax is:

```
new-inline-tags: tag1, tag2, tag3
new-empty-tags: tag1, tag2, tag3
new-blocklevel-tags: tag1, tag2, tag3
new-pre-tags: tag1, tag2, tag3
```

The same tag can be defined as empty and as inline or as empty and as block.

These declarations can be combined to define an a new empty inline or empty block element, but you are not advised to declare tags as being both inline and block!

Note that the new tags can only appear where Tidy expects inline or block-level tags respectively. This means you can't (yet) place new tags within the document head or other contexts with restricted content models. So far the most popular use of this feature is to allow Tidy to be applied to Cold Fusion files.

I am working on ways to make it easy to customize the permitted document syntax using assertion grammars, and hope to apply this to a much smarter version of Tidy for release later this year or early next year.

Limited support for ASP, JSTE and PHP

Tidy is somewhat aware of the preprocessing language called ASP which uses a pseudo element syntax <% ... %> to include preprocessor directives. ASP is normally interpreted by the web server before delivery to the browser. JSTE shares the same syntax, but sometimes also uses <# ... #>. Tidy can also cope with another such language called PHP, which uses the syntax <?php ... ?>

Tidy will cope with ASP, JSTE and PHP pseudo elements within element content and as replacements for attributes, for example:

```
<option <% if rsSchool.Fields("ID").Value
  = session("sessSchoolID")
  then Response.Write("selected") %>
  value='<%=rsSchool.Fields("ID").Value%>'>
  <%=rsSchool.Fields("Name").Value%>
  (<%=rsSchool.Fields("ID").Value%>)
</option>
```

Note that Tidy doesn't understand the scripting language used within pseudo elements and attributes, and can easily get confused. Tidy may report missing attributes when these are hidden within preprocessor code. Tidy can also get things wrong if the code includes quote marks, e.g. if the example above is changed to:

```
value="<%=rsSchool.Fields("ID").Value%>"
```

Tidy will now see the quote mark preceding ID as ending the attribute value, and proceed to complain about what follows. Note you can choose whether to allow line wrapping on spaces within pseudo elements or not using the `wrap-asp` option. If you used ASP, JSTE or PHP to create a start tag, but placed the end tag explicitly in the markup, Tidy won't be able to match them up, and will delete the end tag for you. So in this case you are advise to make the start tag explicit and to use ASP, JSTE or PHP for just the attributes, e.g.

```
<a href="<%=random.site()%>">do you feel lucky?</a>
```

Tidy allows you to control whether line wrapping is enabled for ASP, JSTE and PHP instructions, see the wrap-asp, wrap-jste and wrap-php config options, respectively.

I regret that Tidy does **not** support Tango preprocessing instructions which look like:

```
<@if variable_1='a'>
  do something
<@else>
  do nothing
</@if>
<@include <@cgi><@appfilepath>includes/message.html>
```

Tidy supports another preprocessing syntax called "Tango", but only for attribute values. Adding support for pseudo elements written in Tango looks as if it would be quite tough, so I would like to gauge the level of interest before committing to this work.

Limited support for XML

XML processors compliant with W3C's XML 1.0 recommendation are very picky about which files they will accept. Tidy can help you to fix errors that cause your XML files to be rejected. Tidy doesn't yet recognize all XML features though, e.g. it doesn't understand CDATA sections or DTD subsets.

Creating Slides

The *-slides* option allows you to burst a single HTML file into a number of linked slides. Each H2 element in the input file is treated as delimiting the start of the next slide. The slides are named slide1.html, slide2.html, slide3.html etc. This is a relatively new feature and ideas are welcomed as to how to improve it. In particular, I plan to add support to the configuration file for setting the style sheet for slides and for customizing the slides via a template.

I would be interested in hearing from anyone who can offer help with using JavaScript for adding dynamic effects to slides, for instance similar to those available in Microsoft PowerPoint.

Indenting text for a better layout

Indenting the content of elements makes the markup easier to read. Tidy can do this for all elements or just for those where it's needed. The auto-indent mode has been used below to avoid indenting the content of title, p and li elements:

```
<html>
  <head>
    <title>Test document</title>
  </head>
  <body>
    <p>para which has enough text to cause a line break,
    and so test the wrapping mechanism for long lines.</p>
<pre>
This is
<em>genuine
      preformatted</em>
   text
</pre>
    <ul>
      <li>1st list item</li>
      <li>2nd list item</li>
    </ul>
    <!-- end comment -->
  </body>
</html>
```

Indenting the content does increase the size of the file, so you may prefer Tidy's default style:

```
<html>
<head>
<title>Test document</title>
```

```
</head>
<body>
<p>para which has enough text to cause a line break,
and so test the wrapping mechanism for long lines.</p>
<pre>This is
<em>genuine
      preformatted</em>
   text
</pre>
<ul>
<li>1st list item </li>
<li>2nd list item</li>
</ul>
<!-- end comment -->
</body>
</html>
```

How to run tidy

```
tidy [[options] filename]*
```

HTML tidy is not (yet) a Windows program. If you run tidy without any arguments, it will just sit there waiting to read markup on the stdin stream. Tidy's input and output default to stdin and stdout respectively. Errors are written to stderr but can be redirected to a file with the -f *filename* option.

I generally use the -m option to get tidy to update the original file, and if the file is particularly bad I also use the -f option to write the errors to a file to make it easier to review them. Tidy supports a small set of character encoding options. The default is ASCII, which makes it easy to edit markup in regular text editors.

For instance:

```
tidy -f errs.txt -m index.html
```

which runs tidy on the file "index.html" updating it in place and writing the error messages to the file "errs.txt". Its a good idea to save your work before tidying it, as with all complex software, tidy may have bugs. If you find any please let me know!

Thanks to Jacek Niedziela, The Win32 executable for tidy is now able to use wild cards in filenames. This utilizes the setargv library supplied with VC++.

Tidy writes errors to stderr, and won't be paused by the more command. A work around is to redirect stderr to stdout as follows. This works on Unix and Windows NT, but not on other platforms. My thanks to Markus Wolf for this tip!

```
tidy file.html 2>&1 | more
```

Tidy's Options

To get a list of available options use:

```
tidy -help
```

You may want to run it through more to view the help a page at a time.

```
tidy -help | more
```

Input and Output default to stdin/stdout respectively. Single letter options apart from -f may be combined as in: tidy -f errs.txt -imu foo.html

Matej Vela <vela@debian.org> has written a Unix man page for Tidy, but for the latest details on config options and for the release notes please visit this page: http://www.w3.org/People/Raggett/tidy.

Using a Configuration File

Tidy now supports a configuration file, and this is now much the most convenient way to configure Tidy. Assuming you have created a config file named "config.txt" (the name doesn't matter), you can instruct Tidy to use it via the command line option `-config config.txt`, e.g.

```
tidy -config config.txt file1.html file2.html
```

Alternatively, you can name the default config file via the environment variable named "HTML_TIDY". Note this should be the absolute path since you are likely to want to run Tidy in different directories. You can also set a config file at compile time by defining CONFIG_FILE as the path string, see platform.h.

You can now set config options on the command line by preceding the name of the option immediately (no intervening space) by "–", for example:

```
tidy --break-before-br true --show-warnings false
```

The following options are supported:

tidy-mark: *bool*

If set to *yes* (the default) Tidy will add a meta element to the document head to indicate that the document has been tidied. To suppress this, set tidy-mark to *no*. Tidy won't add a meta element if one is already present.

markup: *bool*

Determines whether Tidy generates a pretty printed version of the markup. Bool values are either *yes* or *no*. Note that Tidy won't generate a pretty printed version if it finds unknown tags, or missing trailing quotes on attribute values, or missing trailing '>' on tags. The default is *yes*.

wrap: *number*

Sets the right margin for line wrapping. Tidy tries to wrap lines so that they do not exceed this length. The default is 66. Set wrap to zero if you want to disable line wrapping.

wrap-attributes: *bool*

If set to *yes*, attribute values may be wrapped across lines for easier editing. The default is no. This option can be set independently of wrap-scriptlets

wrap-script-literals: *bool*

If set to *yes*, this allows lines to be wrapped within string literals that appear in script attributes. The default is *no*. The example shows how Tidy wraps a really really long script string literal inserting a backslash character before the linebreak:

```
<a href="somewhere.html" onmouseover="document.status = '...some \
really, really, really, really, really, really, really, really, \
really, really long string..';">test</a>
```

wrap-asp: *bool*

If set to *no*, this prevents lines from being wrapped within ASP pseudo elements, which look like: <%Â...Â%>. The default is *yes*.

wrap-jste: *bool*

If set to *no*, this prevents lines from being wrapped within JSTE pseudo elements, which look like: <#Â...Â#>. The default is *yes*.

wrap-php: *bool*

If set to *no*, this prevents lines from being wrapped within PHP pseudo elements. The default is *yes*.

literal-attributes: *bool*

If set to *yes*, this ensures that whitespace characters within attribute values are passed through unchanged. The default is *no*.

tab-size: *number*

Sets the number of columns between successive tab stops. The default is 4. It is used to map tabs to spaces when reading files. Tidy never outputs files with tabs.

indent: *no, yes* or *auto*

If set to *yes*, Tidy will indent block-level tags. The default is *no*. If set to *auto* Tidy will decide whether or not to indent the content of tags such as title, h1-h6, li, td, th, or p depending on whether or not the content includes a block-level element. You are advised to avoid setting indent to yes as this can expose layout bugs in some browsers.

indent-spaces: *number*

Sets the number of spaces to indent content when indentation is enabled. The default is 2 spaces.

indent-attributes: *bool*

If set to *yes*, each attribute will begin on a new line. The default is *no*.

hide-endtags: *bool*

If set to *yes*, optional end-tags will be omitted when generating the pretty printed markup. This option is ignored if you are outputting to XML. The default is *no*.

input-xml: *bool*

If set to *yes*, Tidy will use the XML parser rather than the error correcting HTML parser. The default is *no*.

output-xml: *bool*

If set to *yes*, Tidy will generate the pretty printed output writing it as well-formed XML. Any entities not defined in XML 1.0 will be written as numeric entities to allow them to be parsed by an XML parser. The tags and attributes will be in the case used in the input document, regardless of other options. The default is *no*.

add-xml-pi: *bool*
add-xml-decl: *bool*

If set to *yes*, Tidy will add the XML declatation when outputting XML or XHTML. The default is *no*. Note that if the input document includes an <?xml?> declaration then it will appear in the output independent of the value of this option.

output-xhtml: *bool*

If set to *yes*, Tidy will generate the pretty printed output writing it as extensible HTML. The default is *no*. This option causes Tidy to set the doctype and default namespace as appropriate to XHTML. If a doctype or namespace is given they will checked for consistency with the content of the document. In the case of an inconsistency, the corrected values will appear in the output. For XHTML, entities can be written as named or numeric entities according to the value of the "numeric-entities" property. The tags and attributes will be output in the case used in the input document, regardless of other options.

doctype: *omit, auto, strict, loose* or <*fpi*>

This property controls the doctype declaration generated by Tidy. If set to *omit* the output file won't contain a doctype declaration. If set to *auto* (the default) Tidy will use an educated guess based upon the contents of the document. If set to *strict*, Tidy will set the doctype to the strict DTD. If set to *loose*, the doctype is set to the loose (transitional) DTD. Alternatively, you can supply a string for the formal public identifier (fpi) for example:

```
doctype: "-//ACME//DTD HTML 3.14159//EN"
```

If you specify the fpi for an XHTML document, Tidy will set the system identifier to the empty string. Tidy leaves the document type for generic XML documents unchanged.

char-encoding: *raw, ascii, latin1, utf8* or *iso2022*

Determines how Tidy interprets character streams. For *ascii*, Tidy will accept Latin-1 character values, but will use entities for all characters whose value > 127. For *raw*, Tidy will output values above 127 without translating them into entities. For *latin1* characters above 255 will be written as entities. For *utf8*, Tidy assumes that both input and output is encoded as UTF-8. You can use *iso2022* for files encoded using the ISO2022 family of encodings e.g. ISO 2022-JP. The default is *ascii*.

numeric-entities: *bool*

Causes entities other than the basic XML 1.0 named entities to be written in the numeric rather than the named entity form. The default is *no*

quote-marks: *bool*

If set to *yes*, this causes " characters to be written out as &quot; as is preferred by some editing environments. The apostrophe character ' is written out as &#39; since many web browsers don't yet support &apos;. The default is *no*.

quote-nbsp: *bool*

If set to *yes*, this causes non-breaking space characters to be written out as entities, rather than as the Unicode character value 160 (decimal). The default is *yes*.

quote-ampersand: *bool*

If set to *yes*, this causes unadorned & characters to be written out as &amp;. The default is *yes*.

assume-xml-procins: *bool*

If set to *yes*, this changes the parsing of processing instructions to require ?> as the terminator rather than >. The default is *no*. This option is automatically set if the input is in XML.

fix-backslash: *bool*

If set to *yes*, this causes backslash characters "\" in URLs to be replaced by forward slashes "/". The default is *yes*.

break-before-br: *bool*

If set to *yes*, Tidy will output a line break before each <br> element. The default is *no*.

uppercase-tags: *bool*

Causes tag names to be output in upper case. The default is *no* resulting in lowercase, except for XML input where the original case is preserved.

uppercase-attributes: *bool*

If set to *yes* attribute names are output in upper case. The default is *no* resulting in lowercase, except for XML where the original case is preserved.

word-2000: *bool*

If set to *yes*, Tidy will go to great pains to strip out all the surplus stuff Microsoft Word 2000 inserts when you save Word documents as "Web pages". The default is *no*. Note that Tidy doesn't yet know what to do with VML markup from Word, but in future I hope to be able to map VML to SVG.

Microsoft has developed its own optional filter for exporting to HTML, and the 2.0 version is much improved. You can download the filter free from the Microsoft Office Update site.

clean: *bool*

If set to *yes*, causes Tidy to strip out surplus presentational tags and attributes replacing them by style rules and structural markup as appropriate. It works well on the html saved from Microsoft Office'97. The default is *no*.

logical-emphasis: *bool*

If set to *yes*, causes Tidy to replace any occurrence of i by em and any occurrence of b by strong. In both cases, the attributes are preserved unchanged. The default is *no*. This option can now be set independently of the clean and drop-font-tags options.

drop-empty-paras: *bool*

If set to *yes*, empty paragraphs will be discarded. If set to no, empty paragraphs are replaced by a pair of `br` elements as HTML4 precludes empty paragraphs. The default is *yes*.

drop-font-tags: *bool*

If set to *yes* together with the clean option (see above), Tidy will discard font and center tags rather than creating the corresponding style rules. The default is *no*.

enclose-text: *bool*

If set to *yes*, this causes Tidy to enclose any text it finds in the body element within a p element. This is useful when you want to take an existing html file and use it with a style sheet. Any text at the body level will screw up the margins, but wrap the text within a p element and all is well! The default is *no*.

enclose-block-text: *bool*

If set to *yes*, this causes Tidy to insert a p element to enclose any text it finds in any element that allows mixed content for HTML transitional but not HTML strict. The default is *no*.

fix-bad-comments: *bool*

If set to *yes*, this causes Tidy to replace unexpected hyphens with "=" characters when it comes across adjacent hyphens. The default is *yes*. This option is provided for users of Cold Fusion which uses the comment syntax: <!—Â—>

add-xml-space: *bool*

If set to *yes*, this causes Tidy to add xml:space="preserve" to elements such as pre, style and script when generating XML. This is needed if the whitespace in such elements is to be parsed appropriately without having access to the DTD. The default is *no*.

alt-text: *string*

This allows you to set the default alt text for img attributes. This feature is dangerous as it suppresses further accessibility warnings. **YOU ARE RESPONSIBLE FOR MAKING YOUR DOCUMENTS ACCESSIBLE TO PEOPLE WHO CAN'T SEE THE IMAGES!!!**

write-back: *bool*

If set to *yes*, Tidy will write back the tidied markup to the same file it read from. The default is *no*. You are advised to keep copies of important files before tidying them as on rare occasions the result may not always be what you expect.

keep-time: *bool*

If set to *yes*, Tidy won't alter the last modified time for files it writes back to. The default is *yes*. This allows you to tidy files without effecting which ones will be uploaded to the Web server when using a tool such as 'SiteCopy'. Note that this feature may not work on some platforms.

error-file: *filename*

Writes errors and warnings to the named file rather than to stderr.

show-warnings: *bool*

If set to *no*, warnings are suppressed. This can be useful when a few errors are hidden in a flurry of warnings. The default is *yes*.

quiet: *bool*

If set to *yes*, Tidy won't output the welcome message or the summary of the numbers of errors and warnings. The default is *no*.

gnu-emacs: *bool*

If set to *yes*, Tidy changes the format for reporting errors and warnings to a format that is more easily parsed by GNU Emacs. The default is *no*.

split: *bool*

If set to *yes* Tidy will use the input file to create a sequence of slides, splitting the markup prior to each successive <h2>. You can see an example of the results in a recent talk I made on XHTML. The slides are written to "slide1.html", "slide2.html" etc. The default is *no*.

new-empty-tags: *tag1, tag2, tag3*

Use this to declare new empty inline tags. The option takes a space or comma separated list of tag names. Unless you declare new tags, Tidy will refuse to generate a tidied file if the input includes previously unknown tags. Remember to also declare empty tags as either inline or blocklevel, see below.

new-inline-tags: *tag1, tag2, tag3*
Use this to declare new non-empty inline tags. The option takes a space or comma separated list of tag names. Unless you declare new tags, Tidy will refuse to generate a tidied file if the input includes previously unknown tags.

new-blocklevel-tags: *tag1, tag2, tag3*
Use this to declare new block-level tags. The option takes a space or comma separated list of tag names. Unless you declare new tags, Tidy will refuse to generate a tidied file if the input includes previously unknown tags. Note you can't change the content model for elements such as table, ul, ol and dl. This is explained in more detail in the release notes.

new-pre-tags: *tag1, tag2, tag3*
Use this to declare new tags that are to be processed in exactly the same way as HTML's pre element. The option takes a space or comma separated list of tag names. Unless you declare new tags, Tidy will refuse to generate a tidied file if the input includes previously unknown tags. Note you can't as yet add new CDATA elements (similar to script).

Sample Config File

This is just an example to get you started.

```
// sample config file for HTML tidy
indent: auto
indent-spaces: 2
wrap: 72
markup: yes
output-xml: no
input-xml: no
show-warnings: yes
numeric-entities: yes
quote-marks: yes
quote-nbsp: yes
quote-ampersand: no
break-before-br: no
uppercase-tags: no
uppercase-attributes: no
char-encoding: latin1
new-inline-tags: cfif, cfelse, math, mroot,
  mrow, mi, mn, mo, msqrt, mfrac, msubsup, munderover,
  munder, mover, mmultiscripts, msup, msub, mtext,
  mprescripts, mtable, mtr, mtd, mth
new-blocklevel-tags: cfoutput, cfquery
new-empty-tags: cfelse
```

Using Tidy from scripts

If you want to run Tidy from a Perl or other scripting language you may find it of value to inspect the result returned by Tidy when it exits: 0 if everything is fine, 1 if there were warnings and 2 if there were errors. This is an example using Perl:

```
if (close(TIDY) == 0) {
  my $exitcode = $? >> 8;
  if ($exitcode == 1) {
    printf STDERR "tidy issued warning messages\n";
  } elsif ($exitcode == 2) {
    printf STDERR "tidy issued error messages\n";
  } else {
    die "tidy exited with code: $exitcode\n";
  }
} else {
  printf STDERR "tidy detected no errors\n";
}
```

Source Code

The latest versions of the source code can be found at the Source Forge developer's site for Tidy, see http://tidy.sourceforge.net.

Acknowledgements

I would like to thank the many people who have written to me with suggestions for improvements or reporting bugs. Your help has been invaluable.

Jonathan Adair, Drew Adams, Osma Ahvenlampi, Carsten Allefeld, Richard Allsebrook, Jacob Sparre Andersen, Joe D'Andrea, Jerry Andrews, Bruce Aron, Takuya Asada, Edward Avis, Carlos Piqueres Ayela, Nick B, Chang Hyun Baek, Nick B, Denis Barbier, Chuck Baslock, Christer Bernerus, David J. Biesack, John Bigby, Yu Jian Bin, Alexander Biron, Keith Blakemore-Noble, Eric Blossom, Berend de Boer, Ochen M. Braun, Dave Bryan, David Brooke, Andy Brown, Keith B. Brown, Andreas Buchholz, Maurice Buxton, Jelks Cabaniss, John Cappelletti, Trevor Carden, Terry Cassidy, Mathew Cepl, Kendall Clark, Rob Clark, Jeremy Clulow, Dan Connolly, Larry Cousin, Ken Cox, Luis M. Cruz, John Cumming, Ian Davey, Keith Davies, Ciaran Deignan, David Duffy, Emma Duke-Williams, Tamminen Eero, Bodo Eing, Peter Enzerink, Baruch Even, David Fallon, Claus André Färber, Stephanie Foott, Darren Forcier, Martin Fouts, Frederik Fouvry, Rene Fritz, Stephen Fuqua, Martin Gallwey, Pete Gelbman, Francisco Guardiola, David Getchell, Michael Giroux, Davor Golek, Guus Goos, Léa Gris, Rainer Gutsche, Kai Hackemesser, Juha Häikiö, David Halliday, Johann-Christian Hanke, Vlad Harchev, Shane Harrelson, Andre Hinrichs, Bjoern Hoehrmann, G. Ken Holman, Bill Homer, Olaf Hopp, Craig Horman, Jack Horsfield, Nigel Horspool, Pao-Hsi Huang, Stuart Hungerford, Marc Jauvin, Rick Jelliffe, Peter Jeremy, Craig Johnson, Charles LaFountain, Steven Lobo, Zdenek Kabelac, Michael Kay, Jeffery Kendall, Axel Kielhorn, Konstantinos Kleisouris, Johannes Koch, Daniel Kohn, Rudy Kohut, Allan Kuchinsky, Volker Kuhlmann, Michael LaStella, Johnny Lee, Steve Lee, Tony Leneis, Nick Leverton, Todd Lewis, Dietmar Lippold, Gert-Jan C. Lokhorst, Murray Longmore, John Love-Jensen, Satwinder Mangat, Carole Mah, Anton Marsden, Bede McCall, Shane McCarron, Thomas McGuigan, Ian McKellar, Al Medeiros, Chris Nappin, Ann Navarro, Jacek Niedziela, Morten Blinksbjerg Nielsen, Kenichi Numata, Allan Odgaard, Matt Oshry, Gerald Oskoboiny, Paul Ossenbruggen, Ernst Paalvast, Christian Pantel, Dimitri Papadopoulos, Rick Parsons, Steven Pemberton, Daniel Persson, Lee Anne Phillips, Xavier Plantefeve, Karl Prinz, Andy Quick, Jany Quintard, Julian Reschke, Stephen Reynolds, Thomas Ribbrock, Ross L. Richardson, Philip Riebold, Erik Rossen, Dan Rudman, Peter Ruevski, Christian Ruetgers, Klaus Johannes Rusch, John Russell, Eric Schindler, J. Schlauch, Christian Schüler, Klaus Alexander Seistrup, Jim Seymour, Kazuyoshi Shimizu, Geoff Sinclair, Jo Smith, Paul Smith, Steve Spilker, Rafi Stern, Jacques Steyn, Michael J. Suzio, Zac Thompson, Eric Thorbjornsen, Oren Tirosh, John Tobler, Omri Traub, Loïc Trégan, Jason Tribbeck, Simon Trimmer, Steffen Ullrich, Stuart Updegrave, Charles A. Upsdell, Jussi Vestman, Larry W. Virden, Daniel Vogelheim, Nigel Wadsworth, Jez Wain, Randy Waki, Paul Ward, Neil Weber, Bertilo Wennergren, Yudong Yang, Jeff Young, Edward Zalta, Johannes Zellner, Christian Zuckschwerdt

Dave's Address

```
    35 Frome Road
    Bradford on Avon
    Wiltshire
    BA15 2EA
    United Kingdom
```

Dave Raggett <dsr@w3.org> is a consultant with Canon, and previously worked with Openwave, and before that with Hewlett Packard's UK Laboratories. Dave works on assignment to the World Wide Web Consortium, where he is the W3C lead for Voice and Multimodal.