Tkhtml 3.0 - HTML and CSS for Tcl/Tk

Dan Kennedy, Tcl Conference 2006.

Abstract

Tkhtml is a Tk widget written in C that displays documents formatted according to the HTML and CSS standards. This paper reports on the recent work on Tkhtml 3.0 sponsored by Eolas Technologies. The motivations for creating a standards compliant HTML/CSS widget for Tcl/Tk are examined, expected uses of the widget identified and the scope of functionality provided defined. Also discussed is the web browser hv3, developed as part of the Tkhtml project to test the HTML widget. Future extensions and enhancements to Tkhtml are contemplated.

Introduction To Tkhtml 3.0

Adding a widget to display documents formatted using the Hyper-Text Markup Language (HTML) [1] to Tk is not a new idea. There are currently at least three widely used widgets for just this purpose, Tkhtml 2 [2], htmllib [3] and the scrolledhtml widget included in the Iwidgets package [4]. The most common use is to embed hyper-linked documentation in an application ("help" documentation), although there exists at least one fully-functional web browser application built on top of Tkhtml 2, browsex [5]. Both htmllib and scrolledhtml are written in pure Tcl, making them particularly easy to deploy as part of Tcl applications. Tkhtml 2 is written in the C language, and has been used by applications written in Tcl, Ruby, Python and possibly others.

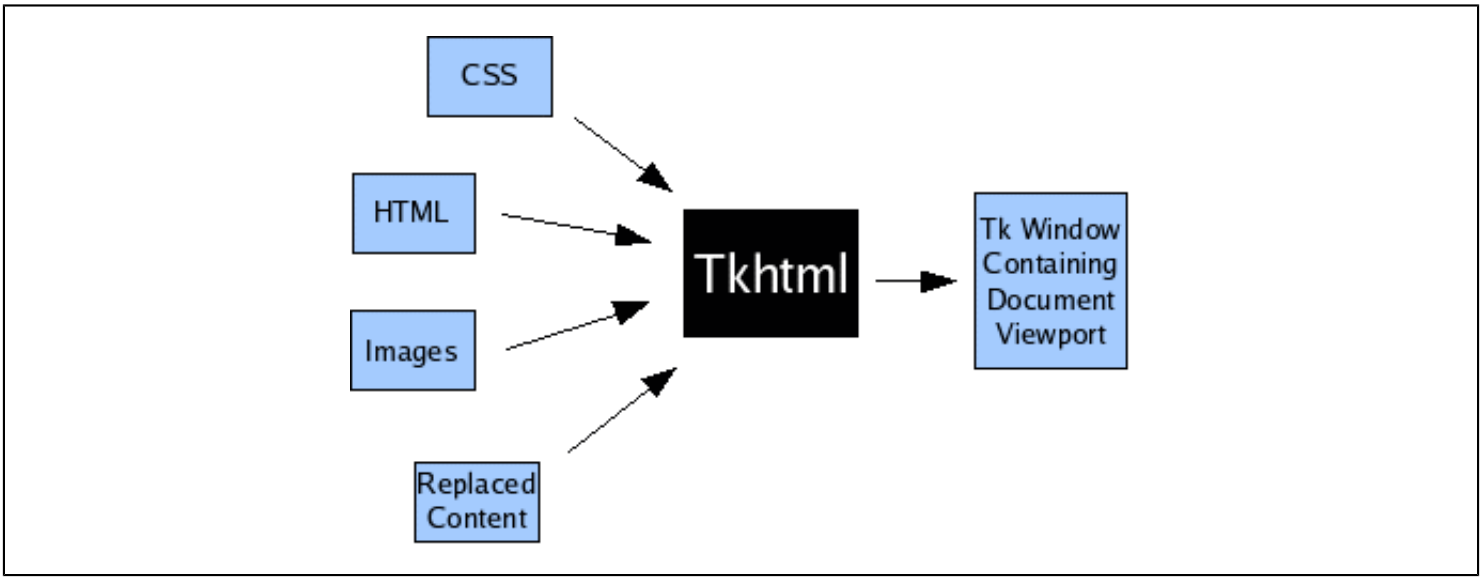


Figure 1 - Role of Tkhtml

All existing HTML renderers for Tcl/Tk have one thing in common: A significant subset (possibly even a majority) of documents found on todays Internet are not rendered as the author intended. This is not due to major technical flaws in existing packages, but simply because the web has moved on from pure HTML. In 2006, most web documents use a combination of HTML and Cascading Style Sheets (CSS) [6] to define the

presentation of documents. The requirement to support CSS is the reason Tkhtml 3 was created instead of channeling effort into improving Tkhtml 2 or another existing package. In many ways, it now seems that "TkCSS" would have been a more descriptive name than "Tkhtml".

Figure 1 roughly depicts the functionality provided by the Tkhtml 3 package. The application configures a Tkhtml widget by specifying:

- A single HTML document.
- Zero or more CSS documents.
- Zero or more images. Images are passed from the application to Tkhtml using Tk image handles.
- Zero or more instances of "replaced content". Replaced content is a concept defined by CSS to mean boxes of content generated by external means that are mapped into the document viewport. Examples of replaced content in an HTML document include form controls, the Tcl browser Plugin and embedded video frames.

The widget displays a scrollable viewport that may be used to view the rendered document. Various introspection interfaces to query the document tree, CSS property values and document layout are also available.

Introduction To HTML/CSS

The role of HTML has changed slightly since it's inception. Last century, HTML was used to define both the content and presentation of documents for display. More recently, the task of specifying presentation has been abrogated to CSS and HTML constructs solely concerned with presentation have been deprecated by the World Wide Web Consortium (WC3) [7]. Content and structure are still specified using HTML. Of course, many documents do not conform to these guidelines and so any HTML widget is required to support both CSS and the older, deprecated HTML presentational attributes.

As of version 2.1, CSS consists of 108 separate properties, around 90 of which are applicable to a visual display agent like Tkhtml (other properties concern aural or fixed-page document renderings). Based on rules specified in various CSS documents (including the default stylesheet, see below), each document element is assigned a value for each applicable property. Rules may specify that certain property values be assigned to elements based on their type, attributes or their position in the document tree relative to other elements, The values of the assigned properties determine the presentation of the document element. For example:

- The **'display'** property determines whether the element generates a block box (similar to the default behavior of a <p> element), an inline box (similar to the default behavior of an <i> element), or even a table box (similar to the default behavior of a <table>).
- The **'float'** property determines whether the element is added to the normal document flow or "floated" to the left or right of the document (similar to an <img> element with the "align" attribute set to "left" or "right"). Text wraps around floated boxes.
- The **'color'** property determines the color of text contained by the element.

Internally in Tkhtml, document layout is determined entirely by CSS properties. The section entitled "The Default Stylesheet" below clarifies the approach to handling legacy document that use deprecated HTML presentation attributes.

Project Motivations and Goals

Despite lack of recognition as such, Tcl/Tk remains one of the most portable and complete application platforms available today. Complex graphical applications move seamlessly between embedded devices and many different personal computer configurations. However, it is not currently possible to create a web browser application. The primary purpose of Tkhtml is to facilitate the development of web browser applications using Tcl/Tk.

The above paragraph brazenly states that it is not possible to create a web browser application using Tcl/Tk, which is obviously not a true statement, if only because the introduction of this paper refers to a "fully-fledged web browser", browsex. A casual inspection of browsex reveals a stable cross-platform application, with similar functionality to all but the latest generation of popular web browsers [20,21] Performance is an order of magnitude better than that of popular browsers, both in terms of initialization time and resource consumption. Despite this, browsex has been all but abandoned (no updates since January 2003). The reason for this is simple: because it uses an older generation of HTML widget, a large number of web pages are rendered incorrectly. Tkhtml aims to address this by supporting modern versions of the relevant standards. Therefore:

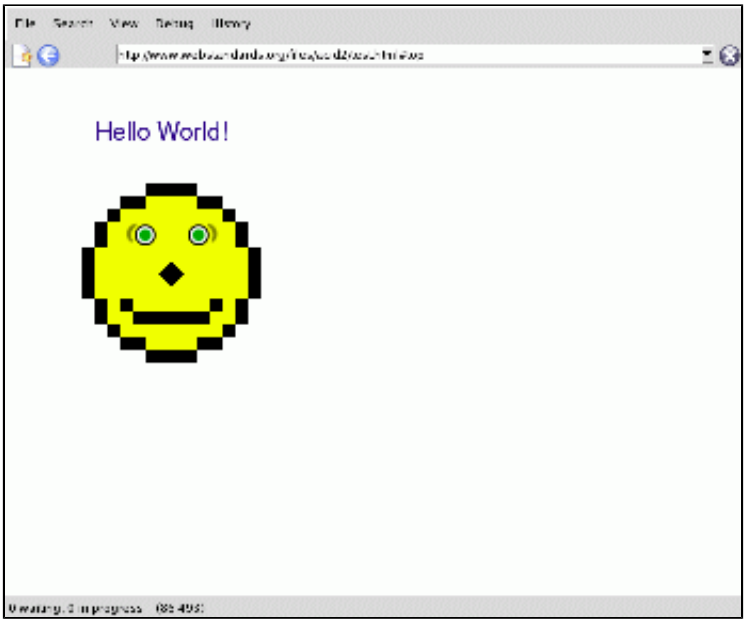


Figure 2 - Shameless gimmickry: the Acid 2 test [8] rendered by a (partially) standards compliant Tcl/Tk web browser [9].

Although targeted at web browser applications, other uses are possible. It is anticipated that applications may use Tkhtml for the following broad categories of purpose. All categories of application are taken into consideration when design or API decisions are taken, although in practice it is fair to say that the other application pattern appear to demand a subset of the functionality of the web browser.

- **Label widget**: Using Tkhtml as a label widget capable of displaying rich text (formatted using HTML).
- **Geometry manager**: Tk application windows where an HTML widget is used in place of a geometry manager. Tkhtml widgets can manage other windows in similar manner to the Tk text and canvas widgets. Some application windows, for example those that resemble forms or "wizard" dialogs, may be more easily created by specifying an HTML document than by using existing methods.
- **Internal document viewer**: Using Tkhtml to view application supplied documents formatted using HTML/CSS, either packaged by the application author (i.e. application help systems) or generated dynamically by the application itself (i.e. reports).
- **Web browser**: Using Tkhtml to create a modern web browser application or component.

Interface Concepts and Design

This section contains a high level conceptual description of the API provided by Tkhtml and in so doing communicates the scope and flexibility of the provided functionality. The goal of this section is not to provide a programming tutorial, but to clarify the role of Tkhtml in the Tcl/Tk software ecosystem.

Along with the application patterns in the preceding sections, the API was designed with the following general principles in mind:

- Tkhtml should provide sufficient functionality and/or sufficiently flexible interfaces to facilitate development of a modern web browser application.
- Tkhtml should include only functionality defined by the relevant portions of the HTML or CSS standards. Functionality not defined by these standards is a matter of policy and therefore must be delegated to the application.
- Whenever standardized web browser functionality can be realistically delegated to the application level, this should be done. This is both to keep the widget as lightweight as possible for applications other than web browsers and for the pragmatic reason that project resources are scarce.

Basics

Using a Tkhtml widget is similar to using any other Tk widget. An instance must be created and handed off to a geometry manager for display. Usually, the widgets requested height and width are set by the -width or -height options in the same way as they are for built-in widgets. The exception is if the boolean -shrink option is set to true. In this case the widgets requested width and height are the size of the currently rendered document. The current widget width, or value of the -width option if the widget is unmapped, is used as the desired width when calculating the layout in this mode.

```
# Create a shrink-wrapped HTML "label"
set html [html $pathName -shrink true]
pack $html

# Create a document frame (default size 800x600)
set html [html $pathName]
pack $html -fill both -expand true
```

Figure 3 - Creating an HTML widget

The global [html] command is currently used to create new widget instances. This may be changed or moved into a namespace just before Tkhtml moves to beta stage.

Note: The example code in figure 3 assumes that the Tcl variable $pathName contains a suitable window name for a new widget. Subsequent example code (figures X, X and X) assume that the Tcl variable $html is the name of an existing HTML widget.

Document Parsing

Before anything can be displayed, the widget must parse a single HTML document and zero or more stylesheets. HTML documents may be incrementally parsed and displayed, but each stylesheet document must be parsed atomically. All documents must be provided to Tkhtml as UTF-8 encoded text.

The example code in figure 4 demonstrates the interfaces used to pass HTML or CSS documents to an HTML widget for parsing. HTML documents may be incrementally parsed by making multiple calls to the [parse] sub-command. However, the final call to the [parse] sub-command should be qualified by the -final option

```
# Parse two HTML document fragments.
$html parse { <html> <body> <p> An abruptly }
$html parse -final { terminated HTML document }

# Parse a CSS stylesheet
$html style -id author-0001 { p {font-size:80%} }
```

```
# Discard all parsed documents and stylesheets
$html reset
```

Figure 4 - Example parsing code

so that the widget knows that the document has been completely parsed. This is so that any open elements may be considered to be implicitly closed and any configure actions (see "Element Handlers" below) taken. The -final option is required even if the caller is sure that the specified document is well-formed. This is because of the esoteric way HTML parsers handle the <html>, <body> and <head> elements: these elements may not be considered completely parsed until the end of the document as defined by the source (i.e. HTTP server) is reached. Once a call to the [parse] subcommand has been made with the -final switch specified, [parse] may not be called again until the [reset] sub-command is invoked. The [reset] sub-command resets the widget to it's initial state, ready to load a new document.

In contrast to HTML documents, CSS documents may not be incrementally parsed, although multiple CSS documents may be loaded concurrently. CSS documents are passed to the widget for parsing using the [style] command. As shown in figure 4, The caller may optionally specify an "id" for the stylesheet by qualifying the [style] command with the -id option. The stylesheet id determines the priority of each stylesheet for use in the CSS "Cascade" algorithm. Essentially, each stylesheet id must begin with the string "author" or "user", indicating whether the stylesheet was provided by the document author or the application end-user. Stylesheets provided by the application user are given a higher priority. If two or more stylesheets are provided by the document author or application user, then the stylesheet with the lexicographically lesser stylesheet id is considered to be of higher priority. A fuller explanation and an example illustrating a method for completely implementing the CSS cascade is provided in the Tkhtml man page [10].

Each time a new stylesheet document or HTML document fragment is passed to the widget the display is updated automatically next time the event loop is entered.

The Default Stylesheet

As well as stylesheets explicitly provided by the user, document layout is governed by the "default stylesheet" document. At any one time there is a single default stylesheet, configured by setting the value of the -defaultstyle option. The default stylesheet defines the display properties that are built-in to the document language. For example, in HTML a <p> element is expected to generate a block level box with vertical margin spacing. The default stylesheet (see figure 5) contains a CSS declaration to make this so.

```
P {display: block; margin: 1em 0;}

IMG[align="left"]  { float:left }
IMG[align="right"] { float:right }
```

Figure 5 - Fragment of a default stylesheet document

As shown in figure 5, the default stylesheet also contains rules for assigning CSS properties based on HTML attributes. The final two rules shown in figure 5 demonstrate mapping between the "align" attribute specified on <img> elements to the CSS 'float' property. As a result, such <img> elements generate floating boxes, as specified by HTML 4.01. For cases where it is not possible to set appropriate CSS properties using standard CSS grammar (or rather, where it would be extremely cumbersome to do so), the default stylesheet may include Tcl code to calculate property values

based on HTML attributes and other considerations. This facility is only available to the default stylesheet, not stylesheets parsed using the widget [style] sub-command. This is for security reasons. If it were not the case, then malicious scripts could be embedded in documents.

Tkhtml includes two built-in default stylesheet documents which the application must manually choose between. One implements HTML "standards mode", the other "quirks mode". Tkhtml does not automatically switch between these based on DOCTYPE tags, but is possible to implement this at the application level. Hv3 (see below) implements Gecko compatible "DOCTYPE sniffing" and selects one of the two default stylesheet documents based on the results.

As of CSS 2.1, almost all presentation rules of HTML may be implemented using CSS. Where this is not the case (primary examples are the "rowspan" and "colspan" attributes of table cells) Tkhtml provides minor extensions to CSS to work around it. It seems likely that these proprietary extensions will be removed and replaced by standard idioms as CSS 3 matures.

In theory, this means that it should be possible to use Tkhtml to display arbitrary XML documents, as long as a default stylesheet is supplied to define the built-in rules of the XML based document language. However this is not currently possible for the following two reasons, both related to the HTML parser inherited from Tkhtml version 2:

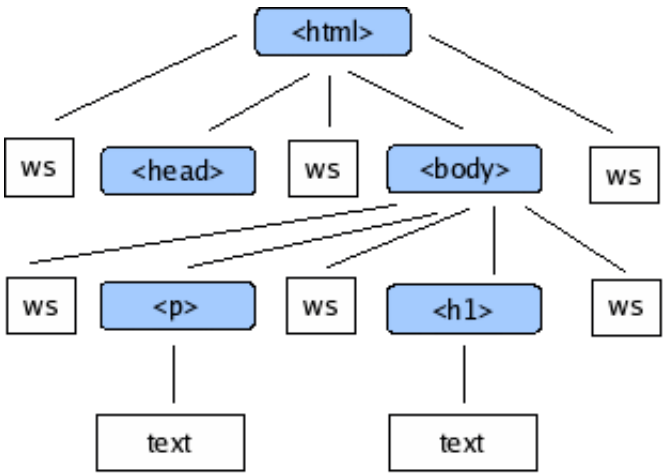
- The parser currently ignores any element types that are not part of HTML 4.01, and
- there is no way to turn off special HTML parsing rules (for example, the rule that one <p> element may not be nested inside another).

It is hoped that these problems, along with minor bugs caused by parsing XHTML using the HTML parser, will be addressed in the near future.

The Document tree

Once an HTML document or part thereof has been parsed, a document tree is generated internally. This tree has one node for each element and one node for each contiguous block of text and/or whitespace. All whitespace present in the original document is also present in the document tree. Figure 6 depicts an HTML document and the corresponding document tree generated by Tkhtml.

```
<html>
  <head></head>
  <body>
    <h1>A heading</h1>
    <p>A paragraph</p>
  </body>
</html>
```



Note: "ws" -> whitespace node.

Figure 6 - HTML Document and Corresponding Document Tree

Tcl scripts access the document tree using node handles (each handle is a Tcl command). Node-handles may be obtained for element, text and whitespace nodes (text and whitespace nodes are really the same thing). Each node-handle provides access to the element type and HTML attributes for an element node, or to the content text of a text or whitespace node. Node-handles also provide an interface to obtain the handles of a nodes parent or children, allowing document tree browsing.

Although this has yet to be implemented at time of writing, eventually node-handles will provide an interface that may be used to programmatically modify the structure of the document tree. Web browser applications require such an interface to implement the Document Object Model (DOM) [11], a standardized interface used by dynamic web pages and AJAX [12] applications.

Querying The Document Tree

Tkhtml provides three ways for a Tcl script to query the widget for node handles:

- The root node is always available.
- By specifying a CSS selector. Tkhtml returns the list of nodes that match the selector. For example, the second example in figure X uses the CSS selector "a[href]" to query for all < a> elements for which an "href" attribute is defined. CSS selectors provide a powerful method for querying the document tree.
- By specifying viewport coordinates. Tkhtml returns the list of node handles that generated displayed content (text, backgrounds, borders or replaced content) visible at the specified coordinates.

Once a single node-handle is obtained, it may be used to browse the tree. For example, an application can perform a linear search of the document by first obtaining the root node and implementing a depth-first traversal algorithm.

```
# Query for the document root:
set node [$html node]

# Loop through all <a&rt; elements
# with the "href" attribute set:
foreach node [$html search {a[href]}] {
  ...
}

# Loop through all nodes that generate
# content currently displayed at the
# coordinates (100,120) of the viewport.
foreach node [$html node 100 120] {
  ...
}
```

Figure 7 - Querying for Node Handles

Supporting the third query mode enumerated above, query by viewport coordinates, allows Tkhtml to elegantly defer much functionality that one might naturally associate with an HTML widget to the application level. For example, Tkhtml itself contains no code to implement document hyper-links. Instead, applications (or mega-widget packages) are expected to take the following approach:

1. Using standard Tk mechanisms (the `[bind]` command), associate a callback script with a button-click on the Tkhtml widget.
2. When a button-click event is detected, query the widget for the document tree node or nodes that generated the displayed content at the coordinates where the button-click event occurred.
3. Check if any of the nodes obtained in step 2 correspond to or are descended from a document tree node that corresponds to hyper-link element. In HTML, hyper-link elements are elements of type <a> with a value specified for the "href" attribute.
4. If step 3 determined that the user has just clicked on a hyper-link, take some application specific action (e.g. load the linked document)

It could be argued that since the majority of applications that use an HTML widget require hyper-link functionality, the procedure described above should be implemented as part of the widget code. However the approach describe above is preferred because it allows the application maximum flexibility in determining exactly which elements are hyper-links, what exactly constitutes activating a hyper-link and the nature of the action taken in response. It is hoped that at some stage a pure Tcl mega-widget based on Tkhtml will emerge that provides pre-packaged versions of this and other commonly required functionality.

Element Handlers

Element handlers provide an alternative to querying for node handles for scripts that perform special processing based on specific types of document elements. Element handlers are Tcl scripts registered with the

widget that are invoked when a particular type of element (i.e. <h1> <p> etc.) is parsed. Three different types of element handlers may be registered. All are invoked by Tkhtml from within calls to the [parse] sub-command. Each element type may have a maximum of one of each of the following (although not all combinations make sense):

- A **parse** handler. Unlike script and node handlers, parse handlers may be associated with both opening (i.e. <form>) and closing tags (i.e. </form>). The specified Tcl script is invoked as soon as an instance of the associated tag is parsed, before any modifications are made to the document tree.
- A **script** handler. Registering a script handler for an element type causes it to be handled differently by the HTML parser. When an opening tag of the specified type is parsed, the parser stops parsing normally and simply scans for a matching closing tag. The text that occurs between the opening and closing tag has no effect on the document tree, but is available for use by the Tcl script registered as the script handler. The Tcl script may specify document text to be inserted and parsed instead of the handled text.
- A **node** handler. A node handler is invoked after modifications are made to the document tree that create a new document node with the specified element type. The script is not invoked until after any child nodes have been parsed and handled (possibly including the invocation of node handlers of their own). This is the most generally useful type of element handler script.

```
<html>
  <head>
    <style>
      h1,h2,h3 { color: green; }
      p { color: red; }
    </style>
  </head>
  <body>

    <table>
      <form action="query.php">
      <tr><td> Search:
          <td> <input name="value" type=text>
    </table>

    <input name="Go" type=submit>
    </form>
  </body>
</html>
```

Figure 8 - Sample HTML document.

Figure 8 contains a malformed but nevertheless fairly typical HTML document. An application parsing the document could register a script handler to intercept the contents of the <style> element and pass them to the widget [style] sub-command. A node handler callback could be registered to detect and perform special processing for the form controls declared by <input> elements. A parse handler could be used to handle the <form> and </form> tags themselves. Although in any reasonable document tree constructed from the example document the second <input> node is not descended from any <form> node, by convention it should be associated with the open <form> tag. Although they may also serve other purposes, parse handlers are primarily to allow applications to make this kind of association.

Image Handling

Tkhtml may include images in rendered documents under three circumstances:

- Background images. An element has a background image if the CSS 'background-image' property is set to a value other than "none".
- As list markers. An element uses an image for a list marker if the CSS 'display' property is set to "list-item" and the 'list-style-image' property is set to a value other than "none".
- When an element is replaced by an image. In HTML documents, this occurs for <img> elements when the "src" attribute is defined.

In the first two cases, the relevant CSS property is set to the URI of the required image. For the third case, Tkhtml defines a custom CSS property, '-tkhtml-replacement-image' which is set to the URI of the elements replacement image, if any. For HTML documents, the default stylesheet (see above) includes a rule that maps the contents of the "src" attribute to the '-tkhtml-replacement-image' property for <img> elements. This rule is shown in figure 9.

```
IMG[src] { -tkhtml-replacement-image: attr(src) }
IMG      { -tkhtml-replacement-image: "" }
```

Figure 9 - Fragment of default stylesheet used to handle <img> elements. The "attr(...)" syntax is introduced in current drafts of CSS version 3.

If image support is required, the application must configure the widget -imagecmd option with a Tcl script that the widget invokes each time an image is required. The script is passed the URI value for the required image (i.e the computed value of the 'background-image', 'list-style-image' or '-tkhtml-replacement-image' property) and must return a Tk image to be used for display. If required, the HTML widget may create scaled copies of the image internally, but will never modify the original. The application is free to modify the size or contents of the Tk image at any later time, and the HTML widget display is updated accordingly. This can be used to implement asynchronous or incremental retrieval of image files, or to implement animated images. The URI passed to the -imagecmd script may be either relative or absolute.

If the -imagecmd script is passed a relative URI, then it should be resolved relative to the HTML document. This can lead to ambiguity if a stylesheet obtained from a different location from the HTML document includes a relative URI (in this case the URI should be resolved relative to the location of the stylesheet, not the document). To resolve this, when a script passes a stylesheet to the widget to be parsed using the [style] sub-command, it has the option of specifying a resolver script. If one is specified, then all relative URIs encountered in the stylesheet are passed to the resolver script to be converted to absolute URIs before the -imagecmd script is invoked.

Replacement Objects

Tkhtml allows applications to specify a Tk window to "replace" a nominated document node. When a node is replaced (by a replacement object), the content of the document node and any descendant nodes is not displayed. Instead, the specified Tk window is mapped into the widget display, using the CSS rules for "replaced elements". HTML form controls and Plugins are the two most common examples of replaced elements required by a web browser application.

The full replacement object interface allows the application to specify zero or more of the following for each document node:

- The name of the replacement object. If the specified name begins with a '.' character, then it is assumed to be the name of a Tk window and that window is displayed instead of the node contents. Alternatively, if the specified replacement object name does not begin with a '.', then the node content is displayed as normal.
- A Tcl script to be evaluated when the document node is deleted. Currently, this only occurs when the [reset] sub-command is invoked, but this may change in the future (see the section entitled "Current Status and Future Work" below).
- A Tcl script to be evaluated whenever the value of one or more of the node's computed CSS properties

changes value. It is possible to access the node's new properties from within the evaluated script. This allows Tkhtml applications to style mapped widgets based on CSS (for example a web browser may wish to set the foreground color or font of an entry widget based on the computed CSS property values).

Dynamic CSS Effects

CSS defines three dynamic pseudo-classes that may be specified as part of a CSS rule so that the specified properties are only applied when the dynamic condition is true. The three conditions are "hover", "active" and "focus", but CSS does not provide any guidelines as to when each condition can be considered to be true. Most CSS applications consider the "hover" condition to be true for a document node whenever the mouse cursor is positioned over content generated by the node. This can be used to implement so-called rollover effects (for example underlining hyper-link text when the mouse cursor is on top of it).

Because they are not defined by CSS, Tkhtml does not attempt to assign interpretations to the dynamic pseudo-selectors. Instead, an interface is provided so that applications can set and clear dynamic conditions on a per node basis using the node-handle interface. For example, to support dynamic :hover selectors, application code may perform the following steps:

1. Register for a callback when the end-user moves the pointer over the HTML widget using the standard Tk bind command.
2. When the callback is received, clear the "hover" condition on any nodes it is currently set on.
3. Query the document for the leaf element that generated the content under the pointer. Set the "hover" condition on the returned node-handle.
4. Optionally, navigate up the tree to the root element, setting the "hover" condition" on each node. Whether or not setting the "hover" condition on a leaf node implies that it is set on all ancestor nodes is a decision left to the application.

```
bind $html <Motion> "hover $html %x %y"

proc hover {html x y} {

  # Clear the "hover" flag on all nodes on
  # which it is currently set.
  foreach node [$html search :hover] {
    $node dynamic clear hover
  }

  # Set the hover flag on all nodes that
  # generate content at the specified
  # coordinates, and all ancestors of
  # said nodes.
  foreach N [$html node $x $y] {
    for {} {$N != ""} {set N [$N parent]} {
      $N dynamic set hover
    }
  }
}
```

Figure 10 - Implementation of :hover for Tkhtml.

Figure 10 illustrates Tcl code to implement this procedure. Tkhtml automatically updates the display when dynamic conditions are modified.

Tkhtml treats the :link and :visited pseudo-selectors in the same way as :hover, :active and :focus.

Text and Tags

Tkhtml also supports interfaces that an application may use to query and interact with the text displayed by the widget. Essentially three functions are provided:

- An application may specify regions of text within the current document to be tagged in a similar manner as permitted by the built-in Tk text widget. It is then possible to override the appearance of tagged text regions, by specifying a foreground and background color.
- An application may query for a plain-text representation of the current document.
- An application may transform character offsets in the plain text representation of the current document to the equivalent point in the document using the indexing system used to add and remove text tags (and vice versa).

In a web browser application, these interfaces may be used to implement two common functions: selection of text with the pointer device and the "Find in Page..." function used to search for text within the current document. For example, to search for and then highlight instances of a supplied string, an application may follow the following procedure:

1. Obtain a plain-text representation of the current document.
2. Search the plain-text representation for instances of the specified string.
3. Convert the character offsets obtained in the previous step to the indexing system used to add and remove tags.
4. Apply a tag to the regions identified in the previous step and configure it with the desired display properties.

```
# Search for the first instance of
# $string in the document currently
# displayed by Tkhtml widget $html.
# If an instance of $string is found,
# cause it to be displayed in white on
# a black background
proc highlight {html string} {

  set plain [$html text text]

  set offset1 [string first $string $plain]
  if {$offset1 >= 0} {
    set offset2 $offset1
    incr offset2 [string length $string]

    set idx1 [$html text index $offset1]
    set idx2 [$html text index $offset2]

    $html tag add highlight $idx1 $idx2
    $html tag configure -bg black -fg white
  }
}
```

Figure 11 - Function to search for and highlight text in a document displayed by Tkhtml.

Figure 11 contains Tcl code to accomplish the above procedure.

Html Viewer 3

Html Viewer 3 (Hv3) is a simple web browser based on Tkhtml 3.0. Hv3 is developed in pure Tcl and is part of the same source tree as Tkhtml. Especially considering that the goal of the Tkhtml project is not to develop a web-browser, but simply an HTML widget, significant time has been spent on Hv3. This is because Hv3 is required to test Tkhtml. Without a functional web browser, it would be extremely difficult to test the HTML widget, both in terms of correctness and performance. As time passes, it is hoped that Hv3 can attract a user-base that will act as informal testers and generate bug reports for Tkhtml. This, rather than attempting to develop comprehensive regression tests in-house, is the strategy successfully used by other open-source rendering components [13,14].

As a general purpose web-browser, Hv3 has two major shortcomings preventing widespread adoption: it does not support javascript (or any other scripting language), and nor does it support web-browser Plugins

such as Macromedia Flash. However, it does support tabbed browsing, integrated web-search, find-as-you-type, auto-completion in the location entry field and several other modern features. Hv3 supports enough of the HTML forms and HTTP cookies mechanisms to use web-applications like GMail and Yahoo Mail.

Hv3 requires only Tkhtml and Tcl/Tk version 8.4 or greater to run, but can optionally use the following extensions to enhance user experience:

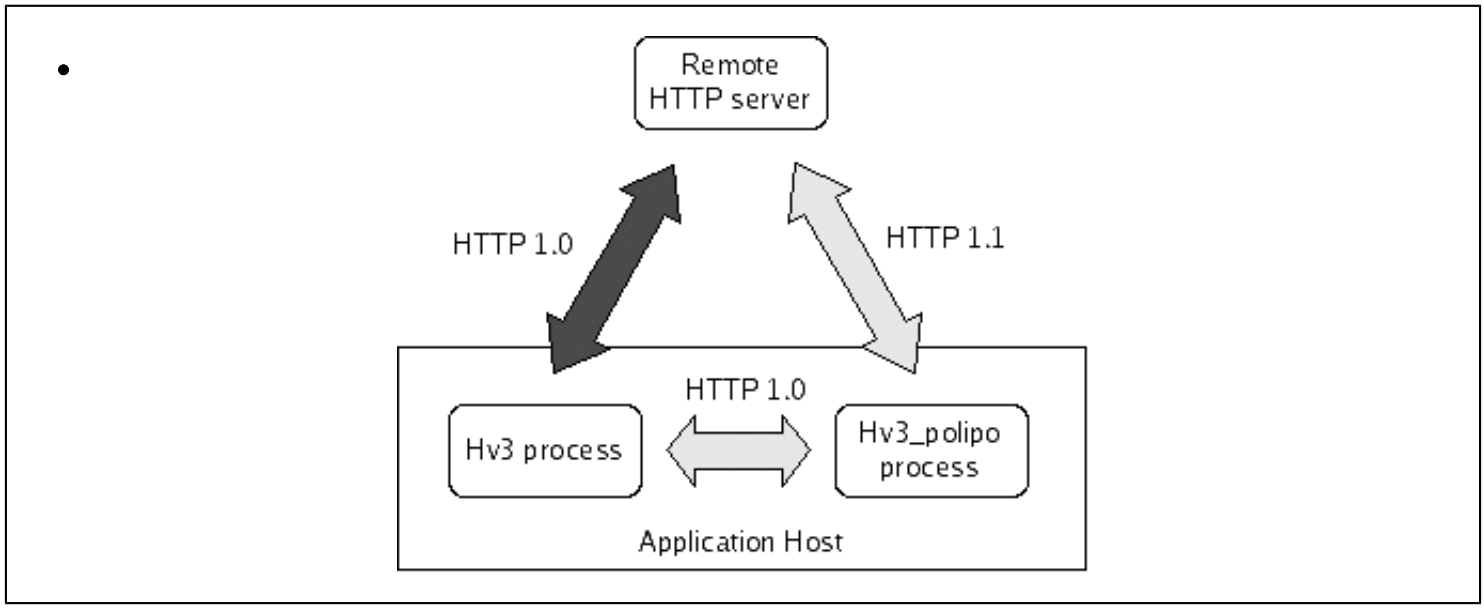


Figure 12 - How Hv3 communicates with remote HTTP servers. If hv3_polipo is available communication takes place using the path indicated with light gray arrows. If hv3_polipo is not available, the path indicated in dark gray is used.

Tls package: The tls package [15] is a Tcl extension that provides Secure Sockets Layer (SSL) capability required for hv3 to support https URIs.

- **Img package**: The Img package [16] is a Tcl extension that provides support for more exotic image types as PNG [17]. Where Img is not available, the Pixane [18] extension can be used instead.
- **hv3_polipo**: hv3_polipo is a slightly modified version of polipo [19], a lightweight standalone HTTP prox When available, each instance of Hv3 starts an instance of hv3_polipo and directs all remote requests throu Both the original polipo and the patches used to create hv3_polipo are available under open-source license compatible with the Tcl core.

The modified HTTP proxy, hv3_polipo, is by far the most important of the three optional extensions. Using hv3_polipo provides the following benefits:

- By itself, hv3 can only use HTTP 1.0 to communicate with remote servers (using the standard Tcl http package). By contrast, hv3_polipo contains a sophisticated and optimized implementation of HTTP 1.1, which significantly reduces latency.
- When using hv3_polipo, Domain Name Server (DNS) queries are performed using asynchronous methods in a separate process from the application GUI. Without hv3_polipo, DNS queries are performed synchronously and in process, causing the GUI to freeze for unacceptable lengths of time.

- hv3_polipo provides an HTTP compatible transparent cache, further reducing the number of network transactions required.

Current Status and Future Work

Current released versions of Tkhtml are marked as "alpha", indicating that features or APIs may still be removed, modified or added. Tkhtml will move to "beta" stage after an API freeze is deemed possible. The two biggest obstacles to this are described below: changes to facilitate support of the DOM and changes to support CSS compatible management of mapped windows. Other APIs may also be extended or refined ,but at this stage seem likely to remain as they currently are.

Tkhtml 3.0 is not yet a complete implementation of CSS 2.1. However, it correctly renders a majority of documents found on the WWW. Correctly, in this case, is defined as identically to other modern web browsers [20,21] with javascript support disabled. Where problems do exist, they are usually due to bugs in the implementation or specification interpretation, not unsupported CSS features. The correctness of the document layout produced by Tkhtml is not currently considered a barrier to moving to "beta" release stage.

Hv3 (see above) provides evidence of the suitability of Tkhtml for developing "scriptless" browsers (browsers that do not support javascript or any other scripting language). Although there is room for improvement, widget performance in terms of speed and memory consumption is competitive with those used by major web browsers. There is no reason to believe that this should not be significantly improved after Tkhtml is moved to beta stage and optimization attempted.

DOM Support

The biggest obstacle to moving Tkhtml into a "beta" stage is that integration with an implementation of the DOM has not yet been properly considered or prototyped. Because some DOM APIs query and manipulate data structures that Tkhtml currently considers internal, Tkhtml is required to provide APIs that provide equivalent flexibility and functionality.

The relevant DOM APIs (DOM level-2 Modules "Core" and "Style") provide web applications with an interface to manipulate and modify the document tree and to explicitly override the computed style of document nodes with externally supplied property values. New document tree nodes may be inserted, and existing nodes may modified or deleted. The HTML widget is required to update the viewport following a modification to the underlying data model. Sometimes this involves recalculating layout, sometimes not. Since a modern web browser is not complete without a standards compliant DOM implementation, Tkhtml must support this efficiently to be considered fit for it's primary purpose.

At time of writing, the performance of the current design under these circumstances is an unknown quantity. This represents the most significant identified risk to the project going forward (that it will take too long to redraw the display after the document tree is modified by a DOM application). Some positive indications can be gleaned from the performance of the features described under "Dynamic CSS Effects" above, but no definite statements made.

The definitions of the APIs described under "Element Handlers" above also require clarification to account for the possibility that the document tree might be constructed and elements of interest added in ways other than by parsing an HTML document.

Managed Window Problems

As described above in the "Replaced Objects" section, Tkhtml allows applications to specify Tk windows to be displayed in place of a specified document node. Difficulties arise in the following situations:

- Sometimes CSS specifies that the mapped window should be baseline aligned. This means the base of the first line of text displayed in the widget (if such a concept is applicable) should be aligned with the current line of text in the browser window. It is not currently possible to query a Tk widget for this information.
- The current API specifies that managed windows must be children of the HTML widget. This means that they are clipped by the HTML window, which is often the required behavior. However, if the the CSS 'overflow' property is set to "auto" or "scrolled" on a document node, then that node establishes a clipping region for it's children. If a node replaced by a mapped window has such an ancestor, Tkhtml attempts to simulate clipping by changing the size of the window via Tk_MoveResizeWindow(). This doesn't always produce the expected results.

The first problem enumerated enough can be solved at the application level by requiring the calling script to specify a pixel offset to be used for baseline alignment. This is a little unsatisfactory, but possible.

The second problem is more difficult to solve. The current solution produces strange results in some circumstances, but given that few web pages expose the problem, may be acceptable in the short term. Alternatively, it may be that applications will be given the option to supply special widgets that support a clipping API as mapped windows to the Tkhtml widget.

References

[1] HTML 4.01 Specification, http://www.w3.org/TR/html4/
[2] Tkhtml version 2, http://www.hwaci.com/sw/tkhtml/index.html
[3] htmllib, http://noucorp.com/tcl/utilities/htmllib/
[4] scrolledhtml - Create and manipulate a scrolled text widget with the capability of displaying HTML documents. http://www.tcl.tk/man/iwidgets3.0/scrolledhtml.n.html
[5] BrowseX Home Page, http://www.browsex.com
[6] Cascading Style Sheets, level 2 revision 1, http://www.w3.org/TR/CSS21/
[7] World Wide Web Consortium, http://www.w3.org/
[8] The Second Acid Test, http://www.webstandards.org/files/acid2/test.html
[9] Html Viewer 3 - Tkhtml test application, http://tkhtml.tcl.tk/hv3.html
[10] tkhtml(n), http://tkhtml.tcl.tk/tkhtml.html
[11] Document Object Model (DOM) Specifications, http://www.w3.org/DOM/DOMTR
[12] Ajax: A New Approach to Web Applications, http://www.adaptivepath.com/publications/essays/archives/000385.php
[13] Mozilla Layout Engine, http://www.mozilla.org/newlayout/
[14] The WebKit Open Source Project http://webkit.org/
[15] Tls SourceForge Project, http://tls.sourceforge.net/
[16] SourceForge.net: tkImg, http://sourceforge.net/projects/tkimg/
[17] Portable Network Graphics (PNG) Specification (Second Edition), http://www.w3.org/TR/PNG/
[18] Pixane, http://www.evolane.com/software/pixane/index.html
[19] Polipo - a caching web proxy, http://www.pps.jussieu.fr/~jch/software/polipo/
[20] Firefox - Rediscover the Web, http://www.mozilla.com/firefox/
[21] Opera browser: Homepage, http://www.opera.com/index.dml